UiO : Department of Informatics
University of Oslo

The Empire Strikes Back

Exploring and Comparing Deep Learning for Image Classification with Other Methods

Edvard Johannesen Bakken

Master's Thesis Spring 2018

Abstract

With an incomprehensibly vast set of data out there and increasingly powerful GPUs to process it, deep learning has been in rapid growth the last years [27]. Image classification is one of the tasks where deep learning has shown great promise, being able to outperform humans on specific tasks [63], and taking all the top spots in the ILSVRC image classification competition [55, 34].

However, deep learning cannot be seen as the silver bullet for all problems as it comes with pitfalls and drawbacks that have to be accounted for [8]. The performance of image classification methods relies on a range of factors; the type and size of the datasets, the measuring metrics, the use case, and so on. There has been done a relatively limited number of large-scale comparisons between Deep Learning and other image-classification methods, and the research that has been done tends to be aged, have presented too few metrics, too few algorithms, or too few datasets to yield statistically significant results [31].

Motivated by all these factors, the applicability, relevance, and need for a large-scale comparison of image classification methods is evident.

Thus, our research question becomes: *How well does Deep Learning perform compared to traditional image-classification methods, and what strengths and weaknesses does it come with?*

To answer this question, we made a well structured and easy to use testing pipeline that enables for easy comparing and testing of automatic image-classification methods. Then, we made an in-depth analysis of the strengths and weaknesses of the different methods and what makes for the performance differences. Lastly, we established a large scale verdict of what methods for automatic image-classification to use given varying types of datasets and training set sizes by different measuring metrics.

We concluded that deep learning was not a silver bullet, and was only preferred if certain criteria were met. Lastly, we proposed a flowchart to help guide the machine learning community in choosing the best machine learning algorithm for a specific image classification task.

Contents

List of Figures

List of Tables

Listings

Acknowledgments

I would like to express my thanks of gratitude to my supervisors Dr. Michael Riegler and Professor Pål Halvorsen for expert guidance and encouragement in carrying out this project. Furthermore, an honorable mention goes to my family and friends for support throughout my studies.

Chapter 1

Introduction

It seems the tech world is all about artificial intelligence (AI) nowadays, and with good reason. AI has been well represented on *MIT Technology Review*'s lists of top ten breakthrough technologies the recent years [50, 51, 52, 53] and has been incorporated into a wide range of everyday technologies. The examples are many, from Snapchat's facial filters to Apple's virtual assistant Siri and Amazon's purchase recommendations.

1.1 Background & Motivation

The AI revolution is happening fast. In fact, Google has gone from having two ongoing AI projects in 2012 to over 1000 four years later [16]. Apple is also setting a statement, shipping all their new phones with separate AI acceleration hardware [2].

With an incomprehensibly vast set of data out there and powerful GPUs to process it, deep learning is the part of AI that has been in rapid growth the last years and has shaped the current AI boom. This is supported by Google's search trend analysis of the term *machine learning*, showing an explosive increase in interest the last years [27].

The task of image classification has been a current research area for a longer period. This has been one of the fields where deep learning algorithms have shown immense potential lately. Furthermore, looking at ILSVRC [55], a popular image classification competition, all the top spots are dominated by deep learning approaches [34].

In this thesis, we will take the field of image classification as an example, where it seems as deep learning is commonly viewed as the go-to method, currently used in a wide range of complex classification tasks. However, deep learning cannot be seen as the silver bullet for all problems as it comes with pitfalls and drawbacks that have to be accounted for [8].

Deep learning consists of complex operations and requires a lot of computational power. Deep learning is also known for their black box approach, meaning that few really understand what happens under the hood, which can be scary considering how hard it is to evaluate a deep learning-algorithm and the impact a dysfunctional system can represent. The ability to analyze and dissect the inner working of a method itself

may be of great value. Also, such a system always has a chance of failing, without being aware of it [41]. Deep learning also requires a lot of training data. This can be a big drawback as training data can be limited, e.g., in the medical world. Another potential disadvantage of deep learning is their difficulty in generating reliable probabilistic results. Probabilistic results can be of great value while analyzing the confidence in a prediction. Other approaches where the decision algorithm is easier to understand gives a much more accurate probabilistic score [58].

The performance of image classification methods can be measured by how accurate it detects images, the setup time, the time it uses to make a classification, the amount of resource use, and many more. Depending on the task at hand, a good classifier is one that optimizes one or more of these performance measures. Better performance increases the usability and aspires more use cases.

The performance of image classification methods relies on a range of factors; the type and size of datasets, the measuring metrics, the use case, and so on. There has been done a relatively limited number of large-scale comparisons between Deep Learning and other image-classification methods, and the research that has been done tends to be aged, have presented too few metrics, too few algorithms or too few datasets to yield statistically significant results [31].

Motivated by all these factors, the applicability, relevance, and need for a large-scale comparison of image classification methods is evident.

1.2 Problem Statement

In this thesis, we will look into the two major groups of methods used for image-classification systems; the traditional handcrafted methods, and the relatively new deep learning based methods. Our goal is to look at a wide variety of method variants and watch their performances on a range of different datasets of varying type, size, and application. This will be used to make a large scale verdict of which methods to use. We will look into the specific methods' strengths and weaknesses and what makes for the performance differences. We especially want to find out when we want to use deep learning methods and when we do not.

Our research question then becomes: ***How well does Deep Learning perform compared to traditional image-classification methods, and what strengths and weaknesses does it come with?***

Main Objective Establish a large scale verdict of what methods for automatic image-classification to use given varying types of datasets and training set sizes by different measuring metrics for various applications. Based on previous research and use cases, we will especially focus on finding the transition from when deep learning works and makes sense, and when it is useless, a waste of resources or too slow.

Sub-objective 1 Make an in-depth analysis of the strengths and weaknesses of the different methods and what makes for the performance differences.

Sub-objective 2 Making a well structured and easy to use testing pipeline that enables for easy further comparing and testing of automatic image-classification methods.

1.3 Scope and Limitations

Based on the research question from Section 1.2, the scope of this thesis is to give a verdict of what image classification methods are best suited for various scenarios.

As each test we will carry out will take anywhere between seconds to tens of hours, our main limitations are the number of tests we are able to run in reasonable time. Limiting the number of tests means limiting the number of datasets, the number of deep learning models, the number of traditional machine learning models and the number of image feature combinations to test. Our goal is totaling the number of tests to no more than a few thousand, accomplishable in a reasonable time.

As our main focus is the comparison of feature extraction classification with deep learning approaches, we will select a diverse list of classifiers from each part. The selected feature extraction classifiers will be: k Nearest Neighbour, Decision Trees, Random Forest, AdaBoost, Linear SVC, SVC Poly 2, SVC RBF, Naive Bayes, Stochastic Gradient Decent, QDA, and Neural Network. The deep learning models we will experiment with are: VGG16, VGG19, NASNetMobile, NASNetLarge, InceptionV3, ResNet50, DenseNet121, DenseNet169, DenseNet201, Xceptron, InceptionResNetV2, and two custom models.

We also decided to limit our focus not to tune method parameters. This is due to two reasons; (1) parameter optimization would further increase the time needed for us to test each method drastically, and (2) parameters are often not optimized due to time, data or computational limitations, thus we will get results representing those of real-world scenarios. We will use default parameter values for all methods.

Although we have tried to choose our datasets as diverse as possible, the reality is that they still represent a relatively small part of classification problems. The results of this thesis are, thus, bound to be application specific to applications relatable to those of our experiments.

1.4 Research method

The ACM Education Board created in 1989 by a task force on the core of computer science, determines and characterizes the structure of how research in computing should be approached. Their report, *Computing As a Discipline* [11], defines computer science in its essence as an intersection between several central processes. The central processes are

applied mathematics, science, and engineering. These central processes are reflected in the paradigms of (1) theory, (2) abstraction and (3) design.

This thesis follows the design paradigm, rooted in engineering, by (1) stating requirements, (2) stating specifications, (3) designing and implementing the system and (4) testing the system.

1.5 Main contributions

The main contributions we have done in this thesis are:

- Presented an overarching view on the fundamentals behind both traditional feature extraction classifiers and newer deep learning classifiers.
- Made an automatic testing pipeline for running and keeping track of a range of image classification experiments.
- Made an in-depth analysis of the strengths and weaknesses of the different image classification methods.
- Come up with a verdict of what image classification methods to use, given a range of problem settings.
- Proposed a simple flowchart for selecting the best image classifier.

1.6 Outline

The following will be the outline of this thesis:

Chapter 2, *Background*: We will look into supporting and foundational theory that we will build upon for the remainder of the thesis, and go through the most relevant papers and research. We will then compare them and discuss how they relate to our research question.

Chapter 3, *Methodology*: We will discuss the main options and considerations for our implementation and go into detail on the design of our testing pipeline.

Chapter 4, *Experiments*: We will present, analyze, and make sense of our results. We will also go into detail about the strengths and weaknesses of the various methods used.

Chapter 5, *Conclusion*: We will look back and summarise our findings. Here we will also see how we stand regarding the research question and goals. We will discuss in which directions, and what to do, to further extend this thesis.

Chapter 2

Background

Humans can easily analyze images and make sense of its content, but this is an arduous task for computers. With significant progress in computer vision the last decade, however, we now have systems able to analyze images better than humans on specific areas [63]. A lot of excellent research gives the foundation for this, and in this chapter, we will dive into and give an overarching view of how this all works. Later in this chapter, we will also look at some comparative studies related to our research question.

2.1 Artificial Intelligence

Artificial intelligence (AI) in the broadest term is a machine capable of mimicking intelligent human behavior. This is the approach we want to take in our quest to make computers able to classify images. AI is a superordinate term covering an entire field of computational methods, thus, the definition is bound to be a little vague.

2.2 Machine Learning

Machine Learning is a subgroup of AI where the programs can progressively improve their performances on a specific task. The different types of machine learning methods can be categorized based on the way the algorithm knows how to improve, or how it learns.

Supervised Learning: These algorithms have training sets with known correct classification labels for each data sample. Supervised learning algorithms use these sets to create a generalized response fitted to the training data.

Suppose we want to make a simple contraption that, given the atmospheric pressure, predicts whether it is going to rain tomorrow. Suppose further that we have been collecting weather information for some time. A supervised learning algorithm learns from this data and produces some weather prediction rule, e.g., it is likely to rain tomorrow if the atmospheric pressure is beneath 1000 hPa.

Unsupervised Learning: With unsupervised learning, the correct classifications to samples are not provided, and the algorithm tries to cluster the inputs solely based on the similarities in the data.

Suppose we run an online shop and want to sell more products. We want to do this by suggesting customers with products they are likely to buy. We assume some customers have similar shopping behavior, so if many customers have bought product A and B, it is a good idea to suggest product B to someone who has product A in their shopping basket. Unsupervised learning algorithms can help with exactly this, by grouping similar customers together based on shopping behavior.

Reinforcement Learning This type is somehow in between the two previously mentioned. The algorithm gets a score on how good it performed on an input, but it does not get any leads on how to improve. The algorithm explores a solution space to get pointers on how to further improve.

Suppose the elevator in our office building is making suboptimal scheduling of its path. We have gotten the task of crafting a new scheduling program. This is a surprisingly complex task, and we do not have information on how the elevator is used. So instead of handcrafting a fixed scheduling routine, we set up a reinforcement learning algorithm that continually tries to learn the optimal control policy. Such a system will continuously update its policy to its environment and can consider, e.g., the time of day.

Evolutionary Learning: Biological evolution algorithms can be looked at as learning algorithms where solutions are represented as creatures in a population. The fittest solutions have the best chance of survival. These algorithms let solutions mate and mutate until the population fitness stagnates.

Suppose we want to solve the traveling salesman problem, that is the problem of choosing the shortest possible route between a given list of cities. This problem has $n!$ possible solutions for n cities, and is, thus, infeasible to solve exhaustively. With evolutionary learning, however, we can generate a population of randomly selected city sequences, which we can mate and mutate, and where the ones with the longest distance have a large chance of being removed. Eventually, the best route sequence stops improving, and we have found a good, if not perfect, route.

For image classification, supervised learning is a natural choice, as we have a set list of classes we want to and also labeled data to work with. The general flow of a supervised learning algorithm for image classification is illustrated in Figure 2.1. We start off with a set of images with known class affiliation. This set is called a *training set*. The assumption we are going to use is that a picture of class a will, on average, be more alike, or have more similar characteristics with other images of class a, than, i.e., images of class b. We will go more in-depth on this in Section 2.4.

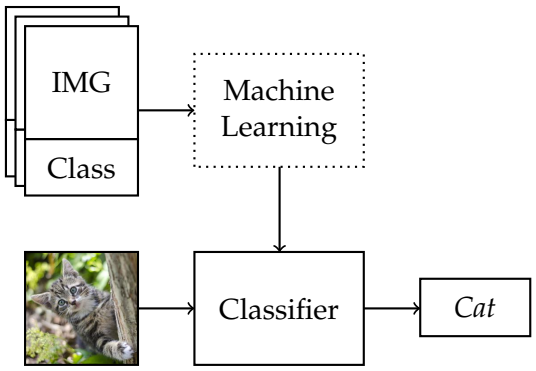


Figure 2.1: Supervised learning for image classification

The machine learning algorithm looks at these images and, depending on the specific algorithm, analyses the data and gives a general classification rule on how to label images. If we now run a new, unseen image through the same classifier, the algorithm will classify the new image according this same rule. If the new images are of a similar distribution, and we have chosen a suitable classification algorithm, we should see good results.

We are going to take the problem of classifying animals as a consistent go-to example. Say we have four classes, namely cats, dogs, frogs, and fish. So, having four folders of pictures, one for each on the categories, the ML algorithm looks through the pictures of, say fish and find commonalities that make them separable from the others. We can imagine the fish images having an abundance of blue, and that this will be the essence of the fish classification rule. New unseen images are now classified as a fish if they consist of much blue.

2.3 Digital images

A computer sees images in a completely different way than us humans do. The way a digital image is stored on a computer is by a 3D-array of size $(w, h, 3)$, where w is the width of the image in pixels, h is the height of the image in pixels and 3 represents the three color values of red, green and blue (RGB). The color values represent the intensity of the different color channels, ranging from 0 to 255, which combines to give mixed colors. See Figure 2.2.

Just looking at the 3D-array values in Figure 2.2b it is challenging to see that this is representing an image of a cat. In fact, it is hard to imagine how any image array of a cat would look like and what separates them from other image arrays. This makes image classification a notoriously difficult task.

For us to be able to use AI, or more precisely, machine learning algorithms on images, and see if pictures are of the same class, we first need a way to compare images. One alternative would be to create some distance measure, $d(a, b)$, that calculates the similarity between image a and b. The

(a) Cat as we see it[1]

$$\begin{bmatrix} (39,48,23) & (34,41,20) & \cdots & (167,166,145) \\ (32,39,16) & (26,31,11) & \cdots & (165,164,142) \\ (24,29,8) & (20,24,4) & \cdots & (162,160,136) \\ (20,23,4) & (21,24,5) & \cdots & (160,156,130) \\ (22,25,6) & (24,27,8) & \cdots & (159,153,124) \\ \vdots & \vdots & \ddots & \vdots \\ (226,211,198) & (226,214,203) & \cdots & (97,93,75) \end{bmatrix}$$

(b) Cat as a computer sees it

Figure 2.2: Different perspectives on images

most straightforward way of doing this may be with the Euclidean measure shown in Equation (2.1).

$$d(a,b) = \sqrt{\sum_{i=1}^{w}\sum_{j=1}^{h}\sum_{k=1}^{3}(a[i,j,k]-b[i,j,k])^2} \tag{2.1}$$

An approach like this does, however, have many weaknesses and pitfalls. Figures 2.3a and 2.3b shows simple image operations on the original image in Figure 2.2a. While we as humans may expect Figures 2.3a and 2.3b to be markedly closer to the original than Figures 2.3c and 2.3d, clearly picturing other motives, we do not see significant differences in distance.

Although we may imagine scenarios where directly comparing pixel values will be a good solution, the complexity of our task makes this approach deficient. Real world cat images different in viewpoint and light conditions; the cats will have different colors and sizes, the background will vary and so on, adding to the difficulty. As our naive method is terrible at handling even small shifts in perspective, we need another way of looking at pictures.

(a) Shifted cat
$d = 4547$

(b) Dark cat
$d = 4742$

(c) Dog[2]
$d = 4561$

(d) Other cat[3]
$d = 4586$

Figure 2.3: Euclidean distances from Figure 2.2a

[3] https://unsplash.com/photos/Qpjl_dXQrD8
[3] https://unsplash.com/photos/HrMD7MngiBE
[3] https://static.pexels.com/photos/14644/pexels-photo-14644.jpeg

2.4 Image features

Instead of looking at the pixel values directly, we can try to extract some properties about the image. *Global image features* represent such properties, giving us some information about the image as a whole. Examples of such features could be a measure of the color distribution or a ratio of the number of edges in a picture. These features are combined to a *feature vector* as an array of numerical values, where each value represent some property of the image. An example is illustrated in Figure 2.4.

Figure 2.4: Feature extractor

The idea is that it is easier to determine an image's class based on these features. Also working with a simple vector of floats in the magnitude of tens or hundreds of values are far more manageable than millions of raw pixel values.

These features can be seen as functions operating on an image. While a simple feature can tell us the overall brightness of an image, it will probably not be able to give us enough information to provide a satisfactory classification alone, though it might work very well in a context of many other features. Global image features can also be very complex. We might have a feature telling us the amount of fur in an image. Such a feature might be able to give an excellent classification score on separating, for example, cats from frogs on its own.

A simple example of how we can calculate image features can be seen in Figure 2.5, where we have created three color histograms, from each of the color channels. The five values from each of these combine to create a full feature vector of length 15.

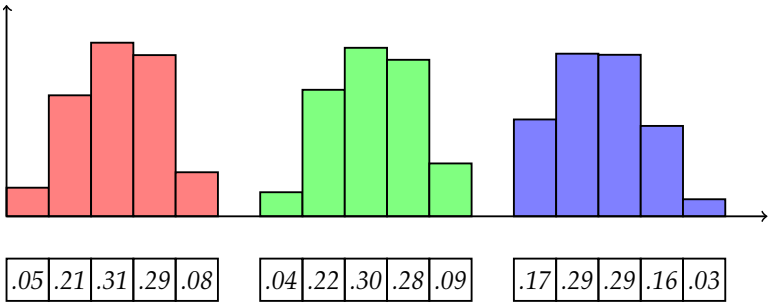


Figure 2.5: Color distribution of cat image from Figure 2.2a

Combining a global feature extractor with the generalized image classification system with supervised learning from Figure 2.1 we get the more extensive system illustrated in Figure 2.6. This is how traditional image classification is done.

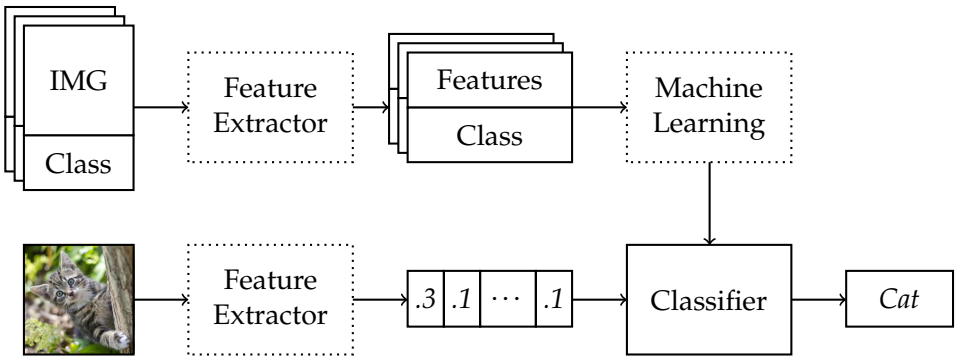
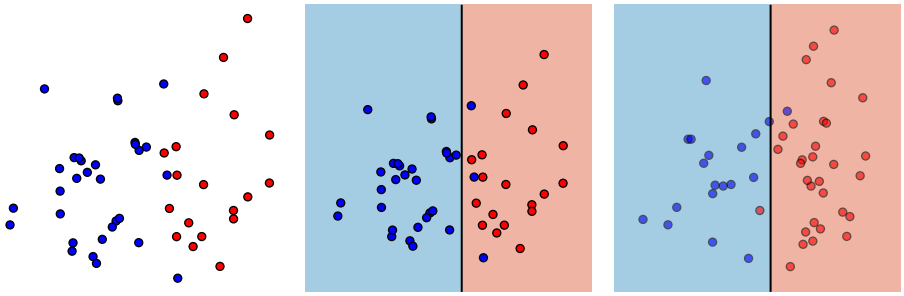


Figure 2.6: Machine learning flow with global feature extractor

2.5 The Problem Space

Now, we will look at the problem space in which we will work. The most straightforward problem space to illustrate and understand is the one with two features (or variables). Such a problem space is shown in Figure 2.7a. Here, each dot in the figure represents one data point, the red ones belonging to one class, and the blue ones to another. The dots position in the x- and y-dimension, denotes the data point's feature values for feature a and feature b, respectively.



(a) Training set

(b) Training set with decision border

(c) Test set with decision border

Figure 2.7: Creation of decision border

What we can see from Figure 2.7a is that the blue points are located left of the red points, and thus have lower values for feature a. If we want to make a rule that separates new unseen data, we can generalize our training data by drawing a line in the solution space, roughly separating the two sets, shown in Figure 2.7b. As our line is entirely vertical, our decision rule simply is to classify a data point to the red class if the feature value a is over some threshold. This line is called the *decision boundary*.

In Figure 2.7c, we have applied the decision rule we found earlier to classify new data. The colors of the points correspond to which class they should be classified to, but we, of course, do not know this when classifying. We can see that our simple classifier performs well, only miss-labeling two of the samples.

Throughout this thesis, we will continue to represent training data with solid strong colors and test data with soft colors.

2.6 Overfitting

An important ML term we need to introduce is *overfitting*. To quote Oxford dictionaries; "The production of an analysis which corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably" [48].

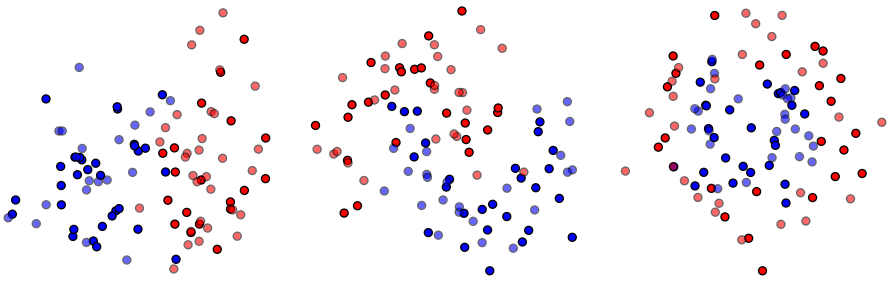
So, overfitting is when our classifier is not looking for the general, but the specifics of our training data. Overfitting can occur when we have too little data, train for too long, allows for too complex decision rules, or have a too parametrized classifier.

2.7 Machine Learning Algorithms

Now we will look at the various algorithms that create these decision rules. First, we make the following distinguishing:

Lazy learning: This type of machine learning algorithms, also called *instance-based* or *memory-based*, are algorithms that do not make generalizations of the data, but rather compare new samples with samples from the training set.

Eager learning: This type of algorithms, given a training set, creates a classification model before looking at the test data.



(a) Linearly separable dataset

(b) Dataset grouped in the shape of moons

(c) Dataset with one class within the other

Figure 2.8: Datasets used to illustrate the classification boundaries

To illustrate how the different algorithms create decision boundaries on some example data, we have three simple datasets to test our classifiers on. This gives insight into how the decision rules look geometrically. The datasets are shown in Figure 2.8.

The resulting decision boundaries after the classification is shown in Figure 2.9. Some of the classifiers have some probabilistic indicator, describing how probable the classification is. The lighter the color, the less probability.

2.7.1 Nearest Neighbour

One of the more straightforward classifiers is the *k Nearest Neighbor* algorithm (kNN) [12]. This is an instance-based classifier, meaning it does not generalize the training set, but instead stores it. The idea is to look for which class is the majority in the surrounding neighborhood. For our use, we will set the neighborhood as the k points in the training set with the shortest distance to the queried point. One could also define the neighborhood as all points within a set radius. This variant is less used and typically perform better for high-dimensional spaces.

In one variant of the algorithm, the influence of each neighboring point is weighted by the inverse of the distance, so that closer points count more. This approach can work better for particular problems. The kNN algorithm can be slow for problems with many samples, as it requires searching through the entire training set for each classification.

The Nearest Neighbor algorithm used on our illustration datasets are shown in Figure 2.9a. We note that the decision boundaries of the kNN algorithm are part wise linear and can form very complex shapes. We see good classification results.

2.7.2 AdaBoost

AdaBoost [17], short for *adaptive boosting* is a classifier that combines a sequence of other classifiers on data continuously modified. It starts by fitting a classifier on the training set. The samples wrongly classified then gets weighted heavier (boosted), making so the next classifier in the sequence "tries harder" to classify them correctly. The new final classification is a majority, or weighting, of the series of classifiers. As outliers continuously are being wrongly classified, these samples get more and more influence, and the classifier is prone to overfit.

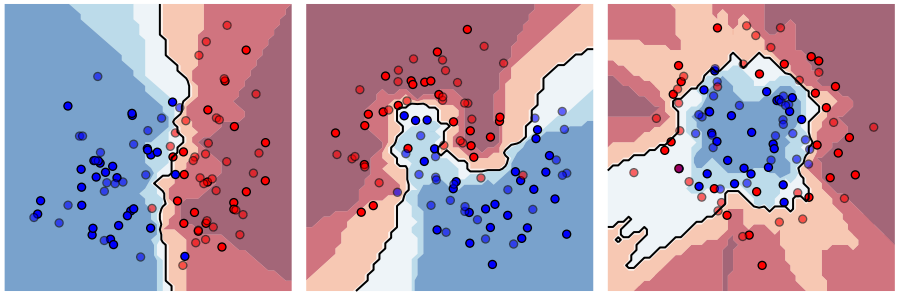
The decision boundaries of an AdaBoost classifier can be found in Figure 2.9b. Here we see a fairly part-wise linear decision boundary, though, with relatively complex decision rules for the two last datasets.

2.7.3 Decision Tree

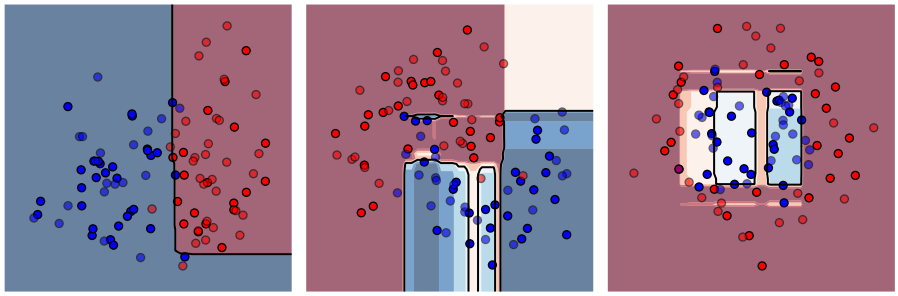
Decision Tree (DT) [5] is a non-parametric classifier consisting of simple decision rules. The decision rules are simple, "if-like", statements that are added in a tree structure. The end node in the tree denotes of which class it should be part.

DTs are simple, fast, and flexible. However, they can overfit the data by making overly complicated trees. Also, decision trees are not able to represent specific concepts (geometries in the solution space) effectively.

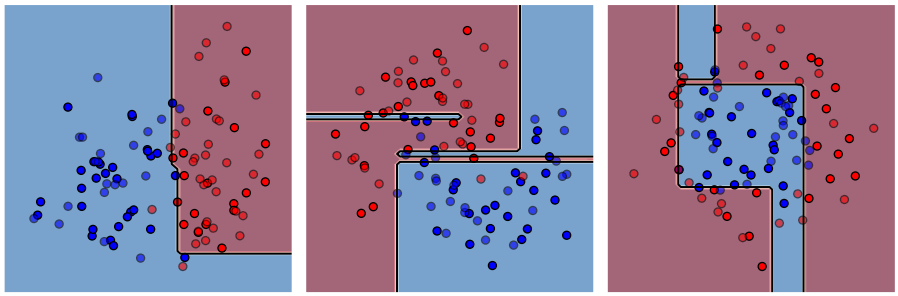
In Figure 2.9c, we can see how the trees end up classifying some sample data. Here we see an part-wise linear classifying boundary that can not be diagonal. This could create trouble, but for out tests it turns out good.



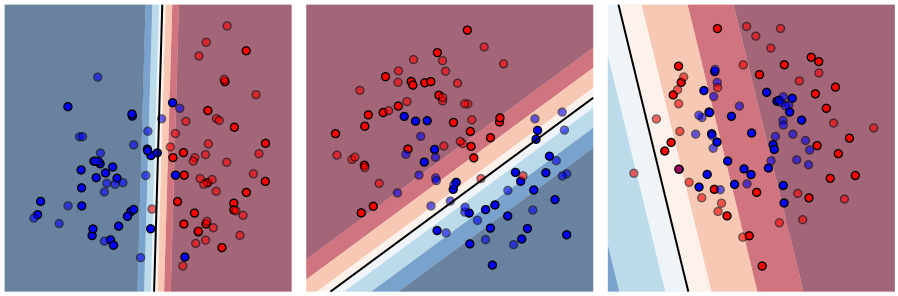
(a) Nearest Neighbour



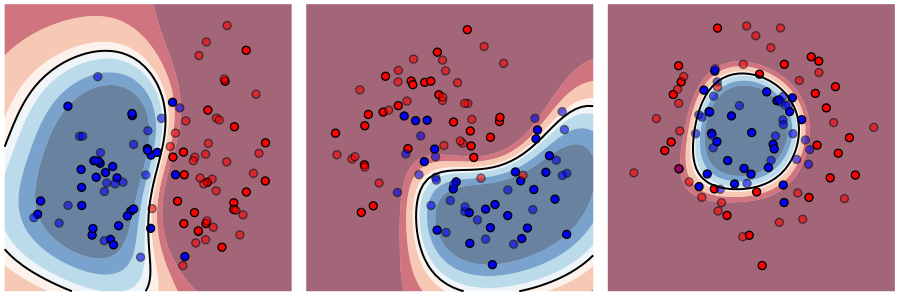
(b) AdaBoost



(c) Decision Tree

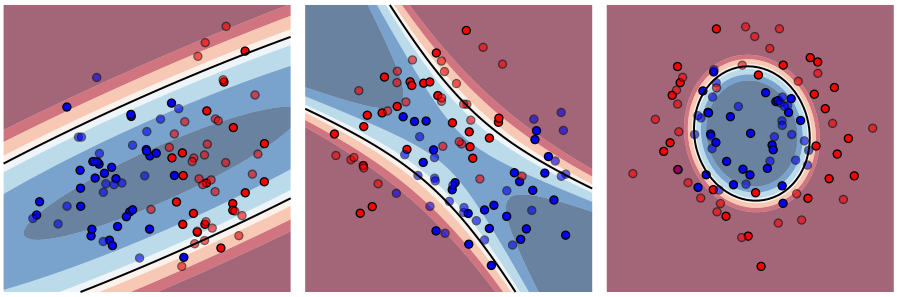


(d) Linear SVC

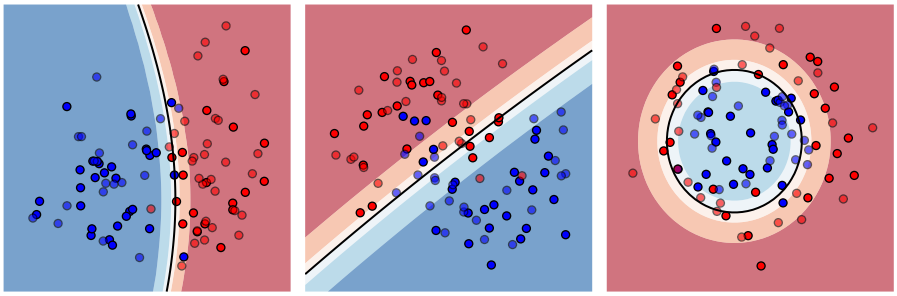


(e) RBF SVC

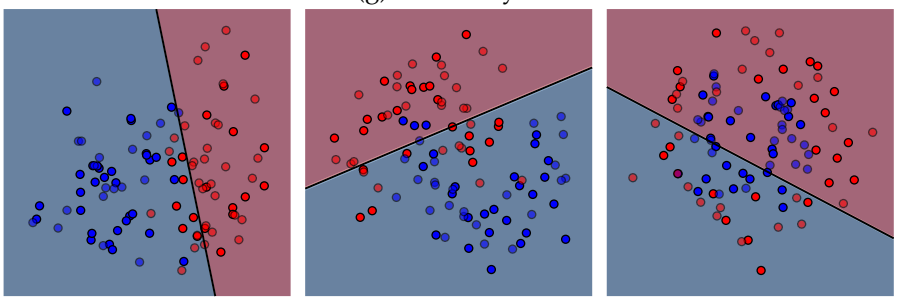
Figure 2.9: The different classifiers on various datasets



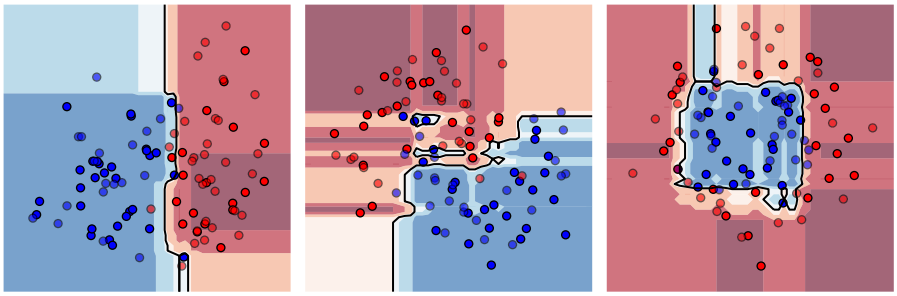
(f) Polynomial SVC



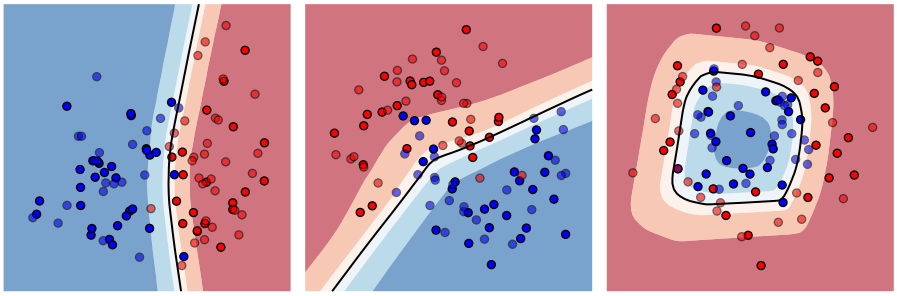
(g) Naive Bayes



(h) SGD classifier

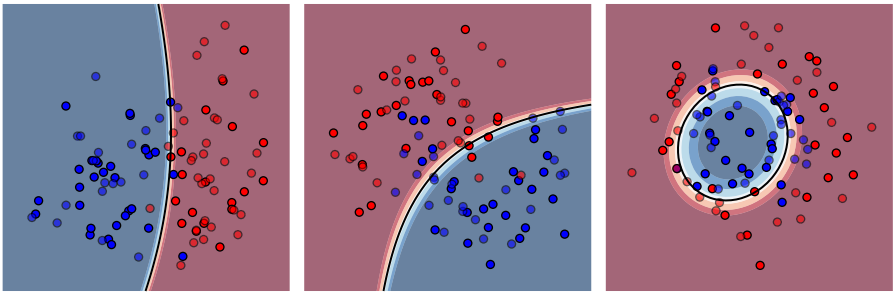


(i) Random Forest



(j) Neural Net

Figure 2.9: The different classifiers on various datasets (cont.)



(k) QDA

Figure 2.9: The different classifiers on various datasets (cont.)

2.7.4 Support Vector Machines

Support Vector Machine (SVM) [57] classifiers start off by us wanting to separate classes in the variable space by a hyperplane, which for two dimensions is a line. We want to choose this line so that it maximizes the distance to the nearest points. However, in real-world scenarios, this gets a bit more complex.

Most datasets have noise and are not perfectly separable, and SVMs thus allows for some outliers. To determine to which degree outliers will affect our classifier, we set a parameter, *regularization*, that specifies how much we want to avoid misclassifying training points. We also have a parameter, *gamma*, that determines how close points need to be to be considered for the calculation of the separation line.

Kernels

Datasets may not be linearly separable, and thus we may use different *kernel*s. The kernel is the similarity measure between two data points. We can also think about the kernel as a mapping to another solution space, and in this new solution space, our separating hyperplane can produce better results. We have different types of SVM kernels:

Linear SVMs are vanilla SVMs, the kernel is merely $\langle x, x' \rangle$, and the resulting decision boundaries must be straight lines as we see in Figure 2.9d. Thus, we see it has trouble with the two last datasets, which not is linearly separable.

RBF (radial-basis function) SVMs are based around kernels on the form $\exp(-\gamma \|\langle x, x' \rangle\|^2)$. Examples of the effect of the RBF kernel can be seen in Figure 2.9e. The RBF kernel makes for the most flexible of the three SVM kernels, while still looking smooth and generalizing.

Polynomial SVMs have kernels on the form $(\gamma \langle x, x' \rangle)^2$. The resulting decision boundaries are ecliptic in shape as seen in Figure 2.9f. We sees it matches perfect with the third dataset, but has great trouble with the other two.

2.7.5 Naive Bayes

The *Naive Bayes* [65] classifier is motivated by Bayesian conditional probability, where we want to maximize the posterior probability, which can be written:

$$P(y|\mathbf{x}) = \frac{P(y)p(\mathbf{x}|y)}{p(\mathbf{x})} \tag{2.2}$$

Assuming the classes independence, we can get an expression for choosing the class with the highest posterior probability:

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} p(x_i|y) \tag{2.3}$$

Then we need to define which data distribution we assume our data to be of. The Gaussian distribution is the most common:

$$p(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \tag{2.4}$$

The resulting classifier, using Equations (2.3) and (2.4), is a maximum likelihood classifier. The naive Bayes classifier makes the independent assumption; that all features are independent, and if they are not, the classification scores will be poor.

The resulting decision boundaries can be seen in Figure 2.9g and are of ecliptic shapes. The gaussian distribution assumption creates trouble in classifying the second dataset of an unorthodox moon-distribution.

2.7.6 SGD Classifier

The *Stochastic Gradient Descent* (SGD) [66] classifier is a continuation of the gradient descent method. This method starts with a random initial value and iteratively updates the values according to the gradient of the loss function (the function describing the "cost" associated with the current solution).

However, for massive datasets, this loss function can be infeasible to calculate. So instead of deriving the loss for the whole dataset, the stochastic gradient descent method selects a subset of the dataset to estimate the loss of, and update the current solution thereafter. Then a new random subset is selected, and the process repeats.

The SGD classifier is a simple and efficient approach but has many hyperparameters that must be set. Example of the solutions given can be found in Figure 2.9h. The decision boundaries are bound to be linear, thus, it performs especially bad on the third dataset.

2.7.7 Random Forest

Random forests [6] are a continuation of the decision tree algorithm. But, instead of creating one tree, it creates multiple (a forest), based on different features. When classifying an unseen element, one classifies it with all of

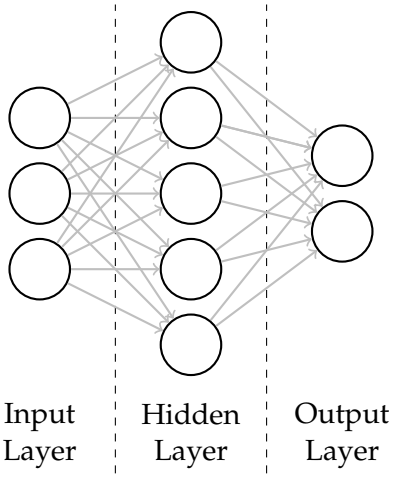


Figure 2.10: Neural network

the trees and adds up the votes from each predicted target. The target with the most votes is the final prediction by the classifier.

The random forest is still very fast, despite requiring more computation than decision trees. The decision boundaries of random forest are quite similar to those of the decision trees, but have a more complex shape, as we can see in Figure 2.9i. All in all one of the best classifiers on the test set.

2.7.8 QDA

QDA (Quadratic Discriminant Analysis) [18] is quite similar to the naive Bayes classifier. Here we used Bayes rule, introduced in Equation (2.2), and choose the class with the highest posterior probability, as in Equation (2.3).

In the probabilistic models for QDA, however, each class is modeled as a multivariate Gaussian distribution with no need for the independence assumption:

$$p(\mathbf{x}|y) = \frac{1}{(2\pi)^n|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu_k)^t\Sigma_k^{-1}(\mathbf{x}-\mu_k)\right) \tag{2.5}$$

The classifier, thus, needs to calculate the class priors, the class means and the covariance matrices. We note that we need two samples to calculate the covariance.

2.7.9 Neural Networks

Neural networks (NN) [22], here used interchangeably with *Multi-Layer Perceptrons* (MLP)[4], are machine learning algorithms with inspiration from neuroscience - the study of the nervous system. A neural net consists of a set of layers, where each layer again comprises a set of nodes. Each node in a layer is connected to all nodes in the following layer, where each such interconnection has a weight. This is illustrated in Figure 2.10.

[4]MLP is a class of NNs with three or more layers of nodes

The first layer is called the *input layer* and is linked to the input values of the sample to be classified. The group of layers ranging from the second to the second to last is called the *hidden layer(s)*. Each node in this layer is computed as the sum of the values of the connected nodes times the corresponding weights and then put into an activation function. The activation function skews the values and removes the linearity, allowing more complex decision boundaries.

The last layer is the *output layer*, which is computed in the same way as the hidden layers. Each node in the output layer typically represents one class each, and the node value is the "score" for that class. The node in the output layer with the highest value is the one that is chosen as the predicted class for the given input.

Training an NN corresponds to altering the interconnection weights. A common NN training algorithm is the back-propagation algorithm. This is an algorithm that, given a sequence of input vectors, works it's way back from the output layer, altering the interconnecting weights so that the correct class gets a higher score, and vice versa. Repeating this process enough times, we end up with weights that hopefully are general enough to predict new and unseen data with high probability.

Figure 2.9j shows the decision boundaries of NNs with one hidden layer. The NNs are quite flexible, even with only one hidden layer, and we can see good classifications.

Deep Learning

We often hear the term *deep learning*. Deep learning is merely neural networks with more than one hidden layer. NNs with more layers allows for more complex classification rules. This also leads to higher calculation times and possibly classification rules that are less analytically understandable (a black-box approach).

Figure 2.10 shows an example of a NN with one hidden layer.

2.8 Convolutional Neural Networks

With the traditional machine learning approaches out of the way, it is time to dive into the new deep learning approaches, namely convolutional neural networks.

Convolutional Neural Networks (CNN) is an extension of the vanilla MLP NNs from Section 2.7.9. The primary motivation behind CNN's is being able to handle high dimensional input efficient while preserving relational information about neighboring nodes.

This makes CNN's able to work with images directly and remove the detour in calculating image features. Doing this with an MLP would, first of all, require huge weight matrices to fully connect the first layers in the network, and thus require a significant amount of machine resources. Secondly, such large and complex matrices might be too fixed on perfect pixel placement, as we saw earlier in Section 2.3, and the network could

be lousy at handling small variations in the input. We would not get a generalized enough classifier.

The idea behind CNN's is to look for small, simple features in crops of the picture. In the following layers, the simple features will be combined to create more and more complex features, until a classification.

Specifically, to find these simple features, we create smaller matrices, called filters, which we "drag" systematically over the input, and then measure the similarity between the filter and different parts of the data. This operation is called convolution. The output of the convolution will be a matrix representing where in the input filter matched the most. We usually will have multiple filters, so that we can search for numerous features. The output, using various filters, will be a 3D matrix.

One usually has multiple layers of these convolutions, and thus the nodes would start by representing simple shapes and colors, and build to generate increasingly sophisticated features in the input.

These relatively complex features can be comparable to the global features we extracted for the traditional image classification approaches. On top of these features, and thus the last convolutional layer, is a fully connected network, as we saw in the MLP NN's.

The training of the CNN's, backpropagation, works the same way as for MLPs. In addition to changing the in-between weights, we also change the filter weights.

2.8.1 CNN Parameters

Although we will not go into too much more details about the inner workings of CNN's, we will introduce some of the most critical parameters:

Activation function: The activation functions lay between the network layers and, simply put, decides which nodes should be fired, i.e., nodes set close to one.

Loss function: The loss function calculates the similarity between the output and the target, and is what we want to optimize.

Optimization function: This function is used to alter the weights to reduce the error.

Learning rate: This parameter sets how much the weights are altered each iteration, or how fast the network evolves.

Batch size: With large training sets, it is normal to split the training set into multiple batches, and then train each of them in sequence. Batch size is the number of samples in such a subset, or batch.

Epochs: An epoch is one pass of *all* the data. One pass is a learning cycle of prediction, cost calculation, and weight update.

2.9 Data Augmentation

As we discussed in Section 2.3, two images with small differences can be perceived very differently by a computer. We also know the importance of a large dataset for neural network approaches in particular. This enables and motivates the use of data augmentation techniques, where we increase the size of our training data by adding duplicate altered images. The image altering operations can be color adjustments, rotations, shifting, zooming, flipping and many more.

This artificial increase in training data has shown excellent performance increases [62].

2.10 Transfer Learning

Neural networks are known to require large amounts of training data, and thus significant training times, to get good results. The wish to reduce the need for extensive training sets was the motivation behind the process of *transfer learning*. Transfer learning is the technique of taking the weights and parameters from an already trained network, and then alter them to make them fit your task.

Transfer learning has been studied a lot recently, with outstanding results, and benefits such as reduced training time [49, 15, 19, 64]. Yonsinski et al. [64] even conclude that "even features transferred from distant tasks are better than random weight."

2.11 CNN Models

CNN models are defined CNN structures. The last five years we have seen many new state-of-the-art models. In this section, we will briefly introduce some of the more influential ones.

All of these models have been trained and evaluated on the imagenet dataset [14]. This is a huge dataset of 10,000,000 training images depicting over 10,000 object classes.

2.11.1 VGG16 and VGG19

VGG16 and VGG19 were introduced by Simonyan et al. [56] in 2014. The models were in the top regarding performance, and also used less computational power than the competition. Their paper showed how important it was for CNN's to be deep for the hierarchical representation to work best [56].

2.11.2 LeNet

LeNet, by Szegedy et al. [61], was the first model to show how complex and creative layer structures could do better than plain sequential stacking.

It won ILSVRC15 2014 [32] and was the catalyst for more creative and exciting model structures to follow.

2.11.3 ResNet50, ResNet101 and ResNet152

Microsoft presented their ResNet50 in a paper from He et al. [23] in 2016. In the same paper, they also presented ResNet101 and ResNet152. The latter set the record for deepest network and won the ILSVRC in 2015 [55, 33]. The models take advantage of the residual modules.

2.11.4 InceptionV3

InceptionV3, published by Szegedy et al. [60] continued to build upon, and improve, the inception module, which had shown great results in the LeNet. Their top-1 and top-5 errors were 21.2% and 5.6% respectively on the ILSVRC challenge, setting a new state-of-the-art [60].

2.11.5 Xception

Xception was published by Chollet et al. [10] and was Googles state-of-the-art model from 2016. It builds upon the inception module from Inception V3 and achieves both better accuracy and improved speed.

2.11.6 MobileNet

In 2017 Howard et al. [24] showed the MobileNet model. Their approach was a little different in that focused as much on latency as accuracy. Their model had two hyperparameters that set the tradeoff between latency and accuracy, and could yield the right sized model for specific constraints.

When the hyperparameters of the MobileNet model was set to generate equally precise accuracy as other state-of-the-art models, ModelNet was shown to be much faster.

2.11.7 InceptionResNetV2

InceptionResNetV2, yet again by Szegedy et al. [59], combines both inception and residual modules. The model continued to increase ILSVRC challenge results with error scores of 19.6% and 4.7% for the top-1 and top-5 respectively.

2.11.8 NasNetMobile and NasNetLarge

The NasNet models by Zoph et al. [67], are not human-invented architectures, but rather able to learn the model architectures directly on the dataset of interest. And yet again, they set new top scores for the ILSVRC challenge and became the new state-of-the-art.

2.11.9 DenseNet121, DenseNet169 and DenseNet201

The last models we will mention are Huang et al.'s [25] DenseNet models. They proposed direct connections between longer distanced layers and achieved state-of-the-art performances with fewer parameters and less computation.

2.11.10 Summary

The Keras documentation [30] provides a summary of some key features about some of the discussed models. The accuracy given refers to the model's performance on the ImageNet dataset, and the size reefers to how large the pretrained weight file is. The compilation is given in Table 2.1.

Model	Size *MB*	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception [10]	88	0.790	0.945	22,910,480	126
VGG16 [56]	528	0.715	0.901	138,357,544	23
VGG19 [56]	549	0.727	0.910	143,667,240	26
ResNet50 [23]	99	0.759	0.929	25,636,712	168
InceptionV3 [60]	92	0.788	0.944	23,851,784	159
InceptionResNetV2 [59]	215	0.804	0.953	55,873,736	572
MobileNet [24]	17	0.665	0.871	4,253,864	88
DenseNet121 [25]	33	0.745	0.918	8,062,504	121
DenseNet169 [25]	57	0.759	0.928	14,307,880	169
DenseNet201 [25]	80	0.770	0.933	20,242,984	201

Table 2.1: A summary of some key model information [30]

2.12 Learning algorithms for classification: A comparison on handwritten digit recognition

LeCun et al. [35] were the ones to introduce Convolutional Neural Networks back in 1989 with the goal of handwritten digit recognition. A few years later, and multiple convolutional neural network models later, they published a comparative study on some of the better classifiers of the day [36]. The study included a Baseline Linear Classifier, Nearest Neighbor Classifier, Fully Connected Multi-Layer Neural Network, Tangent Distance Classifier, Optimal Margin Classifier and multiple versions of their LeNet CNN.

It is worth noting that they did not compute any image features for the algorithms to make use of. The Linear Classifier and the kNN classifier, for instance, would operate directly on pixel values.

Some of their results are shown in Table 2.2, depicting the CNN's, and especially Boosted LeNet 4, to have the overall lowest error rate. However, taking training time, recognition time, and memory consumption into account the conclusion is not as obvious.

Classifier	Test Error %	Training Time *days*	Recognition Time *s*
MLP (400-300-10)	1.6	7	0.1
LeNet 5	0.9	20	0.04
Boosted LeNet 4	0.7	35	0.5
kNN	2.4	0	1
Tangent Distance	1.1	0	2
Soft Margin	1.1	10	2

Table 2.2: A few selected results from LeCun et al.'s [36] paper

The image classification problem was, and still is, a research area in rapid development with the constant increase in computational power, more substantial and better training sets and better recognizers. Due to the massive memory consumption of the non-NN classifiers, the bottom line was thus that the neural nets' advantage was predicted to become more striking as datasets continue to increase in size.

The paper is now 23 years old, making its relevance for today's problems limited, with a lot of new algorithms being crafted and a high increase in computational power over the last two decades. It's also evaluated on just one dataset and doesn't take advantage of separately computed image features making it suboptimal for our scope.

2.13 StatLog: Comparison of Classification Algorithms onLarge Real-World Problems

StatLog is a very comprehensive study comparing 17 algorithms on 12 datasets [31]. They tested two metrics, accuracy and cost, and recorded both training and recognition times. The study was the most comprehensive and well-known study at the time, but being from 1995 in has limited relevance today.

They come up with these specific conclusions:

- If the data has an extreme distribution, symbolic algorithms are a good choice.
- If the data is close to having a (multivariate) normal distribution, and if cost should be minimized, the discriminant functions perform well.
- If the attributes are all equally scaled and equally important, nearest neighbor algorithms are a good choice.
- Very low correlation datasets, with few other complications, is the only places Bayes classifiers should be considered.
- On datasets favored by both the symbolic algorithms and the discriminant functions, SMART (smooth multiple additive regression technique) and Neural networks with back-propagation may be chosen.

Both these approaches require considerable machine resources overhead.

King et al.'s [31] major conclusion, however, is: "There is no single *best* algorithm, and it is a case of *horses for courses*. The best algorithm for a particular dataset depends crucially on features of that dataset."

2.14 A Comparison of Generic Machine Learning Algorithms for Image Classification

Another of the few papers comparing multiple machine learning algorithms on multiple datasets for image classification is Marée et al.'s article from 2004 [40]. They compare seven algorithms, including decision trees, random forests, and SVMs. They evaluated the algorithms on four datasets, selected to give a good variety. The datasets they chose had images ranging from 28×28 pixels to 128×128 pixels. One of the datasets, MNIST, was the same as the one used in LeCun's study mentioned in Section 2.12.

They chose not to use image features but use the pixel values directly to get a generic approach. They ran each algorithm with multiple parameters and picked the best one. They argue the error rate would be slightly underestimated, but since the number of parameters in each method is small, the results wouldn't be compromised for comparative purposes.

None of the tested algorithms uses neural networks. However, Marée et al. do compare their results to other methods evaluated on the same datasets, which of some are neural networks. The data of the methods not tested in this paper gave a benchmark for how state-of-art algorithms could perform but was a bit sparse in the sense of not being tested on all datasets and not having all evaluation metrics.

Classifier	MNIST	ORL	COLI-100	OUTEX
Classical Decision Tree	11.5	29.25	20.8	89.35
Bagging	4.42	9.5	2.24	73.15
Random Forest	3.0	1.25	1.17	66.9
Boosting	2.29	3.75	0.54	69.44
SVM	1.95	1.25	0.44	71.99
LeNet-4	0.7	-	-	-

Table 2.3: A few selected error rates (%) from Marée et al.'s [40] study.

The SVM and extra-trees with sub-windows were the algorithms with the highest performance regarding error rates and was competitive with other state-of-art techniques. Some of the results are given in Table 2.3. Adding training time to the evaluation carried a less clear conclusion, partly because of mostly self-implemented algorithms. The takeaway from the article is how some of the generic techniques are only slightly inferior to state-of-art techniques.

2.15 An Empirical Comparison of Supervised Learning Algorithms Using Different Performance Metrics

Perhaps the best known comprehensive study comparing supervised learning algorithms on this side of the century is Caruana et al.'s study from 2006 [7].

They compare SVMs, neural nets, decision trees, memory-based learning, bagged threes, boosted trees and boosted stumps - each of multiple variants and different parameter settings, totaling to 2000 models. The metrics used are divided into three categories. The threshold metrics are accuracy, F-score and Lift. The ordering/rank metrics are ROC Area, average precision, and precision/recall break-even point. Lastly, the probability metrics are squared error, cross-entropy, and calibration. They tested the algorithms on a total of seven binary classification problems.

2.15.1 Considerations and Design

To allow for averaging of results the performance result was scaled to the range $[0, 1]$ - 0 being the baseline and 1 being the best performing model.

As one would not want to handicap algorithms whose parameters were hard to select, two sets of experiments were conducted. In one a validation set of 1000 elements was used to find the best performing model from each algorithm, and then calculate this models performance on the larger test set. The second set of experiments was conducted without a validation set. Each model was evaluated on the test set, and the best performing one was reported. The results from the first set of experiments represent more realistic performances but might not convey the potential of the algorithm - if the model was selected more carefully.

The difference in performances between the two experiments turned out to be small and qualitatively similar. The differences would also be lower if a more sophisticated selection procedure were used. The second variation was thus the method used for the rest of the paper.

2.15.2 Results

The best performing models overall was neural nets, SVMs, and bagged trees. Averaging over all nine metrics neural nets was the highest scoring and found to be an excellent general purpose learning method. SVMs are known to perform well in very high dimensional spaces, which none of the tested datasets was. Nevertheless, SVMs was one of the top performers. Bagged trees yielded performances close below neural nets and SVMs. They also appeared to be a safe, general-purpose learning method. Considering their ease of training,they could be a good choice in many cases.

Boosted trees perform very well on the threshold and rank metrics, scoring highest on five out of the six metrics. On the probability metrics,

Model	ACC	FSC	ROC	APR	RMS	MXE	MEAN	SAR
ANN	3.3	**3.1**	3.3	3.7	**2.3**	2.4	**2.93**	2.4
SVM	3.1	**3.1**	3.1	3.0	2.4	**2.0**	**2.93**	**2.1**
BAG-DT	3.6	4.3	3.4	3.3	2.4	2.3	3.22	2.7
BTS-DT	**2.3**	3.3	**2.1**	**2.1**	5.1	5.1	3.41	4.0
KNN	5.0	4.0	4.6	4.9	3.7	3.9	4.34	4.1
BST-STMP	4.9	5.3	4.9	5.0	6.9	6.9	5.49	6.9
DT	5.9	4.9	6.6	6.0	5.1	5.4	5.69	5.7

Table 2.4: Average rankings for each algorithm by a few selected metrics from Caruana et al.'s [7] study.

however, it achieves poorly. Maximum margin methods such as boosted trees were suspected to yield poorly calibrated probabilities. The memory-based methods were not competitive with the top performing algorithms. Neither single decision trees performed well. Here the training set of 4000 elements was suspected to small as the recursive partitioning quickly would run out of data. Boosted trees with weaker stump models did outperform the single trees on 5 out of 6 of the non-probability metrics. But the abysmal performance on probability measures gave a bottom rank.

Looking at the performances by each dataset, each model performs well on some problems and is mediocre on others. The top place for the best performer is split, quite evenly, between neural networks, SVMs, and bagged trees on the seven datasets. But if the probability measures are removed boosted trees now are the overall winner scoring highest on 3 of the datasets.

Generally, the decision trees that performs best uses Bayesian smoothing, the best SVMs uses RBF kernels, and the best neural nets are the ones with large hidden layers.

SVMs used Platt probability calibration throughout the paper. Adding this calibration to the boosted trees the probability estimates massively improves. With Platt, probability calibration boosted trees became the best average performing algorithm over the seven datasets and nine metrics.

2.15.3 Conclusion

The methods performing well on threshold metrics generally also performed well on ranking metrics. The probability metrics, however, was not as correlated. There is variability across datasets and metrics, even though some methods perform distinctively better on average. Even the best models occasionally perform poorly and vice versa.

2.16 Summary

Some of the summarizing information about the papers presented above is compiled into Table 2.5.

Paper	Year	Datasets	Metrics	Algorithms	Train Time	Rec. Time
Lecun et al. [36]	1995	3	1	11	Yes	Yes
King et al. [31]	1995	12	2	17	Yes	Yes
Marée et al. [40]	2004	4	1	7	Yes	No
Caruana et al. [7]	2006	7	9[1]	7[2]	No	No

[1] Not counting average and SAR
[2] Each algorithm ran with multiple parameters

Table 2.5: Compilation of key characteristics from the reviewed papers

Reading through the related work, it is apparent that there is no satisfying answer to our central research question.

- Firstly, the few large-scale comparative studies out there are aged. The set of algorithms tested are missing new and improved algorithms, shown to be very interesting, but are yet to be compared in an extensive, general study.
- Secondly, many studies don't measure more than one metric [13, 31]. Different metrics are better for various applications, e.g., precision and recall are more appropriate in information retrieval [7]. An algorithm can be the top scoring for one metric and not so for another. Specific algorithms are also optimized for different criteria, e.g., SVMs optimizes for accuracy while neural nets optimize for squared error or cross entropy [7].
- Thirdly, looking at the statistical significance between algorithm performances has undeniable value, but this is often not reported. This is an important measure to calculate to what degree the difference between two algorithm performances is due to randomness or if one is genuinely better.
- Fourthly, studies only compare a handful of algorithms often restricted to one or two classes of algorithms [31]. This is of course not optimal given the broad set of different machine learning algorithms that all have their strengths and weaknesses.
- Lastly, most comparative studies have used too few datasets [31] to draw very general conclusions.

Looking at these five points, we can conclude that there is not a general comparison of machine learning algorithms suitable to answer our primary research question. Especially is there a lack of structured comparisons between deep learning algorithms and traditional machine learning methods for image classification.

This is why we think a large-scale comparative study of image classification algorithms would be interesting, both to answer our very

own research question and as a resource to the rapidly growing machine learning community [27].

Chapter 3

Methodology

To answer our research question described in Section 1.2, we are going to follow the steps defined in Section 1.4. We will start off by stating the requirements of our system, stating the specifications of our system, designing and implementing the system and test the system.

3.1 System requirements

We will have the following requirements for our system:

- We want to create a pipeline that runs tests with both traditional handcrafted classifications using global image features and new deep learning classification. This means we want our system to have a range of both traditional and deep learning approaches.
- We want the pipeline to run automatically and take care of all pre-processing for us. The pre-processing consists of two parts; the preparation of our dataset, which we will just need to do once, and the preparation of the image features.
- We want our system to generate the results using a range of scores and metrics. In addition to a set of metrics, we want to automatically track the time used for feature extraction, training, and classification.
- We want our system to save the results for later use with an easy way of retrieval. This means saving the experiments configuration, timing, and metrics to a database.
- We want our pipeline to cache calculations that are computational heavy and repeatedly used. In our case, this corresponds to saving the extracted image features into a library, and later check for already extracted features at each run.
- We want the pipeline to be fully automatic, in that we can initialize multiple experiments which will not need any follow up.

The system will be split into two pipelines. An overview of the flow of the systems can be seen in Figures 3.1 and 3.2.

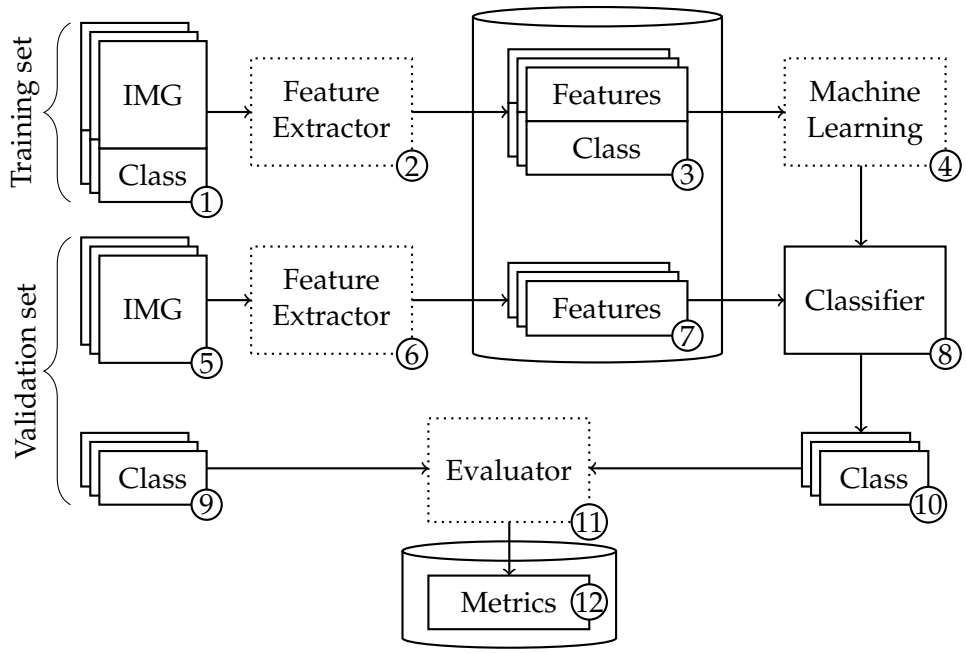


Figure 3.1: Overview of the pipeline for machine learning with feature extraction

3.2 Existing software

There exists a range of tools and software that can help us in our quest to create an automated testing pipeline. We want to take advantage of state-of-the-art tools that currently are being used in real-world applications. This section introduces just such software.

3.2.1 LIRE

LIRE (Lucene Image Retrieval) is a lightweight and easy to use Java library for visual information retrieval systems [38]. The library provides a simple way to retrieve images based on color and texture characteristics. It builds on the well-established text search engine Lucene[1]. LIRE creates a Lucene index of image features for content-based image retrieval using local and global state-of-the-art methods. LIRE builds on successful academic research achievements for content-based image retrieval and makes for a universal foundation agreeing on algorithms and software implementations preventing redevelopment of well-known approaches. It has a range of global and local features implemented from the start for a fast and easy setup process, besides, to be easily expandable with new features.

LIRE will provide us with global image feature extraction that we will use in the traditional machine learning approaches.

[1] `https://lucene.apache.org/core/`

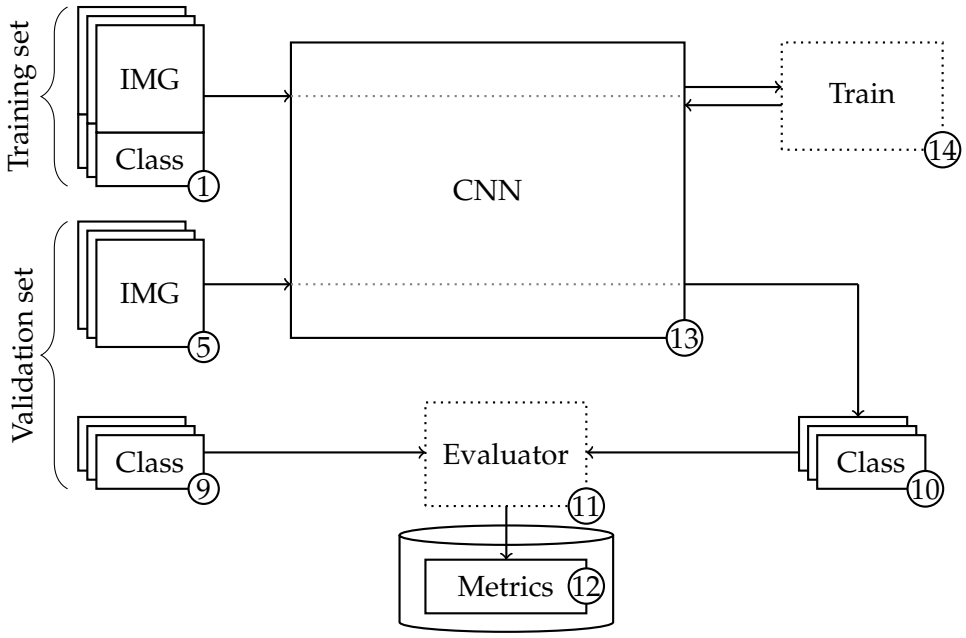


Figure 3.2: Overview of the pipeline for the CNN's

3.2.2 Python for Data Science

Python [54] is a high-level programming language with excellent data science capabilities, libraries and community [44]. Thus, we find it the best choice for our implementation. We will take advantage of Pythons core packages for scientific computing:

- NumPy [43], a fundamental package for numerical computing, defining arrays, matrices and operations on them.
- SciPy [29], a collection of numerical algorithms and domain-specific toolboxes.
- MatPlotLib [26], a popular and multipurpose plotting library for 2D and 3D plotting.

3.2.3 Scikit-learn

Scikit-learn [45] is a Python package, building on the SciPy ecosystem, which includes state-of-the-art implementations of many traditional machine learning algorithms. Scikit-learn presents it's implementations with similar interfaces, making it very easy to extend and test multiple classifiers, with little effort. This is an ideal feature for us, who will need to implement many classifiers.

3.2.4 Tensorflow

Tensorflow [1] is Google's open-source library for data flow-programming across multiple tasks and platforms, including Python. It uses data flow

graphs, where the nodes represent mathematical operations and the graph edges represent multidimensional data arrays that flow between them. It can run in parallel on multiple CPUs and GPUs.

3.2.5 Keras

Keras [9] is a high-level neural networks API, running on top of Tensorflow - among others. It was developed to enable uncomplicated and fast implementations. It also comes out of the box with many CNN models, including weight files, pre-trained on the imagenet [14] dataset.

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense

# prepare datasets
X, y = make_classification(n_features=3, n_classes=2,
    ↳ n_informative=3, n_redundant=0)
X_train, X_test, y_train, y_test =
    ↳ train_test_split(X, y, test_size=.5)

# create model
model = Sequential()
model.add(Dense(5, input_dim=3, init='uniform',
    ↳ activation='relu'))
model.add(Dense(1, init='uniform',
    ↳ activation='sigmoid'))
model.compile(loss='binary_crossentropy',
    ↳ optimizer='adam', metrics=['accuracy'])

# train model
model.fit(X_train, y_train, nb_epoch=250,
    ↳ batch_size=50)

# evaluate model
print(model.evaluate(X_test, y_test)[1])
```

Listing 3.1: Minimal Keras example

The example in Listing 3.1 shows how simple one can set up, train and evaluate a neural network. It starts of by creating a dataset to classify, with three features and two classes, split into a training and a test set. The model is then crated by adding a layer of size five, then, a layer of size one, on top of the input, and compiled with the desired loss, activation and optimizer. Thirdly, the model is then trained for a set number of epochs before we evaluate and print out the result.

The created network is similar to that in Figure 2.10 except for the

output layer which in Listing 3.1 consists of one node instead of two.

3.2.6 Sacred

Sacred[2] is a tool to configure, log, organize, and reproduce computational experiments. With minimal overhead, it provides a simple way to run multiple experiments with different configurations and keep track of the results and corresponding configuration. It can store the runs to both a server, e.g., mongoDB[3], and as files for each run.

```
from sacred import Experiment
experiment = Experiment()

@experiment.config
def default_config():
    classifier_name = 'neural network'
    dataset = 'kvasir'

@experiment.automain
def classifier_exp(_run, classifier_name, dataset)
    return classifier(classifier_name, dataset)
```

Listing 3.2: Simple example of how to use Sacred

Listing 3.2 shows a simple example of how to use sacred. First, we import and initiate sacred. Then we define our variable parameters with a default configuration. Lastly, we call the function doing the experiment, using the parameters from the configuration. When this program is called, the *automain* runs automatically, the configuration values, as well as whatever is returned from the *automain*, are stored. Whatever is printed to the terminal, e.g., errors and warnings, are also stored, and it is possible to manually store results while or after the experiment runs.

```
$ ./classifier_experiment.py with dataset='Holidays'
```

Listing 3.3: Running Sacred experiment from terminal

```
from classifier_experiment import experiment
experiment.run(config_updates={dataset: 'Holidays"})
```

Listing 3.4: Running Sacred experiment from Python

An experiment can then be called either from the terminal, as in Listing 3.3, or from Python, as in Listing 3.4.

[2] https://github.com/IDSIA/sacred
[3] https://www.mongodb.com/

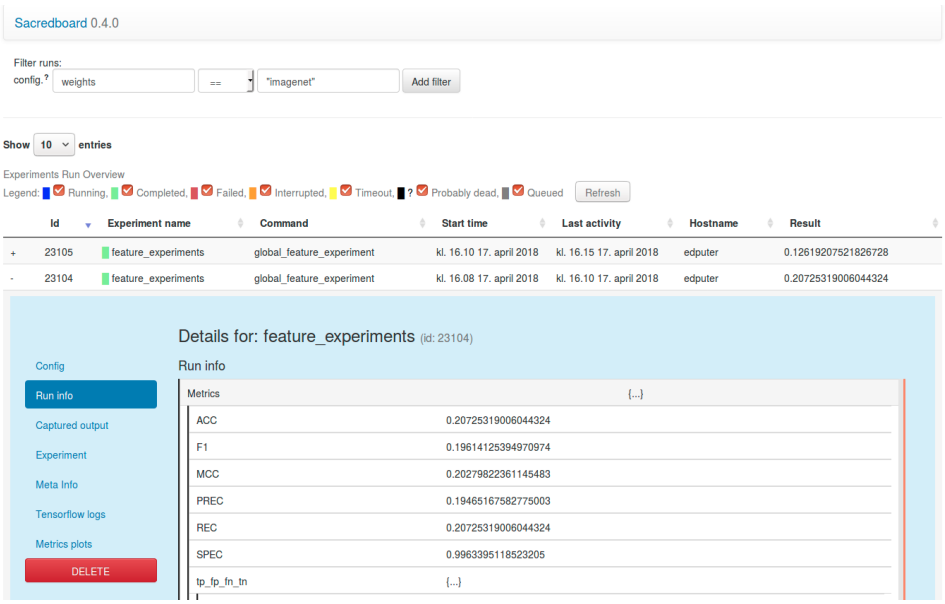


Figure 3.3: The Sacredboard dashboard

		Actual categories	
		True	False
Predicted categories	True	True positive	False positive
	False	False negative	True negative

Table 3.1: The calculation of TP, TN, FP, and FN.

3.2.7 Sacredboard

For keeping track of the experiments, the Sacredboard[4] dashboard is great. It connects to the MongoDB database, used by sacred, and displays all previous and current runs, with their run status, configuration, results, captured output and more. A screenshot picturing it in action is shown in Figure 3.3.

3.3 Metrics

There are a lot of metrics we could have used for evaluating the performance of the classifiers. Here, we will use the suggested metrics given in Pogorelov et al.'s [47] paper about the Kvasir dataset. The relation between TP, TN, FP, and FN are illustrated in Table 3.1.

True positive (TP): The number of samples that were correctly identified by the classifier. Samples that were part of the current category, and was classified as such.

True negative (TN): The number of negative samples that were correctly

[4] `https://github.com/chovanecm/sacredboard`

identified. Samples that were *not* part of the current category, and was classified as such.

False positive (FP): The number of wrongly identified samples. Samples that were classified as part the current category, but was not.

False negative (FN): The number of wrongly identified negative samples. Samples that were *not* classified as part of the current category, but really was.

Recall (REC): Also referred to as *sensitivity*, *true positive rate* or *hit rate* is the fraction of all positive samples that are correctly classified as such.

$$REC = \frac{TP}{\text{number of positives}} = \frac{TP}{TP + FN} \tag{3.1}$$

Precision (PREC): Also referred to as *positive predictive value*, is the ratio of samples we classified as true, that actually was true.

$$PREC = \frac{TP}{\text{number of predicted positives}} = \frac{TP}{TP + FP} \tag{3.2}$$

Specificity (SPEC): Also known as *true negative rate*, is the ratio of negative samples that are correctly classified as such.

$$SPEC = \frac{TN}{\text{number of negatives}} = \frac{TN}{TN + FP} \tag{3.3}$$

Accuracy (ACC): This is the ratio of correctly classified samples.

$$ACC = \frac{TP + TN}{\text{number of samples}} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.4}$$

Matthews correlation coefficient (MCC): A metric that looks at true and false positives and negatives. The metric is a balanced measure and can thus be used when the class sizes are of very different sizes [4].

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{3.5}$$

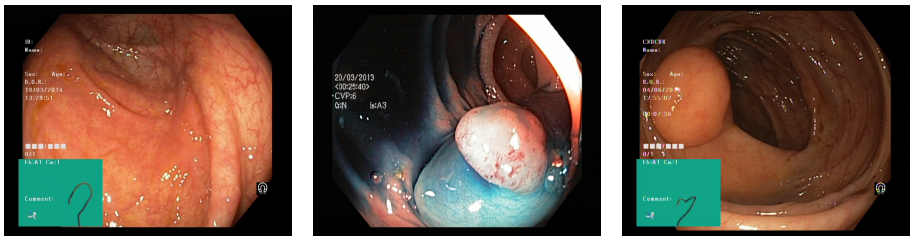
F1 score (F1): Also referred to as *F-score* and *F-measure*, is the harmonic mean between the precision and recall.

$$F1 = 2 \times \frac{PREC \times REC}{PREC + REC} = \frac{2TP}{2TP + FP + FN} \tag{3.6}$$

In addition to the above metrics, we will measure computation times for extraction, training, and prediction.

(a) INRIA Holidays Dataset

(b) UKBench Dataset



(c) Kvasir Dataset

(d) Caltech-256 Dataset

Figure 3.4: Sample images from the Datasets

3.4 Datasets

In this section, we will introduce a handful of datasets. The datasets are selected to be large, both in quantity and size, and as diverse as possible.

3.4.1 INRIA Holidays Dataset

The first dataset we will introduce is the INRIA Holidays Dataset [28]. The dataset consists of high resolution, mostly personal holiday photos in 500 image classes, each of a distinct scene or object. There are 1491 images in total, averaging to 2.98 images per category. The image class sizes vary from two to 13 images per class. Sample images are shown in Figure 3.4a.

The images over 8 mega pixels (MP) in size are scaled down to 8MP, to avoid trouble with the feature extraction.

3.4.2 UKBench Dataset

The UKBench Dataset [42] consists of images of household objects. Each of the 2550 classes comprises four images of the same object from different viewpoints. The dataset totals to 10200 images. Sample images are shown in Figure 3.4b.

3.4.3 Kvasir Dataset

The Kvasir Dataset [46] was created for the purpose of automatic polyp detection in the gastrointestinal tract. It provides images collected by using endoscopic equipment, classified by eight features in the digestive tract and verified by medical doctors. Each class consists of 1000 images, totaling to 8000 images. Sample images are shown in Figure 3.4c.

3.4.4 Caltech-256 Dataset

The Caltech-256 dataset [20] was created by scripts scraping internet search sites, for images containing various keywords. Vision dataset users submitted the object categories, and duplicate images were removed. The dataset contains 29780 pictures divided into 256 categories, averaging to 116 images per class. Sample images are shown in Figure 3.4d.

The images over 8MP in size are scaled down to 8MP. The Caltech-256 dataset originally contained 257 classes, but we have deprecated the 257th class, containing noisy images.

3.4.5 Datasets summary

A summary of the dataset details is shown in Table 3.2. Here we can see that the datasets are quite diverse concerning the number of images, number of classes, number of pictures per category and average image size. In fact, each of the four datasets scores the highest in one of these ranks.

We will note that Holidays and UKBench are datasets where each class contains images of the same exact motive, only from another viewpoint, while Kvasir and Caltech-256 are of different motives.

Dataset	Images	Classes	Img/Class			Size in MP		
			Min	*Avg*	*Max*	*Min*	*Avg*	*Max*
Holidays	1491	500	2	3.0	13	3.1	**4.6**	8.0
UKBench	10200	**2550**	4	4	4	.31	.31	.31
Kvasir	8000	8	1000	**1000**	1000	.33	.80	2.1
Caltech-256	**29780**	256	80	116	800	.0056	.16	8.0

Table 3.2: Comparison of the four datasets

3.5 Implementation

The program we have built is module based and will be described via one file at a time. The overview of how the files are related, is illustrated in Figure 3.5.

The pipeline is avaliable on Github [3].

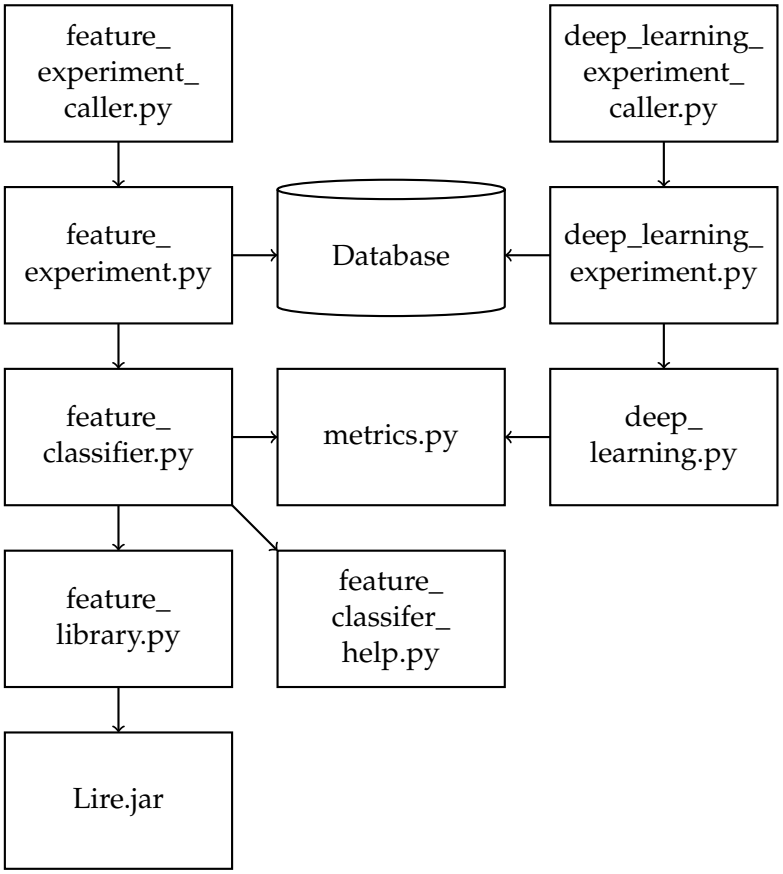


Figure 3.5: Overview of how the different files are connected

3.5.1 Dataset Preprocessing

The datasets all have different properties and format, thus, needs a few pre-processing steps that we will present in the following two scripts.

dataset_restructurer.py

As the datasets we use are structured in different ways, we made a simple program to automatically restructure the datasets to the right format. The format we want to achieve is shown in Figure 3.6. After the datasets are downloaded and unpacked, this script will do the rest.

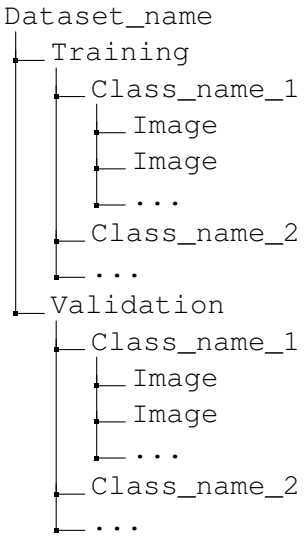


Figure 3.6: Directory structure we want our dataset to be in

dataset_image_resizer.py

Some datasets come with some huge images. These can cause problems when extracting certain features using LIRE. Thus, we have created a script to scale down all images over a certain size. We first used file size as the restrictive measure, but later figured out that the number of pixels was a better measure.

The default value is set to 8 MP as this is large enough not to reduce the amount of information in a picture more than necessary, while causing no problems with our setup.

3.5.2 Feature Extractor

The process of feature extraction is split into two programs; the core feature extractor and a wrapper to efficiently handle the data and save computation time.

Lire.jar

This program acts as the core feature extractor and is based on the LIRE library. The program acts as a simple java program to extract image features, and is packaged in a JAR (Java ARchive) file, making everything into one file, easy to handle. The file contains all Java class files, including all requirements and resources, only a vanilla Java installation is required.

```
>> java -jar Lire.jar image_folder output.arff
   ↳ feature1 feature2 ...
```

Listing 3.5: Running Lire

Feature	Abbr.	Length
AutoColorCorrelogram	ACC	256
BinaryPatternsPyramid	BPP	756
CEDD	CEDD	144
ColorLayout	CL	33
EdgeHistogram	EH	80
FCTH	FCTH	192
FuzzyColorHistogram	FCH	125
FuzzyOpponentHistogram	FOH	576
Gabor	Gabor	60
JCD	JCD	168
JpegCoefficientHistogram	JCH	192
LocalBinaryPatterns	LBP	256
LuminanceLayout	LL	64
OpponentHistogram	OH	64
PHOG	PHOG	630
RotationInvariantLocalBinaryPatterns	RILBP	36
ScalableColor	SC	64
SimpleColorHistogram	SCH	64
Tamura	Tamura	18

Table 3.3: List of features with feature length.

The program is easy to execute from the terminal as shown in Listing 3.5. The first parameter is the path to the lire.jar file. The second parameter determines the input image folder to extract features from, where each sub-folder contains images of the same class. The output is a single file, specified by the third parameter, including all images from all classes. The format of this file is an *.arff* text file with one line for each picture, with file name, class, and the feature values. The rest of the parameters is the set of global features we want to extract.

The available features, and their length, are listed in Table 3.3, along with abbreviations we will use from here on.

feature_extractor.py

The extraction of global image features is an expensive task, as we will see in Section 4.2, and as we want to run many experiments on the same features from the same datasets, it makes a lot of sense to cache, or store, the extracted features for later use. This is the main objective of the *feature_extractor.py* program.

Before we look into how the program operates, we will see how the library stores data. We have chosen the Hierarchical Data Format [21] (HDF) to store and work with library files, as it is well suited for storing large data files, such as multidimensional arrays. We will use a single *.h5* file that acts as a container for all our library files.

We use the program by calling the *prepare_data()*-function, that takes an

input folder path and a list of features as arguments, similarly to *LIRE.jar*. The first step of the program checks whether or not there exist a library file, and if so, which of the features are not in the library. The second step is to make a call to *Lire.jar* with the list of missing features. The output from this call, the *features.arff*-file, is then converted to Numpy [43] format and merged with the files *features* and *classes* in *feature_library.h5*.

The third step of the program is to make a subset of exactly the requested features, out of a potentially much larger library, meaning creating a subset of the *features* and *classes* files, which exists within the library. The reduced files are stored in the library as *features_p* and *classes_p*. This function can also be called with an optional argument, *discard_fraction*, that reduces the size of each class at the same rate, enabling us to see how the reduction in dataset size affects the results.

3.5.3 Metrics

As we will have two separate pipeline systems, we chose to make a common program to calculate the metrics. The metrics we will implement are the same as described in Section 3.3.

metrics.py

The *metrics.py* program is a short program to calculate metrics. It takes two lists as input, one with predicted labels, and one with actual labels, and returns a dictionary with the results.

The *Scikit-learn* Python packaged, described in Section 3.2.3, provides *accuracy*, *precision*, *recall*, *F1-score*, and *Matthews correlation coefficient* metrics, while *Imbalanced-learn* [37] provides the *specificity score*. In addition to the standard metrics, the program also returns the TP, TN, FP, and FN values, making it possible to calculate more metrics after the experiment has concluded.

3.5.4 Feature Classification

The classification part is based on scikit-learn and divided into two parts.

feature_classifier_help.py

The *feature_classifier_help.py* program provides helper programs for *feature_classifier.py* which we will look at below.

The first helper function, *oversample_help*, assists in the cases where we need to over-sample data points from classes with few data points. It takes a list of class names and returns a dictionary of with the number of samples per class, adjusted not to let any class have only one sample.

The second helper functions, *load_datasets_scikit*, will simply load the data from the feature library into four variables, *x_train*, *y_train*, *x_test* and *y_test*, which will be returned.

feature_classifier.py

feature_classifier.py is the core of the system for image classification with global feature extraction. As it is the core of the system, and quite short, we will go through it in its entirety. The code is shown in Listing 3.6.

```
def classify(dataset_path, classifier_name,
    ↪ classifier_params):
    assert classifier_name in classifier_set, "
        ↪ Classifier '{}' not valid".format(
        ↪ classifier_name)
    # Loading dataset data
    x_train, y_train, x_test, y_test =
        ↪ load_datasets_scikit(dataset_path)
    # As QDA requires two or more training-samples per
        ↪  class, we oversample classes with one sample
    if classifier_name == 'QDA':
        x_train, y_train = RandomOverSampler(ratio=
            ↪ oversample_help).fit_sample(x_train,
            ↪ y_train)

    # Classify and predict
    classifier = classifier_set[classifier_name](**
        ↪ classifier_params)
    classifier.fit(x_train, y_train)
    y_pred = classifier.predict(x_test)

    # Calculate metrics
    return metrics(y_test, y_pred)
```

Listing 3.6: The core of the feature classifier

The program takes tree arguments; *dataset_path*, *classifier_name* and *classifier_params*, which all are quite self explanatory. After the assert checking if the classifier name is valid, we load our dataset with the function described above in our helper program. As described in the comments, we then over-sample classes with one data samples if we are using the QDA classifier.

The classifier is then loaded from a *classifier_set* dictionary defined right above in the code, translating the classifier name string to a function. The classifier is then trained with the training data, by calling the classifiers *fit* function. Next, we generate predictions on our validation data by calling the classifiers *predict* function. Lastly, we calculate and return the metrics as described in Section 3.5.3.

The available classifiers, which we introduced earlier in Section 2.7, are:

- Nearest Neighbors
- Linear SVC
- NuSVC
- Gaussian Process
- Decision Tree
- Random Forest
- Neural Net
- AdaBoost
- Naive Bayes
- QDA
- SGDClassifier

3.5.5 Tracking and Running Feature Experiments

To keep track of, run and store results from our experiments, we will make use the Sacred package.

feature_experiments.py

The *feature_experiments.py* program is a Sacred experiment wrapper around our feature classifier. It is similar to the Sacred example we showed in Section 3.2.6, only more complex.

After the imports, we initialize the experiment instance and add our chosen experiment observers. We decided to add both a MongoDB database observer, storing all experiments as database entries, and a file storage observer, saving the experiments to file. The former will be our primary observer, enabling easy tracking with Sacredboard and many possibilities for post-processing of the data. The latter will act mainly as a backup system. Listing 3.7 shows how we start out our wrapper.

```
experiment = Experiment()
experiment.observers.append(MongoObserver.create())
experiment.observers.append(FileStorageObserver.create
    ↪ ('my_runs'))
```

Listing 3.7: Initializing and adding observers to Sacred experiment

Next, we need to define our experiment parameters. Sacred will automatically track these. The code for an example default configuration can be found in Listing 3.8.

```
@experiment.config
def default_config():
    dataset = 'kvasir'
    classifier = 'SVC'
    classifier_detailed = 'SVC Poly 2'
    features = [
        'CEDD',
        'Tamura'
    ]
```

```
    discard_fraction = 0
    parameters = {'kernel': 'poly', 'degree': 2}
    tags = ['Group_4']
```

Listing 3.8: Default config for feature experiments

dataset This is the path to the dataset.

classifier Which classifier from scikit-learn we want to use.

parameters This is a dictionary with parameters that will be sent to the current classifier, e.g., if we wanted our SVC classifier to have a polynomial kernel of the second degree, we would set parameters as in Listing 3.8.

classifier_detailed As we may do multiple experiments with the same classifier, only with different parameters, this configuration entry exist so we can give our classifier a more specific name, and thus make later searches in the database simpler.

features This is a list containing the name of all the global features, from LIRE, that we want to use in our experiment.

tags This is a list of strings for attaching custom labels to our experiments. This is also for later convenience while extracting from the database.

discard_fraction This float determines the fraction of the dataset not to be used. E.g., if we want to see how a smaller dataset affects the results.

The rest of the program mainly calls *prepare_data* and *classify* from *feature_library.py* and *feature_classifier.py* respectively. However there are two additions; (1) the returned scores from the classification is stored as *metrics* in the Sacred experiment, and (2) everything is wrapped in a timing function, using a *line profiler*[5] to keep track of the time spent in the various internal functions. The interesting timing results, extraction, training and prediction times, are stored with sacred.

feature_experiment_caller.py

To loop through different configurations and start all the experiments, we have *feature_experiment_caller.py*. There are not too much exciting going on in this file, but there are two things we will mention; (1) to avoid the whole program stopping if one test crashes, we have put the experiment calls in a *try/except* statement, so the following experiments will continue to run. Secondly (2), to avoid running the same experiment multiple times, we search through the database for identical configurations. An example of such a search is shown in Listing 3.9.

[5] https://github.com/rkern/line_profiler

```
from pymongo import MongoClient
runs = MongoClient().sacred.runs
count = runs.find({
    'config.features': features_sorted,
    'config.dataset': dataset,
    'config.classifier_detailed': classifier_detailed,
    'config.tags': tags
}).count()
```

Listing 3.9: Searching in the MongoDB database

3.5.6 CNN Classification

The other pipeline is the one to run CNN's. The setup of this pipeline is somewhat different, but the results will be of the same format and stored together with the other results. This pipeline is based on the Keras API.

deep_learning.py

The core of the CNN classifier is the *deep_learning.py* program. The implementation and design choices are shaped by making the Keras CNN classifications as easy as possible to perform. The program is created as a class.

Initialization The initialization of the class takes one argument for setting the dataset path, and optional arguments for image size, batch size, model type, what weights to use and more.

The weights can either be set at random or to weights pre-trained on the imagenet dataset if the model supports this. The models that can be selected are:

- Xceptron
- VGG16
- VGG19
- ResNet50
- InceptionV3
- InceptionResNetV2
- MobileNet
- NASNetMobile
- NASNetLarge
- DenseNet121
- DenseNet169
- DenseNet201
- small_net
- large_net

All models, except *small_net* and *large_net*, are models available in Keras, and also have weights pre-trained on the imagenet dataset available. *small_net* and *large_net* are custom models. The preset models was introduced in Section 2.11, while the custom ones will be described later in this section.

The image width and image height arguments are set at initialization. The models, however, require a minimum or absolute size. The initialization process takes care of this and sets the image dimensions to the closest legal value.

Keras data generators are also set automatically, with preprocessing that matches the model. Using these generators, number of classes, and training and validation dataset sizes are also set automatically.

Creating the Model The creation of the model is started with a call on the *create_model()* method, that again calls either *create_preset_model()* or *create_custom_model()*. The code for the latter is given in Listing 3.10.

```
def create_preset_model(self):
    input_shape = (self.image_width, self.image_height
       ↪ , 3)
    input_tensor = Input(shape=input_shape)
    base_model = application[self.model_type](
        weights=self.weights,
        include_top=self.original_top,
        input_tensor=input_tensor,
        input_shape=input_shape)
    if self.original_top:
        predictions = Dense(self.nb_classes,
           ↪ activation='softmax')(base_model.layers
           ↪ [-2].output)
    else:
        x = base_model.output
        x = GlobalAveragePooling2D()(x)
        x = Dense(self.dense_size, activation='relu')(
           ↪ x)
        predictions = Dense(self.nb_classes,
           ↪ activation='softmax')(x)
    self.model = Model(inputs=base_model.input,
       ↪ outputs=predictions)

    if self.weights:
        for layer in base_model.layers:
            layer.trainable = False

    self.model.compile(loss=self.loss,
                       optimizer=self.optimizer,
                       metrics=self.metrics)
```

Listing 3.10: The method for creating preset models

The method of setting up a preset model starts out by creating an input tensor. Then the application is found in the application dictionary, and the weight, input type, and the *include_top* arguments are set. The *include_top* argument determines if the fully connected layers at the end of the model should be included as is, only with an altered number of outputs. We could also not include it, and add a simple custom top instead. We only allow

the top layers to be trained, freezing the rest. Lastly, the model is compiled with the selected loss function, optimizer function, and metrics.

```
def create_custom_model(self):
    self.model = Sequential()

    if self.model_type == 'large_net':
        for i in [32, 64, 128]:
            self.model.add(Convolution2D(i, (3, 3),
                ↳ input_shape=(self.image_width,
                ↳ self.image_height, 3)))
            self.model.add(Activation('relu'))
            self.model.add(Convolution2D(i, (3, 3)))
            self.model.add(Activation('relu'))
            self.model.add(MaxPooling2D(pool_size=(2,
                ↳ 2)))
            self.model.add(Dropout(0.25))

        self.model.add(Flatten())
        self.model.add(Dense(1024))
        self.model.add(Activation('relu'))
        self.model.add(Dropout(0.5))
        self.model.add(Dense(self.nb_classes))
        self.model.add(Activation('softmax'))

    self.model.compile(loss=self.loss,
                       optimizer=self.optimizer,
                       metrics=self.metrics)
```

Listing 3.11: The method for creating custom models

The function in Listing 3.11 shows how we create one of our custom models. We stack convolutional layers, activations functions, pooling layers, dropout layers and fully connected layers before we compile it.

The *small_net* model consists of a total of six convolutional layers, while *large_net* has nine, both with two fully connected layers on top. More details on the custom models can be found on our github repository [3].

Training The natural next method is then the training method. We call this method with the number of epochs we want to train our model for. When training finishes, fitness matrices from each epoch is returned.

Evaluation The last important part of the program is the *evaluate()* and *metric()* methods. These do what you expect; the former calculates all the predictions and store them within the class, and the latter evaluates them by the metrics we have seen in Section 3.5.3 and returns them.

Other Methods There are also more methods implemented, as the possibility to store and load weights, change which layers are trainable, and the possibility to set other loss or optimizer functions. The class is also easily expandable to more models and features.

3.5.7 Tracking and Running CNN Experiments

We want to track and run experiments the same way we did for the machine learning algorithms with global feature extraction. The setup here is thus very similar, although with a with some modifications.

deep_learning_experiments.py

The Sacred wrapper for the deep learning class, first of all, has a different configuration. An example Sacred configuration is shown in Listing 3.12.

```
@experiment.config
def default_config():
    dataset = 'kvasir'
    epochs = 100
    advanced_preprocessing = False
    model = 'InceptionV3'
    weights = 'imagenet'
    image_width = 150
    image_height = 150
    original_top = False
    tags = []
```

Listing 3.12: Default config for CNN experiments

dataset: Path to the dataset to use.

epochs: How many epochs to run the CNN.

model: String with CNN model to use.

weights: Whether to use pre-trained weights or not. Should be set to either *imagenet* or *None*.

image_width and image_height: The wanted image size. These values are automatically altered to fit the selected model.

original_top: Boolean deciding whether to use original model top or custom top

tags: This is a list of strings for attaching custom labels to our experiments. This is also for later convenience while extracting from the database.

deep_learning_experiment_caller.py

This is similar to the *feature_experiment_caller.py* introduced in Section 3.5.5, only with different parameters.

3.5.8 Extracting Results

Finding and working with the results can be done with a standard database interface. Listing 3.13 shows a simple example on how to get database results. We first get the collection, finds all experiments for our search, sorts them by highest F1 score, limits the number of results to three, and prints out the results dictionaries.

See MongoDB[6] for more details.

```
from pymongo import MongoClient

collection = MongoClient().sacred.runs
cursor = collection.find({'config.tags.0': 'run_1',
                          'config.dataset': 'Kvasir',
                          'status': 'COMPLETED',
                          'config.weights': 'None'})
cursor.sort([('info.Metrics.F1', 1)]).limit(3)

for document in cursor:
    print(document)
```

Listing 3.13: Simple result extraction from MongoDB

3.6 Summary

The pipeline we have implemented conforms to the systems requirements stated in Section 3.1. We are satisfied with the pipeline as a whole, and, in particular, the core of the CNN system. We also feel we have gotten fine return for our time investment, and all the system requirements was met.

Considerable time went into familiarizing ourselves with all the tools and software used in our implementation, in fact, it was maybe the most time consuming part.

Even though we are satisfied with our implementation, we have multiple times gotten ideas for improvements, or more sleek ways of carrying out our pipeline, due to repeatedly better insights into our problem. Most of these was implemented, but not all. Development of the feature library, for example, was suspended at one point as we simply needed to focus on the main experiments, and could not keep re-extracting the costly image features, which was needed with each system overhaul.

[6] https://api.mongodb.com/python/current/

Chapter 4

Experiments

Before we dive into the experiments, we will resurface the goals we defined in Section 1.2. Our objectives are to:

- Establish a large scale verdict of what methods for automatic image-classification to use given varying types of datasets and training set sizes by different measuring metrics for various applications. Based on previous research and use cases, we will especially focus on finding the transition from when deep learning works and makes sense, and when it is useless, a waste of resources or too slow.
- Make an in-depth analysis of the strengths and weaknesses of the different methods and what makes for the performance differences.
- Making a well structured and easy to use testing pipeline that enables for easy further comparing and testing of automatic image-classification methods.

The last of these points are already addressed with the pipeline implementation in the previous chapter. In this chapter we will try to achieve the remaining two goals by running experiments and analyse them. We will describe their setups, and present and discuss our results.

4.1 Experiment Environments

We will start by looking at our two testing environments, whose specifications are shown in Table 4.1.

Setup A is what we will describe as a typical workstation setup, and this environment will be used to run our experiments with traditional classifiers and feature extraction.

Setup B has a the more powerful GPU, and is thus better suited for the GPU intensive CNN experiments. We note that this environment has a weaker CPU. Optimally these should be more equal, but since the GPU will be the main bottleneck for these experiments, we do not expect it to be of too much significance.

	Setup A	Setup B
Operating system	Ubuntu 16.04 LTS 64-bit	Ubuntu 16.04 LTS 64-bit
CPU	Intel Core i7 @ 3.4 GHz x 8	Intel Xeon @ 2.6 GHz x 2
Memory	8 GB	8 GB
GPU	NVIDIA GeForce GTX 980	NVIDIA Tesla K80
GPU memory	4 GB	12 GB

Table 4.1: An overview of our two environments.

4.2 Feature Extraction

We built our feature extraction around LIRE, as described in Section 3.5.2, where we are going to experiment with all of the available global image features. We will not go into specifics about the features, for that we recommend *Visual Information Retrieval using Java and LIRE* by Lux and Marques [39].

If we look at the average feature extraction times by feature and dataset, shown in Table 4.2, we note some differences. First of all, we see that especially the Holidays images are far slower to calculate than the rest. If we remember the average image sizes from the different datasets, Table 3.2, we can see a strong correlation between image size and feature extraction time. Holidays had the largest images with an average of 4.6 MP, and Caltech-256 the smallest with an average of 0.16 MP.

We also note that some features, especially *FCH* and *JCH*, are significantly slower to compute than others.

The features were extracted over time, as they was needed. Adding up the extraction time for the 19 features and 49471 images totals to just shy of 17 days.

4.3 Feature Classification Experiments

Having extracted the features, we will now move on to the experiments that uses them.

4.3.1 Configuration

We have multiple choices to make regarding the configuration.

- What feature combinations will we use?
- What classifiers will we use?
- What parameters will we use?

To start with the features; we want to test all features, and as many feature combinations as possible. Thus we will run each classifier with following feature combination:

Feature	Holidays	UKbench	Kvasir	Caltech-256
ACC	2.17	0.14	0.36	0.07
BPP	3.51	0.23	0.52	0.11
CEDD	0.31	0.02	0.06	0.01
CL	0.26	0.02	0.05	0.01
EH	0.40	0.03	0.08	0.01
FCTH	0.31	0.02	0.06	0.01
FCH	13.91	0.94	2.47	0.49
FOH	0.91	0.06	0.16	0.03
Gabor	0.15	0.02	0.04	0.01
JCD	0.47	0.03	0.09	0.02
JCH	5.13	0.32	0.87	0.16
LBP	0.30	0.02	0.07	0.01
LL	0.40	0.03	0.08	0.02
OH	0.43	0.03	0.08	0.01
PHOG	1.05	0.07	0.16	0.04
RILBP	0.60	0.04	0.10	0.02
SC	0.33	0.02	0.07	0.01
SCH	0.42	0.02	0.06	0.01
Tamura	0.42	0.06	0.11	0.05
Avg.	1.66	0.11	0.29	0.06

Table 4.2: The average feature extraction time by dataset, given in s/IMG.

- All 19 single features
- All combinations of two features
- A set of five diverse handpicked features (ACC, CL, JCD, LL, and PHOG), selected to see how the classifiers behaves to a large sized diverse feature vector
- All 19 features at once

When it comes to classifiers we have chosen to test a diverse, large set of classifiers. The set of 11 implemented classifiers from scikit-learn is a good fit.

When it comes to parameters, each algorithm has a good amount to set. The task of setting and optimizing these can be a long and tedious process, as each algorithm can be tweaked to accommodate specific data. Our understanding is that this tweaking is done to a limited degree in the real world, and that running each classifiers with fixed parameters would give more realistic results. This will also tell us how robust and diverse the classifier is, in being able to handle diverse problems without tuning. We have therefor decided to run each classifier with fixed parameters, and we will use scikit-learn's default settings. The only exception is the SVC, which will be run with three different kernels.

We will experiment on the four datasets presented in Figure 2.8.

The total number of feature experiments then becomes:

$$192 \text{ feature combinations} \times 11 \text{ classifiers} \times 4 \text{ datasets} = 8448 \text{ experiments}$$

4.3.2 Evaluating the Results

To determine if results are good or not we need some context. First, as the datasets have different number of classes, random guessing would yield different results depending on the dataset. If we make the assumption that all classes has equal probability of being guessed, as some dataset classes are of varying size, the expected value of random guessing is shown in Table 4.7.

Dataset	Classes	$E(ACC)$
Holidays	500	.002
UKBench	2550	.00039
Kvasir	8	.125
Caltech-256	256	.0039

Table 4.3: The expected accuracy values of random guessing.

In stead of comparing our scores to the expected ones, we can look at MMC, where a score of zero corresponds to random guessing and one corresponds to perferct classification. Thus, this metric can be looked on as normalized, and easier to make sence of.

But, looking at this metric alone is not enough. Different MCC scores can be more or less impressive depending on how hard it is to classify the specific dataset. Thus, what we will focus on most is how the different classifiers compare to each other. Roughly, we will speak of a classifier as having good scores if it preforms better than the median classifier.

We will present one table for each classifier and dataset combination. It will start off with the three best, the worst, and the avarage single feature. Then, we have the three best, and the worst, and the average feature combinations of two features. After that, we have our five handpicked features, and lastly the classification using all available features.

4.3.3 Nearest Neighbor

The k Nearest Neighbor (kNN) method has two main parameters. The first one is the number of points in our neighborhood, which defaults to five. The second parameter is how the samples in the neighborhood are weighted. This parameter defaults to *uniform*, meaning all samples contribute equally, in contrast to weighing where the closer counts more.

Looking at the results in Table 4.4, we can see some interesting things. Firstly, as kNN is an instance based algorithm, it requires close to no training time at all, it is more of an initialisation time. Secondly, as a result of this, the prediction time is quite high. This is especially true for datasets with many samples and large feature spaces.

Where the potential of kNN are shown, are with datasets with few samples per class and potentially many classes. This can be seen for the UKBench dataset, where kNN is one of the better performers, regardless of the metric.

Pros

- Few parameters
- Good at handling irregular decision boundaries
- Negligible training time
- Handles datasets with many classes well

Cons

- Slow predictions, especially when many samples or large feature spaces
- No feature weighting, thus exposed for noisy features

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	0.14	0.06	2.33	0.58	0.54	0.58	0.56	0.58	1.00
JCD	168	0.03	0.04	1.21	0.50	0.46	0.50	0.48	0.50	1.00
FCTH	192	0.02	0.04	1.13	0.50	0.45	0.50	0.47	0.50	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.02	0.01	0.10	0.01	0.00	0.01	0.00	0.01	1.00
Avg.	199	0.11	0.05	1.35	0.24	0.21	0.24	0.21	0.24	1.00
ACC, EH	336	0.17	0.08	3.10	0.61	0.57	0.61	0.60	0.61	1.00
ACC, CL	289	0.16	0.06	2.67	0.61	0.56	0.61	0.59	0.61	1.00
ACC, JCD	424	0.17	0.10	3.75	0.59	0.55	0.59	0.57	0.59	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, LL	124	0.05	0.03	0.83	0.03	0.02	0.03	0.02	0.03	1.00
Avg.	398	0.22	0.10	2.53	0.26	0.22	0.26	0.23	0.26	1.00
5 h.p. ft.	1151	0.29	0.33	10.14	0.31	0.28	0.31	0.30	0.31	1.00
All 19 ft.	3778	2.13	1.25	31.73	0.42	0.37	0.42	0.38	0.42	1.00

(a) UKBench dataset

Table 4.4: Execution times and performances for different combinations of global features using a **Nearest Neighbors** classifier

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	2.17	0.01	0.32	0.30	0.26	0.30	0.27	0.30	1.00
FCTH	192	0.31	0.00	0.24	0.30	0.24	0.30	0.24	0.30	1.00
JCD	168	0.47	0.00	0.24	0.30	0.24	0.30	0.25	0.30	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.15	0.00	0.05	0.02	0.01	0.01	0.01	0.02	1.00
Avg.	199	1.66	0.00	0.24	0.16	0.13	0.16	0.13	0.16	1.00
ACC, CL	289	2.43	0.01	0.35	0.33	0.28	0.33	0.29	0.33	1.00
ACC, SC	320	2.50	0.01	0.36	0.33	0.28	0.33	0.29	0.33	1.00
ACC, EH	336	2.58	0.01	0.40	0.31	0.27	0.31	0.29	0.31	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, LL	124	0.54	0.00	0.15	0.03	0.02	0.03	0.02	0.03	1.00
Avg.	398	3.31	0.01	0.41	0.17	0.13	0.17	0.14	0.17	1.00
5 h.p. ft.	1151	4.35	0.03	1.32	0.21	0.18	0.20	0.21	0.21	1.00
All 19 ft.	3778	31.48	0.08	4.39	0.26	0.21	0.26	0.22	0.26	1.00

(b) Holidays dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
PHOG	630	0.04	0.68	14.92	0.17	0.16	0.16	0.22	0.17	1.00
EH	80	0.01	0.14	2.58	0.17	0.15	0.17	0.19	0.17	1.00
CL	33	0.01	0.03	0.45	0.13	0.11	0.12	0.15	0.13	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.01	0.05	0.03	0.01	0.01	0.01	0.02	0.01	1.00
Avg.	199	0.06	0.28	3.61	0.09	0.08	0.08	0.10	0.09	1.00
CEDD, EH	224	0.03	0.31	5.80	0.20	0.18	0.19	0.22	0.20	1.00
ACC, PHOG	886	0.11	1.01	20.38	0.19	0.17	0.19	0.25	0.19	1.00
CL, PHOG	663	0.05	0.71	15.52	0.19	0.17	0.18	0.25	0.19	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, SC	124	0.02	0.11	0.55	0.05	0.04	0.04	0.06	0.05	0.99
Avg.	398	0.12	0.53	6.43	0.10	0.09	0.10	0.12	0.10	1.00
5 h.p. ft.	1151	0.16	1.36	26.05	0.18	0.16	0.17	0.23	0.18	1.00
All 19 ft.	3778	1.12	5.04	57.58	0.14	0.12	0.14	0.15	0.14	1.00

(c) Caltech-256 dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	0.36	0.05	1.21	0.69	0.69	0.65	0.69	0.69	0.96
JCD	168	0.09	0.03	0.45	0.68	0.67	0.63	0.68	0.68	0.95
CEDD	144	0.06	0.03	0.44	0.68	0.67	0.63	0.67	0.68	0.95
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Tamura	18	0.11	0.01	0.06	0.32	0.31	0.22	0.32	0.32	0.90
Avg.	199	0.29	0.04	0.83	0.53	0.52	0.46	0.53	0.53	0.93
ACC, EH	336	0.44	0.07	1.89	0.71	0.71	0.67	0.71	0.71	0.96
ACC, CL	289	0.41	0.05	1.22	0.71	0.71	0.67	0.71	0.71	0.96
ACC, FCTH	448	0.42	0.09	2.18	0.70	0.69	0.65	0.69	0.70	0.96
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
EH, Tamura	98	0.19	0.02	0.27	0.33	0.32	0.23	0.33	0.33	0.90
Avg.	398	0.58	0.08	1.51	0.57	0.57	0.51	0.57	0.57	0.94
5 h.p. ft.	1151	0.75	0.24	8.64	0.63	0.62	0.58	0.63	0.63	0.95
All 19 ft.	3778	5.50	0.87	15.09	0.66	0.66	0.61	0.66	0.66	0.95

(d) Kvasir dataset

Table 4.4: Execution times and performances for different combinations of global features using a **Nearest Neighbors** classifier (cont.)

4.3.4 Decision Tree

The decision tree classifier also has relatively few parameters. There are one measuring the split quality, called *criterion* , and a few determining limitations to the tree structure. The default does not set limitations on max depth nor minimum number of samples in a node.

Form the results in Table 4.5, we first see that the prediction times are very small, in fact the lowest of all classifiers tested. The training times are also quite low, although do increase when noisy, or bad, features are included. However, the inclusion of noisy features does not negatively affect the overall classification results significantly.

A weakness of the decision tree classifier is its potential to overfit the data. We can see a tendency for the algorithm to preform worse on datasets with scant classes. The risk of overfitting can be reduced, or controlled, by having depth restrictions in our tree or a minimum number of samples at each node.

Pros

- Very low prediction time
- Results not affected by noisy features
- Simple to interpret classification rules
- Handles large datasets well

Cons

- Tendency to overfit scant classes
- Slow training with noisy features
- Has a hard time constructing specific classification rules
- Has bias towards large classes

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
FCTH	192	0.02	1.55	0.01	0.33	0.30	0.33	0.32	0.33	1.00
JCD	168	0.03	3.68	0.01	0.30	0.27	0.30	0.28	0.30	1.00
CEDD	144	0.02	2.69	0.01	0.28	0.25	0.28	0.26	0.28	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.02	25.52	0.01	0.02	0.01	0.02	0.01	0.02	1.00
Avg.	199	0.11	14.18	0.01	0.15	0.14	0.15	0.14	0.15	1.00
FCTH, SC	256	0.05	4.71	0.01	0.32	0.29	0.32	0.30	0.32	1.00
JCD, SCH	232	0.06	9.85	0.01	0.31	0.28	0.31	0.30	0.31	1.00
JCD, RILBP	204	0.07	8.81	0.01	0.31	0.28	0.31	0.29	0.31	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
LL, PHOG	694	0.10	60.16	0.01	0.03	0.03	0.03	0.03	0.03	1.00
Avg.	398	0.22	37.26	0.01	0.19	0.17	0.19	0.18	0.19	1.00
5 h.p. ft.	1151	0.29	114.91	0.01	0.23	0.20	0.23	0.21	0.23	1.00
All 19 ft.	3778	2.13	916.77	0.02	0.27	0.24	0.27	0.25	0.27	1.00

(a) UKBench dataset

Table 4.5: Execution times and performances for different combinations of global features using a **Decision Tree** classifier

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
FCTH	192	0.31	0.06	0.00	0.27	0.24	0.27	0.24	0.27	1.00
SCH	64	0.42	0.11	0.00	0.22	0.19	0.22	0.19	0.22	1.00
CEDD	144	0.31	0.10	0.00	0.22	0.19	0.22	0.20	0.22	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
LL	64	0.40	0.18	0.00	0.02	0.02	0.02	0.01	0.02	1.00
Avg.	199	1.66	0.43	0.00	0.14	0.12	0.13	0.12	0.14	1.00
CEDD, FCTH	336	0.62	0.18	0.00	0.27	0.24	0.27	0.26	0.27	1.00
FCTH, SC	256	0.64	0.15	0.00	0.25	0.23	0.25	0.25	0.25	1.00
CEDD, RILBP	180	0.91	0.22	0.00	0.25	0.23	0.25	0.24	0.25	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
LL, Tamura	82	0.82	0.43	0.00	0.05	0.05	0.05	0.05	0.05	1.00
Avg.	398	3.31	1.01	0.00	0.16	0.14	0.16	0.15	0.16	1.00
5 h.p. ft.	1151	4.35	3.86	0.00	0.16	0.14	0.16	0.14	0.16	1.00
All 19 ft.	3778	31.48	16.03	0.01	0.17	0.15	0.17	0.16	0.17	1.00

(b) Holidays dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
CL	33	0.01	0.98	0.00	0.09	0.09	0.09	0.09	0.09	1.00
JCD	168	0.02	1.48	0.00	0.08	0.08	0.07	0.08	0.08	1.00
EH	80	0.01	1.58	0.00	0.08	0.08	0.07	0.08	0.08	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.01	8.41	0.00	0.02	0.02	0.01	0.02	0.02	0.99
Avg.	199	0.06	4.30	0.00	0.06	0.06	0.06	0.06	0.06	1.00
JCD, Tamura	186	0.07	3.32	0.00	0.10	0.10	0.10	0.10	0.10	1.00
CL, Tamura	51	0.06	2.73	0.00	0.10	0.10	0.10	0.10	0.10	1.00
CL, FCTH	225	0.02	2.05	0.00	0.10	0.10	0.09	0.10	0.10	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, SC	124	0.02	9.43	0.00	0.04	0.04	0.04	0.05	0.04	1.00
Avg.	398	0.12	8.96	0.06	0.08	0.08	0.07	0.08	0.08	1.00
5 h.p. ft.	1151	0.16	34.07	0.00	0.10	0.10	0.10	0.10	0.10	1.00
All 19 ft.	3778	1.12	128.35	0.01	0.11	0.11	0.10	0.11	0.11	1.00

(c) Caltech-256 dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
JCD	168	0.09	0.05	0.00	0.63	0.63	0.57	0.63	0.63	0.95
ACC	256	0.36	0.20	0.00	0.62	0.62	0.57	0.63	0.62	0.95
SCH	64	0.06	0.04	0.00	0.63	0.62	0.57	0.62	0.63	0.95
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
LL	64	0.08	0.06	0.00	0.32	0.32	0.23	0.33	0.32	0.90
Avg.	199	0.29	0.17	0.00	0.50	0.50	0.42	0.50	0.50	0.93
ACC, OH	320	0.44	0.22	0.00	0.66	0.66	0.62	0.66	0.66	0.95
CL, JCD	201	0.14	0.10	0.00	0.66	0.66	0.61	0.66	0.66	0.95
ACC, FCTH	448	0.42	0.24	0.00	0.66	0.66	0.61	0.66	0.66	0.95
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
RILBP, Tamura	54	0.22	0.11	0.00	0.36	0.36	0.26	0.36	0.36	0.91
Avg.	398	0.58	0.36	0.00	0.58	0.58	0.52	0.58	0.58	0.94
5 h.p. ft.	1151	0.75	1.39	0.00	0.65	0.65	0.60	0.65	0.65	0.95
All 19 ft.	3778	5.50	3.99	0.01	0.66	0.66	0.62	0.66	0.66	0.95

(d) Kvasir dataset

Table 4.5: Execution times and performances for different combinations of global features using a **Decision Tree** classifier (cont.)

4.3.5 Random Forest

The random forest classifier builds on the decision tree classifier and has much of the same parameters. The *criterion* is set as in the decision tree, and there is not any limitations to the tree structure, just as for the decision tree. In addition, we have a parameter for the number of estimators, a.k.a. the number of trees in the forrest, default to ten. Furthermore, we have a parameter determining the number of features we will consider when looking for the best split, default to the square root of the number of features. The random forest implementation we used averaged the probabilistic scores from each decision tree, in stead of simply counting votes.

Comparing this classifier to decision tree, we can see that the prediction time is is higher, but still very low. We note that the prediction time for the UKBench dataset, with most classes, have a notably higher prediction time, as also was the case with decision trees. The training times of random forest are universally lower, especially regarding the worst case scenarios.

The classification scores are good, and all over higher than the decision tree. The random forest turns out to be a more stable classifier, with less overfitting and less fluctuations in training time with the cost of higher, yet still low, prediction time.

Pros

- Low prediction time
- Handles large datasets well
- More stable than decision trees

Cons

- Bias towards large classes
- Not as fast prediction times nor as easy to understand as decision trees

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
FCTH	192	0.02	1.78	0.19	0.47	0.43	0.47	0.45	0.47	1.00
ACC	256	0.14	7.77	0.19	0.42	0.39	0.42	0.42	0.42	1.00
JCD	168	0.03	2.73	0.19	0.42	0.39	0.42	0.42	0.42	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
LL	64	0.03	5.25	0.19	0.02	0.02	0.02	0.02	0.02	1.00
Avg.	199	0.11	8.32	0.19	0.22	0.19	0.21	0.21	0.22	1.00
FCTH, OH	256	0.05	2.79	0.18	0.48	0.44	0.48	0.48	0.48	1.00
ACC, JCD	424	0.17	8.09	0.19	0.47	0.43	0.47	0.47	0.47	1.00
FCTH, SCH	256	0.05	3.43	0.19	0.47	0.43	0.47	0.46	0.47	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
LL, PHOG	694	0.10	20.49	0.19	0.04	0.03	0.04	0.04	0.04	1.00
Avg.	398	0.22	13.50	0.19	0.25	0.23	0.25	0.25	0.25	1.00
5 h.p. ft.	1151	0.29	22.75	0.19	0.25	0.23	0.25	0.26	0.25	1.00
All 19 ft.	3778	2.13	74.74	0.20	0.32	0.28	0.32	0.31	0.32	1.00

(a) UKBench dataset

Table 4.6: Execution times and performances for different combinations of global features using a **Random Forest** classifier

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
FCTH	192	0.31	0.08	0.03	0.38	0.34	0.38	0.35	0.38	1.00
JCD	168	0.47	0.12	0.03	0.37	0.33	0.36	0.34	0.37	1.00
CEDD	144	0.31	0.11	0.03	0.35	0.31	0.35	0.31	0.35	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
LL	64	0.40	0.23	0.03	0.03	0.02	0.03	0.02	0.03	1.00
Avg.	199	1.66	0.27	0.03	0.20	0.18	0.20	0.18	0.20	1.00
FCTH, OH	256	0.74	0.12	0.03	0.37	0.33	0.36	0.34	0.37	1.00
FCTH, SCH	256	0.74	0.13	0.03	0.36	0.32	0.36	0.34	0.36	1.00
JCD, OH	232	0.90	0.15	0.03	0.36	0.32	0.36	0.33	0.36	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
EH, LL	144	0.80	0.29	0.03	0.07	0.06	0.07	0.06	0.07	1.00
Avg.	398	3.31	0.40	0.03	0.23	0.20	0.22	0.20	0.23	1.00
5 h.p. ft.	1151	4.35	0.86	0.03	0.22	0.20	0.22	0.22	0.22	1.00
All 19 ft.	3778	31.48	1.82	0.04	0.28	0.25	0.28	0.27	0.28	1.00

(b) Holidays dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
CL	33	0.01	1.34	0.02	0.12	0.11	0.11	0.11	0.12	1.00
EH	80	0.01	1.44	0.02	0.11	0.10	0.11	0.11	0.11	1.00
JCD	168	0.02	1.19	0.02	0.11	0.10	0.11	0.10	0.11	0.99
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.01	7.47	0.02	0.02	0.02	0.02	0.02	0.02	0.99
Avg.	199	0.06	2.55	0.02	0.08	0.07	0.08	0.08	0.08	0.99
CL, EH	113	0.02	2.03	0.02	0.13	0.12	0.12	0.14	0.13	1.00
CL, Tamura	51	0.06	3.09	0.02	0.13	0.12	0.13	0.13	0.13	1.00
JCD, Tamura	186	0.07	1.96	0.02	0.13	0.12	0.13	0.13	0.13	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, SC	124	0.02	6.28	0.02	0.06	0.05	0.05	0.05	0.06	0.99
Avg.	398	0.12	3.44	0.08	0.10	0.09	0.10	0.10	0.10	0.99
5 h.p. ft.	1151	0.16	7.25	0.02	0.13	0.12	0.13	0.14	0.13	1.00
All 19 ft.	3778	1.12	13.51	0.03	0.13	0.12	0.13	0.14	0.13	1.00

(c) Caltech-256 dataset

Feature	Feature SPace *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	0.36	0.13	0.00	0.71	0.71	0.67	0.71	0.71	0.96
CL	33	0.05	0.09	0.00	0.68	0.68	0.64	0.68	0.68	0.95
FCTH	192	0.06	0.08	0.00	0.68	0.68	0.63	0.68	0.68	0.95
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
LL	64	0.08	0.10	0.00	0.36	0.36	0.27	0.36	0.36	0.91
Avg.	199	0.29	0.13	0.00	0.55	0.54	0.48	0.54	0.55	0.94
ACC, CL	289	0.41	0.15	0.00	0.72	0.72	0.68	0.72	0.72	0.96
ACC, FCTH	448	0.42	0.14	0.00	0.71	0.71	0.67	0.71	0.71	0.96
ACC, JCD	424	0.45	0.14	0.00	0.71	0.71	0.67	0.71	0.71	0.96
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
LL, Tamura	82	0.20	0.14	0.00	0.39	0.39	0.31	0.39	0.39	0.91
Avg.	398	0.58	0.18	0.00	0.63	0.62	0.57	0.62	0.63	0.95
5 h.p. ft.	1151	0.75	0.31	0.00	0.71	0.70	0.66	0.71	0.71	0.96
All 19 ft.	3778	5.50	0.52	0.01	0.70	0.70	0.66	0.70	0.70	0.96

(d) Kvasir dataset

Table 4.6: Execution times and performances for different combinations of global features using a **Random Forest** classifier (cont.)

4.3.6 AdaBoost

The AdaBoost classifier is run with the the default 50 estimators, and the boosted ensemble is built around the decision tree classifier.

From the experiment results of AdaBoost in Table 4.7, we can see worse results than a simple decision tree on all metrics and datasets. It is especially bad at the UKBench and Holidays datasets, which are the datasets with the most number of classes. It can seem like the classifier has trouble with many classed datasets, as this naturally causes many initially wrong classifications, and thus many samples in need of boosting. The following classifications will thus not be significantly easier to do, and the difficult, heavily boosted, samples will have large influence on the result.

Both the training times and the prediction times are also higher than both the decision trees and the random forests. Comparing to all the other classifiers, not only the decision tree based, the training and prediction times are average at best, and thus the speed advantage of the decision trees are lost.

We can speculate that the AdaBoost classifier would have been more competitive in, e.g., a bi-classification setting.

Pros

- Handles large classes well

Cons

- Bias towards large classes
- Not as fast, accurate, nor as easy to understand as decision trees
- Has trouble with many classes

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
JCD	168	0.03	39.77	5.25	0.00	0.00	0.03	0.00	0.00	1.00
FCH	125	0.94	44.23	5.24	0.00	0.00	0.03	0.00	0.00	1.00
FOH	576	0.06	40.89	5.24	0.00	0.00	0.03	0.00	0.00	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
RILBP	36	0.04	39.55	5.22	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	199	0.11	44.41	5.29	0.00	0.00	0.01	0.00	0.00	1.00
JCD, PHOG	798	0.10	45.66	5.24	0.00	0.00	0.03	0.00	0.00	1.00
JCD, LL	232	0.07	39.24	5.23	0.00	0.00	0.03	0.00	0.00	1.00
FCTH, FOH	768	0.08	40.54	5.24	0.00	0.00	0.03	0.00	0.00	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
BPP, RILBP	792	0.27	43.46	5.23	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	398	0.22	48.67	5.25	0.00	0.00	0.02	0.00	0.00	1.00
5 h.p. ft.	1151	0.29	51.09	5.25	0.00	0.00	0.03	0.00	0.00	1.00
All 19 ft.	3778	2.13	147.71	5.29	0.00	0.00	0.01	0.00	0.00	1.00

(a) UKBench dataset

Table 4.7: Execution times and performances for different combinations of global features using a **AdaBoost** classifier

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	2.17	1.38	0.84	0.02	0.01	0.04	0.00	0.02	0.99
CEDD	144	0.31	1.12	0.81	0.01	0.00	0.03	0.00	0.01	0.99
RILBP	36	0.60	1.12	0.82	0.02	0.00	0.02	0.00	0.02	0.99
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Tamura	18	0.42	1.26	0.82	0.01	0.00	0.00	0.00	0.01	0.99
Avg.	199	1.66	1.44	0.83	0.01	0.00	0.02	0.00	0.01	0.99
ACC, JCH	448	7.30	2.64	0.82	0.02	0.01	0.06	0.01	0.02	0.99
ACC, SCH	320	2.60	1.52	0.82	0.02	0.01	0.06	0.01	0.02	0.99
FOH, SCH	640	1.33	1.34	0.83	0.02	0.01	0.05	0.01	0.02	0.99
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, Tamura	78	0.57	3.15	0.81	0.01	0.00	0.00	0.00	0.01	0.99
Avg.	398	3.31	1.83	0.82	0.01	0.00	0.02	0.00	0.01	0.99
5 h.p. ft.	1151	4.35	2.88	0.84	0.02	0.01	0.05	0.01	0.02	0.99
All 19 ft.	3778	31.48	9.36	0.89	0.01	0.00	0.02	0.00	0.01	0.99

(b) Holidays dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
JCH	192	0.16	18.23	0.44	0.05	0.02	0.04	0.02	0.05	0.99
FCTH	192	0.01	12.88	0.45	0.06	0.02	0.05	0.02	0.06	0.99
EH	80	0.01	12.44	0.44	0.06	0.02	0.05	0.01	0.06	0.99
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.01	34.52	0.44	0.05	0.01	0.03	0.01	0.05	0.99
Avg.	199	0.06	15.85	0.44	0.05	0.02	0.04	0.01	0.05	0.99
JCD, LBP	424	0.03	0.27	2.64	0.08	0.07	0.08	0.08	0.08	1.00
FCH, SCH	189	0.50	0.27	2.64	0.08	0.07	0.08	0.08	0.08	1.00
FCH, PHOG	755	0.52	0.27	2.64	0.08	0.07	0.08	0.08	0.08	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, Tamura	78	0.06	36.78	0.43	0.04	0.01	0.02	0.00	0.04	0.98
Avg.	398	0.12	20.47	0.50	0.05	0.02	0.04	0.02	0.05	0.99
5 h.p. ft.	1151	0.16	38.17	0.46	0.05	0.02	0.04	0.02	0.05	0.99
All 19 ft.	3778	1.12	123.16	0.52	0.05	0.02	0.04	0.02	0.05	0.98

(c) Caltech-256 sataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
SCH	64	0.06	0.50	0.02	0.48	0.40	0.42	0.49	0.48	0.93
CL	33	0.05	0.49	0.02	0.48	0.40	0.42	0.45	0.48	0.93
ACC	256	0.36	1.19	0.02	0.48	0.39	0.42	0.43	0.48	0.93
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.04	1.97	0.02	0.28	0.25	0.18	0.27	0.28	0.90
Avg.	199	0.29	1.11	0.02	0.38	0.33	0.31	0.38	0.38	0.91
FOH, Tamura	594	0.28	1.69	0.03	0.52	0.52	0.46	0.55	0.52	0.93
BPP, FOH	1332	0.69	4.28	0.05	0.56	0.52	0.50	0.55	0.56	0.94
BPP, CL	789	0.58	2.95	0.04	0.55	0.51	0.49	0.54	0.55	0.94
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
FOH, JCH	768	1.03	3.21	0.04	0.23	0.14	0.15	0.22	0.23	0.89
Avg.	398	0.58	2.00	0.03	0.42	0.36	0.35	0.41	0.42	0.92
5 h.p. ft.	1151	0.75	6.45	0.04	0.48	0.43	0.42	0.53	0.48	0.93
All 19 ft.	3778	5.50	20.34	0.10	0.46	0.36	0.42	0.45	0.46	0.92

(d) Kvasir dataset

Table 4.7: Execution times and performances for different combinations of global features using a **AdaBoost** (cont.)

4.3.7 Linear SVC

The SVCs are highly parameterized algorithms, with specifications for penalty, loss and tolerance, all default to the scikit-learn default.

We note that the SVC with linear kernel uses a different implementation than the other SVCs with other kernels.

The results of our first SVC classifier is displayed in Table 4.8. Firstly, we can observe very good overall classification scores. Top results on Kvasir, Holidays and UKBench are impressive, while the results on Caltech-256 are considerably lower but, compared to the other classifiers, still respectable, especially when using all 19 features.

It can seem like the classifier has trouble generalizing the complexity of the Caltech-256 dataset.

The prediction times are among the fastest while the training times are varying. Notably, the training times on Caltech-256 are substantially higher.

Pros

- Very fast prediction times
- Good at handling noise
- Handles both many and large classes and large feature spaces
- Easy to interpret decision rule

Cons

- High training time for large datasets
- Difficulty in generalizing complex solutions

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	0.14	41.75	0.04	0.76	0.74	0.76	0.78	0.76	1.00
JCD	168	0.03	21.30	0.03	0.66	0.63	0.66	0.66	0.66	1.00
FCTH	192	0.02	15.09	0.03	0.60	0.58	0.60	0.62	0.60	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.02	90.26	0.03	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	199	0.11	92.65	0.04	0.22	0.21	0.22	0.23	0.22	1.00
ACC, CL	289	0.16	68.38	0.04	0.87	0.85	0.87	0.87	0.87	1.00
ACC, RILBP	292	0.18	74.11	0.04	0.86	0.85	0.86	0.87	0.86	1.00
ACC, SC	320	0.16	69.90	0.04	0.85	0.83	0.85	0.85	0.85	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, LL	124	0.05	139.91	0.03	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	398	0.22	101.48	0.05	0.34	0.33	0.35	0.36	0.34	1.00
5 h.p. ft.	1151	0.29	199.12	0.11	0.66	0.64	0.66	0.66	0.66	1.00
All 19 ft.	3778	2.13	756.97	0.25	0.75	0.73	0.75	0.75	0.75	1.00

(a) UKBench dataset

Table 4.8: Execution times and performances for different combinations of global features using a **Linear SVC** classifier

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	2.17	2.28	0.01	0.62	0.57	0.62	0.57	0.62	1.00
JCD	168	0.47	1.12	0.01	0.58	0.53	0.57	0.54	0.58	1.00
CEDD	144	0.31	1.04	0.02	0.56	0.51	0.56	0.52	0.56	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.15	10.23	0.01	0.01	0.00	0.01	0.00	0.01	1.00
Avg.	199	1.66	3.74	0.01	0.22	0.20	0.22	0.21	0.22	1.00
ACC, LBP	512	2.48	8.14	0.01	0.70	0.66	0.70	0.67	0.70	1.00
ACC, RILBP	292	2.77	5.38	0.01	0.70	0.65	0.70	0.65	0.70	1.00
ACC, CL	289	2.43	3.81	0.01	0.69	0.65	0.69	0.66	0.69	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, Tamura	78	0.57	5.39	0.01	0.01	0.01	0.01	0.01	0.01	1.00
Avg.	398	3.31	5.25	0.01	0.33	0.30	0.33	0.31	0.33	1.00
5 h.p. ft.	1151	4.35	11.58	0.05	0.56	0.51	0.56	0.51	0.56	1.00
All 19 ft.	3778	31.48	44.98	0.07	0.62	0.58	0.62	0.59	0.62	1.00

(b) Holidays dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
PHOG	630	0.04	404.75	0.01	0.12	0.12	0.11	0.14	0.12	1.00
JCD	168	0.02	134.11	0.00	0.15	0.11	0.14	0.10	0.15	0.99
BPP	756	0.11	315.58	0.01	0.10	0.11	0.10	0.12	0.10	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.01	1063.68	0.00	0.02	0.00	0.01	0.00	0.02	0.99
Avg.	199	0.06	479.44	0.00	0.05	0.04	0.06	0.05	0.05	0.99
JCH, PHOG	822	0.20	730.08	0.01	0.18	0.19	0.18	0.23	0.18	1.00
EH, JCD	248	0.03	164.76	0.00	0.19	0.18	0.19	0.20	0.19	1.00
BPP, PHOG	1386	0.15	367.33	0.02	0.18	0.18	0.17	0.19	0.18	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, LL	124	0.04	1634.00	0.00	0.01	0.00	0.00	0.01	0.01	1.00
Avg.	398	0.12	640.94	0.05	0.08	0.07	0.08	0.11	0.08	1.00
5 h.p. ft.	1151	0.16	282.70	0.01	0.22	0.23	0.22	0.24	0.22	1.00
All 19 ft.	3778	1.12	1428.96	0.04	0.29	0.29	0.28	0.29	0.29	1.00

(c) Caltech-256 dataset

Feature	Feature SPace *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
JCD	168	0.09	1.24	0.00	0.71	0.71	0.67	0.71	0.71	0.96
CEDD	144	0.06	1.22	0.00	0.69	0.69	0.65	0.69	0.69	0.96
FCTH	192	0.06	0.89	0.00	0.69	0.69	0.65	0.69	0.69	0.96
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
RILBP	36	0.10	3.08	0.00	0.18	0.08	0.09	0.25	0.18	0.88
Avg.	199	0.29	3.84	0.00	0.47	0.42	0.42	0.50	0.47	0.92
EH, FCTH	272	0.13	2.26	0.00	0.72	0.72	0.69	0.74	0.72	0.96
ACC, FCH	381	2.83	8.32	0.00	0.73	0.72	0.69	0.74	0.73	0.96
CEDD, JCD	312	0.15	1.77	0.00	0.72	0.72	0.68	0.72	0.72	0.96
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, Tamura	78	0.16	5.92	0.00	0.16	0.09	0.11	0.35	0.16	0.88
Avg.	398	0.58	5.45	0.00	0.58	0.55	0.53	0.64	0.58	0.94
5 h.p. ft.	1151	0.75	9.81	0.00	0.71	0.71	0.67	0.71	0.71	0.96
All 19 ft.	3778	5.50	35.09	0.02	0.77	0.76	0.74	0.79	0.77	0.97

(d) Kvasir dataset

Table 4.8: Execution times and performances for different combinations of global features using a **Linear SVC** classifier (cont.)

4.3.8 SVC Poly 2

The second SVC have a *polynomial* kernel of degree 2. Other parameters are gamma, tolerance, and the decision function shape.

The results, illustrated in Table 4.9, are very similar to the linear SVC, however, the results on the the Caltech-256 dataset are significantly better. The polynomial kernel allows for more complex decision rules, which suits Caltech-256 better.

For runs with "good" feature choices, we see that the training time is about half of that of the linear SVC. The worst cast training times, however, are much worse. The explanation behind this is that the linear SVC implementation has a default number of maximum iterations, set to 1000, while this implementation has no iteration cap. In Table 4.9d we can see one experiment needing more than two days to finnish. We also had a few experiments not finishing within 100 hours, who was canceled.

The prediction times for datasets with many classes are slow, in fact among the slowest of all.

Pros

- Faster training than linear kernel
- Not too exposed for noise
- Handles large classes well

Cons

- Slow predictions, especially with many classes

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	0.14	25.89	62.80	0.83	0.82	0.83	0.84	0.83	1.00
JCD	168	0.03	19.38	61.93	0.69	0.66	0.69	0.68	0.69	1.00
FCTH	192	0.02	20.30	62.18	0.66	0.63	0.66	0.64	0.66	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.02	16.58	62.28	0.01	0.00	0.01	0.00	0.01	1.00
Avg.	199	0.11	21.70	63.44	0.38	0.35	0.38	0.37	0.38	1.00
ACC, CL	289	0.16	28.01	62.75	0.88	0.87	0.88	0.88	0.88	1.00
ACC, EH	336	0.17	31.63	63.20	0.86	0.84	0.86	0.86	0.86	1.00
ACC, SC	320	0.16	29.08	63.01	0.85	0.83	0.85	0.85	0.85	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, LL	124	0.05	17.77	62.05	0.06	0.05	0.06	0.05	0.06	1.00
Avg.	398	0.22	32.56	65.87	0.43	0.41	0.43	0.43	0.43	1.00
5 h.p. ft.	1151	0.29	79.79	67.31	0.52	0.50	0.52	0.54	0.52	1.00
All 19 ft.	3778	2.13	223.88	96.64	0.65	0.62	0.65	0.65	0.65	1.00

(a) UKBench dataset

Table 4.9: Execution times and performances for different combinations of global features using a **SVC Poly 2** classifier

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	2.17	0.74	1.54	0.62	0.57	0.62	0.56	0.62	1.00
JCH	192	5.13	0.61	1.41	0.43	0.39	0.43	0.40	0.43	1.00
CL	33	0.26	0.42	1.37	0.42	0.37	0.42	0.38	0.42	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.15	2.36	1.35	0.03	0.01	0.02	0.00	0.03	0.99
Avg.	199	1.66	0.73	1.50	0.26	0.22	0.26	0.23	0.26	1.00
ACC, CL	289	2.43	0.79	1.59	0.66	0.62	0.66	0.62	0.66	1.00
ACC, SC	320	2.50	0.82	1.63	0.64	0.59	0.63	0.60	0.64	1.00
ACC, EH	336	2.58	0.87	1.64	0.64	0.59	0.64	0.59	0.64	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, LL	124	0.54	0.55	1.47	0.06	0.04	0.06	0.04	0.06	1.00
Avg.	398	3.31	0.90	1.74	0.35	0.31	0.35	0.32	0.35	1.00
5 h.p. ft.	1151	4.35	2.03	2.62	0.45	0.40	0.45	0.40	0.45	1.00
All 19 ft.	3778	31.48	5.66	4.42	0.55	0.50	0.54	0.51	0.55	1.00

(b) Holidays dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
PHOG	630	0.04	187.93	13.98	0.24	0.22	0.23	0.22	0.24	1.00
EH	80	0.01	31.57	5.69	0.21	0.20	0.21	0.20	0.21	1.00
JCH	192	0.16	56.81	7.37	0.17	0.16	0.16	0.16	0.17	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
LL	64	0.02	6549.22	5.65	0.04	0.03	0.04	0.04	0.04	0.99
Avg.	207	0.06	1003.22	7.65	0.13	0.11	0.13	0.11	0.13	0.99
ACC, PHOG	886	0.11	258.56	17.67	0.29	0.28	0.29	0.29	0.29	1.00
CEDD, EH	224	0.03	64.36	7.78	0.28	0.27	0.27	0.27	0.28	1.00
EH, JCD	248	0.03	69.47	8.06	0.28	0.27	0.27	0.27	0.28	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, SC	124	0.02	538.27	6.40	0.08	0.06	0.07	0.06	0.08	0.99
Avg.	399	0.12	225.17	10.43	0.17	0.16	0.16	0.16	0.17	1.00
5 h.p. ft.	1151	0.16	330.33	21.03	0.30	0.29	0.30	0.30	0.30	1.00
All 19 ft.	3778	1.12	882.17	58.53	0.26	0.25	0.25	0.25	0.26	1.00

(c) Caltech-256 dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	0.36	1.70	0.53	0.73	0.73	0.69	0.73	0.73	0.96
CEDD	144	0.06	0.74	0.38	0.72	0.71	0.68	0.72	0.72	0.96
JCD	168	0.09	0.89	0.46	0.71	0.71	0.67	0.71	0.71	0.96
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
LL	64	0.08	13.82	0.24	0.33	0.32	0.24	0.33	0.33	0.90
Avg.	199	0.29	3318.16	0.59	0.58	0.58	0.52	0.58	0.58	0.94
ACC, FCTH	448	0.42	2.13	0.98	0.75	0.75	0.71	0.75	0.75	0.96
CL, JCD	201	0.14	1.61	0.39	0.75	0.75	0.71	0.75	0.75	0.96
EH, FCTH	272	0.13	1.40	0.66	0.75	0.74	0.71	0.75	0.75	0.96
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
LL, Tamura	82	0.20	19479.22	0.65	0.36	0.37	0.27	0.37	0.36	0.91
Avg.	398	0.58	962.40	1.03	0.66	0.66	0.61	0.66	0.66	0.95
5 h.p. ft.	1151	0.75	6.84	3.17	0.75	0.75	0.72	0.75	0.75	0.96
All 19 ft.	3778	5.50	18.00	8.79	0.76	0.76	0.73	0.76	0.76	0.97

(d) Kvasir dataset

Table 4.9: Execution times and performances for different combinations of global features using a **SVC Poly 2** classifier (cont.)

4.3.9 SVC RBF

The final SVC have an *RBF* kernel. The important parameters are similar to those of the polynomial SVC; gamma, tolerance, and the decision function shape.

The RBF kernel is yet more complex than the polynomial. This implies it is more prone to overfit, and require more samples per class for good results.

This is supported by our results, shown in Table 4.10, where we can see that the datasets with small classes, UKBench and Holidays, perform worse than the SVCs with simpler kernels, while the results of Kvasir and Caltech-256 are similar for low feature spaces but overfits for high feature spaces.

The training times are almost identical to those of the polynomial SVM, only with better worst case, it is easier to fit the data to a more flexible decision rule.

Pros

- Faster training than linear kernel
- Handles large classes well
- Potential complex decision rules

Cons

- Prone to overfit
- Needs many samples per class
- Overfits data in large feature spaces

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	0.14	30.00	63.51	0.76	0.75	0.76	0.80	0.76	1.00
JCD	168	0.03	22.83	62.09	0.70	0.67	0.70	0.69	0.70	1.00
FCTH	192	0.02	24.14	62.00	0.67	0.64	0.67	0.65	0.67	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.02	15.57	62.78	0.01	0.00	0.01	0.00	0.01	1.00
Avg.	199	0.11	25.01	64.90	0.36	0.34	0.36	0.36	0.36	1.00
ACC, FCTH	448	0.16	44.17	64.02	0.81	0.80	0.81	0.83	0.81	1.00
ACC, JCD	424	0.17	42.42	63.68	0.80	0.79	0.80	0.83	0.80	1.00
ACC, CEDD	400	0.17	40.60	63.96	0.80	0.79	0.80	0.83	0.80	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
FCTH, Tamura	210	0.09	24.92	71.79	0.05	0.04	0.05	0.05	0.05	1.00
Avg.	398	0.22	39.06	67.09	0.39	0.37	0.39	0.39	0.39	1.00
5 h.p. ft.	1151	0.29	95.31	68.35	0.45	0.44	0.45	0.49	0.45	1.00
All 19 ft.	3778	2.13	276.34	85.28	0.52	0.49	0.52	0.52	0.52	1.00

(a) UKBench dataset

Table 4.10: Execution times and performances for different combinations of global features using a **SVC RBF** classifier

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
CEDD	144	0.31	0.64	1.50	0.20	0.13	0.20	0.12	0.20	1.00
FOH	576	0.91	1.42	1.88	0.13	0.11	0.17	0.13	0.13	0.99
JCD	168	0.47	0.66	1.33	0.15	0.08	0.15	0.07	0.15	0.99
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
JCH	192	5.13	0.71	1.45	0.01	0.00	0.00	0.00	0.01	0.99
Avg.	199	1.66	0.73	1.48	0.06	0.04	0.08	0.05	0.06	0.99
CL, JCD	201	0.73	0.74	1.36	0.23	0.19	0.24	0.22	0.23	0.99
CL, FCTH	225	0.57	0.79	1.39	0.24	0.19	0.24	0.21	0.24	0.99
CEDD, CL	177	0.57	0.70	1.32	0.21	0.19	0.23	0.21	0.21	0.99
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
JCH, SCH	256	5.55	0.82	1.24	0.01	0.00	0.00	0.00	0.01	0.99
Avg.	398	3.31	1.09	1.70	0.05	0.04	0.08	0.05	0.05	0.99
5 h.p. ft.	1151	4.35	2.45	2.58	0.01	0.01	0.04	0.02	0.01	0.99
All 19 ft.	3778	31.48	7.22	4.69	0.01	0.00	0.00	0.00	0.01	0.99

(b) Holidays dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
EH	80	0.01	58.78	6.19	0.17	0.15	0.17	0.26	0.17	0.99
CEDD	144	0.01	56.29	7.16	0.17	0.12	0.16	0.13	0.17	0.99
JCD	168	0.02	60.80	7.45	0.17	0.11	0.16	0.12	0.17	0.99
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Tamura	18	0.05	26.72	5.47	0.03	0.00	0.02	0.03	0.03	0.97
Avg.	199	0.06	111.66	8.94	0.08	0.04	0.08	0.08	0.08	0.98
EH, JCD	248	0.03	117.40	8.57	0.28	0.26	0.28	0.31	0.28	1.00
EH, FCTH	272	0.03	122.26	8.91	0.28	0.26	0.27	0.30	0.28	1.00
CEDD, EH	224	0.03	111.66	8.25	0.28	0.26	0.27	0.32	0.28	0.99
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
SC, Tamura	82	0.06	61.61	7.97	0.03	0.00	0.02	0.05	0.03	0.97
Avg.	398	0.12	210.34	11.98	0.06	0.04	0.07	0.09	0.06	0.98
5 h.p. ft.	1151	0.16	579.63	22.39	0.04	0.02	0.08	0.06	0.04	0.97
All 19 ft.	3778	1.12	1894.46	64.28	0.03	0.01	0.03	0.05	0.03	0.97

(c) Caltech-256 dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	0.36	5.67	0.97	0.72	0.72	0.68	0.72	0.72	0.96
CEDD	144	0.06	0.90	0.47	0.72	0.71	0.68	0.72	0.72	0.96
JCD	168	0.09	1.00	0.55	0.71	0.71	0.67	0.71	0.71	0.96
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Tamura	18	0.11	1.25	0.18	0.13	0.03	0.03	0.58	0.13	0.88
Avg.	199	0.29	4.44	0.83	0.45	0.43	0.40	0.56	0.45	0.92
CL, JCD	201	0.14	2.42	0.81	0.75	0.75	0.72	0.75	0.75	0.96
CEDD, CL	177	0.11	2.55	0.76	0.75	0.75	0.72	0.75	0.75	0.96
EH, FCTH	272	0.13	2.86	1.11	0.75	0.74	0.71	0.75	0.75	0.96
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
SCH, Tamura	82	0.18	4.58	0.64	0.13	0.03	0.04	0.50	0.13	0.88
Avg.	398	0.58	10.80	1.83	0.43	0.41	0.38	0.60	0.43	0.92
5 h.p. ft.	1151	0.75	32.60	4.77	0.40	0.42	0.38	0.65	0.40	0.91
All 19 ft.	3778	5.50	107.40	15.30	0.16	0.09	0.12	0.76	0.16	0.88

(d) Kvasir dataset

Table 4.10: Execution times and performances for different combinations of global features using a **SVC RBF** classifier (cont.)

4.3.10 Naive Bayes

The naive Bayes classifiers have almost no parameters. The only important one is what data distribution that is predicted, which is set to the Gaussian distribution.

The results are shown in Table 4.11. If we compare it to kNN, the first we will note is that the results of naive Bayes are equal or slightly inferior. Compared to the other classifiers, it scores about average over all datasets and metrics.

What makes the algorithm naive is the independence assumption, that the features are independent from each other, and it will perform poorly if this is not met.

When it comes to speed we can see that the training times are very low, almost as low as kNN. The prediction times is correlated with number of classes and is also low. For Kvasir, with only eight classes, it is almost negligible.

Pros

- No parameters
- Very fast prediction
- Handles large classes well
- Works well in high dimentions

Cons

- Restricted decision boundary
- Require independent features
- Disturbed by noise

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
FCTH	192	0.02	0.48	2.59	0.31	0.32	0.31	0.39	0.31	1.00
CEDD	144	0.02	0.47	2.10	0.30	0.32	0.30	0.38	0.30	1.00
JCD	168	0.03	0.48	2.46	0.25	0.27	0.25	0.32	0.25	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
JCH	192	0.32	0.48	3.29	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	199	0.11	0.48	3.03	0.10	0.10	0.11	0.12	0.10	1.00
ACC, LBP	512	0.16	0.51	8.08	0.44	0.47	0.45	0.56	0.44	1.00
ACC, RILBP	292	0.18	0.49	4.62	0.40	0.42	0.40	0.50	0.40	1.00
ACC, EH	336	0.17	0.50	5.35	0.38	0.41	0.39	0.50	0.38	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
JCH, LL	256	0.35	0.49	4.38	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	398	0.22	0.50	7.28	0.12	0.12	0.12	0.15	0.12	1.00
5 h.p. ft.	1151	0.29	0.54	32.10	0.27	0.28	0.27	0.35	0.27	1.00
All 19 ft.	3778	2.13	0.72	100.51	0.41	0.42	0.41	0.50	0.41	1.00

(a) UKBench dataset

Table 4.11: Execution times and performances for different combinations of global features using a **Naive Bayes** classifier

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
FCTH	192	0.31	0.06	0.49	0.23	0.19	0.23	0.20	0.23	1.00
CEDD	144	0.31	0.06	0.42	0.20	0.17	0.21	0.19	0.20	1.00
JCD	168	0.47	0.06	0.48	0.21	0.16	0.21	0.19	0.21	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
EH	80	0.40	0.06	0.29	0.04	0.01	0.03	0.01	0.04	0.99
Avg.	199	1.66	0.06	0.59	0.10	0.07	0.10	0.08	0.10	1.00
CEDD, FCTH	336	0.62	0.07	0.85	0.25	0.20	0.25	0.22	0.25	1.00
FCTH, JCD	360	0.78	0.07	0.90	0.22	0.18	0.22	0.21	0.22	1.00
CEDD, JCD	312	0.78	0.07	0.83	0.22	0.18	0.22	0.20	0.22	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
EH, Gabor	140	0.55	0.06	0.48	0.04	0.01	0.03	0.01	0.04	0.99
Avg.	398	3.31	0.07	1.15	0.09	0.06	0.09	0.07	0.09	0.99
5 h.p. ft.	1151	4.35	0.08	3.75	0.04	0.03	0.05	0.04	0.04	0.99
All 19 ft.	3778	31.48	0.10	20.02	0.09	0.07	0.09	0.09	0.09	0.99

(b) Holidays dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
PHOG	630	0.04	0.15	1.76	0.16	0.14	0.15	0.17	0.16	1.00
EH	80	0.01	0.09	0.14	0.15	0.14	0.15	0.15	0.15	1.00
CL	33	0.01	0.09	0.06	0.13	0.11	0.13	0.11	0.13	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
FOH	576	0.03	0.15	1.38	0.02	0.01	0.01	0.02	0.02	1.00
Avg.	199	0.06	0.10	0.44	0.05	0.04	0.05	0.05	0.05	1.00
CL, EH	113	0.02	0.09	0.20	0.22	0.20	0.21	0.22	0.22	1.00
Gabor, PHOG	690	0.05	0.16	1.94	0.18	0.16	0.17	0.18	0.18	1.00
CL, PHOG	663	0.05	0.16	1.88	0.17	0.16	0.17	0.19	0.17	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
FOH, OH	640	0.04	0.15	1.55	0.02	0.01	0.02	0.03	0.02	1.00
Avg.	398	0.12	0.13	1.01	0.06	0.05	0.06	0.08	0.06	1.00
5 h.p. ft.	1151	0.16	0.21	3.25	0.10	0.10	0.10	0.18	0.10	1.00
All 19 ft.	3778	1.12	0.48	10.34	0.13	0.13	0.13	0.15	0.13	1.00

(c) Caltech-256 dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
CL	33	0.05	0.01	0.00	0.67	0.66	0.62	0.66	0.67	0.95
ACC	256	0.36	0.01	0.01	0.57	0.52	0.52	0.59	0.57	0.94
SC	64	0.07	0.01	0.00	0.53	0.51	0.46	0.54	0.53	0.93
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
LBP	256	0.07	0.01	0.01	0.23	0.13	0.16	0.26	0.23	0.89
Avg.	199	0.29	0.01	0.01	0.42	0.37	0.36	0.45	0.42	0.92
CL, EH	113	0.13	0.01	0.01	0.68	0.67	0.63	0.67	0.68	0.95
CL, Tamura	51	0.17	0.01	0.00	0.65	0.65	0.61	0.65	0.65	0.95
CL, SC	97	0.12	0.01	0.00	0.64	0.63	0.59	0.64	0.64	0.95
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
EH, LBP	336	0.15	0.01	0.02	0.24	0.14	0.17	0.30	0.24	0.89
Avg.	398	0.58	0.01	0.02	0.49	0.44	0.43	0.52	0.49	0.93
5 h.p. ft.	1151	0.75	0.03	0.10	0.62	0.59	0.58	0.66	0.62	0.95
All 19 ft.	3778	5.50	0.12	0.32	0.64	0.63	0.59	0.64	0.64	0.95

(d) Kvasir dataset

Table 4.11: Execution times and performances for different combinations of global features using a **Naive Bayes** classifier (cont.)

4.3.11 QDA

Similarly to Naive Bayes, QDA has almost no parameters.

Table 4.12 shows the QDA results. The results are extremely bad for all except the Kvasir dataset. The QDA makes an assumption of the predictors, or classes, to be independent and not correlated.

Internally sparse classes makes for trouble for the QDA while preforming a matrix inversion.

On the Kvasir dataset, however, both the training times, the prediction times, and the prediction performance is very low.

The QDA simply does not work well for sparse classes, thus, it gives us little data to analyze, and it is hard for us to draw any clear conclusions.

The QDA is

Pros

- Fast training
- Fast predictions

Cons

- Requires features not to be correlated, thus, having many small classes can be a problem

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
Gabor	60	0.02	0.42	0.39	0.01	0.01	0.01	0.01	0.01	1.00
OH	64	0.03	0.42	0.40	0.00	0.00	0.00	0.00	0.00	1.00
ACC	256	0.14	0.47	1.38	0.00	0.00	0.00	0.00	0.00	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
OH	64	0.03	0.42	0.40	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	199	0.11	0.45	1.11	0.00	0.00	0.00	0.00	0.00	1.00
JCH, LBP	448	0.34	0.50	2.43	0.00	0.00	0.00	0.00	0.00	1.00
PHOG, Tamura	648	0.13	0.54	3.51	0.00	0.00	0.00	0.00	0.00	1.00
JCH, Tamura	210	0.38	0.49	1.22	0.00	0.00	0.00	0.00	0.00	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
OH, RILBP	100	0.07	0.43	0.54	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	398	0.22	0.49	2.12	0.00	0.00	0.00	0.00	0.00	1.00
5 h.p. ft.	1151	0.29	0.62	6.07	0.00	0.00	0.00	0.00	0.00	1.00
All 19 ft.	3778	2.13	1.12	29.18	0.00	0.00	0.00	0.00	0.00	1.00

(a) UKBench dataset

Table 4.12: Execution times and performances for different combinations of global features using a **QDA** classifier

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
CL	33	0.01	0.26	0.05	0.14	0.13	0.14	0.15	0.14	0.99
Tamura	18	0.05	0.14	0.03	0.11	0.10	0.11	0.09	0.11	1.00
EH	80	0.01	0.78	0.12	0.09	0.04	0.08	0.06	0.09	0.99
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.01	0.47	0.60	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	199	1.66	0.08	0.22	0.00	0.00	0.00	0.00	0.00	1.00
OH, RILBP	100	1.03	0.08	0.14	0.00	0.00	0.00	0.00	0.00	1.00
OH, SCH	128	0.85	0.08	0.16	0.00	0.00	0.00	0.00	0.00	1.00
OH, Tamura	82	0.85	0.08	0.12	0.00	0.00	0.00	0.00	0.00	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
OH, RILBP	100	1.03	0.08	0.14	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	398	3.31	0.09	0.38	0.00	0.00	0.00	0.00	0.00	1.00
5 h.p. ft.	1151	4.35	0.23	1.14	0.00	0.00	0.00	0.00	0.00	1.00
All 19 ft.	3778	31.48	0.28	3.81	0.00	0.00	0.00	0.00	0.00	1.00

(b) Holidays dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
CL	33	0.01	0.26	0.05	0.14	0.13	0.14	0.15	0.14	0.99
Tamura	18	0.05	0.14	0.03	0.11	0.10	0.11	0.09	0.11	1.00
EH	80	0.01	0.78	0.12	0.09	0.04	0.08	0.06	0.09	0.99
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.01	0.47	0.60	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	199	0.06	0.81	0.29	0.04	0.02	0.03	0.03	0.04	0.99
CL, Tamura	51	0.06	0.61	0.08	0.12	0.09	0.11	0.12	0.12	0.99
RILBP, Tamura	54	0.07	0.54	0.09	0.09	0.05	0.08	0.05	0.09	0.99
LL, Tamura	82	0.07	0.72	0.12	0.08	0.04	0.07	0.05	0.08	0.99
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
ACC, OH	320	0.08	0.47	0.60	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	398	0.12	1.23	0.51	0.02	0.01	0.02	0.01	0.02	0.99
5 h.p. ft.	1151	0.16	2.42	1.58	0.01	0.01	0.00	0.01	0.01	1.00
All 19 ft.	3778	1.12	5.31	5.07	0.01	0.01	0.00	0.01	0.01	1.00

(c) Caltech-256 dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
CL	33	0.05	0.02	0.00	0.71	0.71	0.67	0.71	0.71	0.96
FCH	125	2.47	0.10	0.01	0.68	0.68	0.64	0.68	0.68	0.95
JCH	192	0.87	0.21	0.01	0.57	0.56	0.50	0.57	0.57	0.94
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
SC	64	0.07	0.05	0.00	0.13	0.04	0.04	0.15	0.13	0.88
Avg.	199	0.29	0.23	0.01	0.36	0.29	0.30	0.36	0.36	0.91
CL, FCH	158	2.52	0.14	0.01	0.71	0.71	0.66	0.71	0.71	0.96
CL, EH	113	0.13	0.09	0.01	0.69	0.70	0.65	0.70	0.69	0.96
EH, FCH	205	2.55	0.24	0.01	0.68	0.68	0.63	0.68	0.68	0.95
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
FCTH, Tamura	210	0.17	0.47	2.22	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	398	0.58	0.50	0.07	0.31	0.23	0.24	0.33	0.31	0.90
5 h.p. ft.	1151	0.75	1.64	0.11	0.19	0.15	0.08	0.13	0.19	0.88
All 19 ft.	3778	5.50	3.36	0.36	0.30	0.23	0.22	0.20	0.30	0.90

(d) Kvasir dataset

Table 4.12: Execution times and performances for different combinations of global features using a **QDA** classifier (cont.)

4.3.12 Stochastic Gradient Decent

The SGD classifier is a highly parametrized classifier, but we will not go into detail about the spesifics.

From Table 4.13 we can see that the prediction times are low on all datasets, and is one of the fastest we have tested.

The training times are also quite low, although, with some variance. For the Kvasir and the Holidays datasets, we have very quick training times, while the many classed UKBench dataset is the slowest. The complexity of the training is linear to both the number of training samples and the feature space length, making the classifier scale well.

The prediction results are decent but compared to the other classifiers below average on all metrics.

Pros

- Fast predictions
- Handles large classes well
- Works well in high dimensions

Cons

- Many parameters
- Easily disturbed by noise

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	0.14	21.92	0.04	0.27	0.26	0.27	0.29	0.27	1.00
JCD	168	0.03	14.35	0.03	0.14	0.11	0.14	0.13	0.14	1.00
CEDD	144	0.02	12.82	0.03	0.13	0.10	0.13	0.12	0.13	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.02	6.66	0.03	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	199	0.11	16.89	0.03	0.04	0.04	0.04	0.04	0.04	1.00
ACC, EH	336	0.17	27.88	0.04	0.39	0.38	0.39	0.42	0.39	1.00
ACC, BPP	1012	0.37	73.73	0.09	0.34	0.33	0.34	0.38	0.34	1.00
ACC, SC	320	0.16	26.73	0.04	0.34	0.31	0.34	0.34	0.34	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
EH, RILBP	116	0.07	10.43	0.03	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	398	0.22	31.34	0.05	0.05	0.04	0.05	0.05	0.05	1.00
5 h.p. ft.	1151	0.29	83.25	0.09	0.26	0.24	0.26	0.28	0.26	1.00
All 19 ft.	3778	2.13	258.21	0.26	0.11	0.09	0.11	0.10	0.11	1.00

(a) UKBench dataset

Table 4.13: Execution times and performances for different combinations of global features using a **SGDClassifier** classifier

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
JCD	168	0.47	0.44	0.01	0.21	0.17	0.21	0.18	0.21	1.00
ACC	256	2.17	0.59	0.01	0.18	0.16	0.18	0.18	0.18	1.00
CEDD	144	0.31	0.40	0.01	0.13	0.11	0.14	0.14	0.13	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.15	0.25	0.01	0.00	0.00	-0.00	0.00	0.00	1.00
Avg.	199	1.66	0.49	0.01	0.05	0.04	0.06	0.05	0.05	1.00
ACC, EH	336	2.58	0.73	0.01	0.24	0.23	0.25	0.28	0.24	1.00
ACC, FCTH	448	2.49	0.92	0.01	0.24	0.21	0.24	0.24	0.24	1.00
ACC, CEDD	400	2.48	0.84	0.01	0.22	0.20	0.22	0.23	0.22	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
FCH, JCD	293	14.38	0.66	0.01	0.00	0.00	0.00	0.00	0.00	1.00
Avg.	398	3.31	0.84	0.01	0.05	0.04	0.05	0.04	0.05	1.00
5 h.p. ft.	1151	4.35	2.23	0.03	0.15	0.13	0.16	0.14	0.15	1.00
All 19 ft.	3778	31.48	7.45	0.07	0.04	0.04	0.08	0.05	0.04	1.00

(b) Holidays dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
PHOG	630	0.04	14.67	0.01	0.12	0.11	0.12	0.19	0.12	1.00
BPP	756	0.11	17.20	0.01	0.10	0.09	0.10	0.15	0.10	1.00
EH	80	0.01	3.25	0.00	0.09	0.09	0.09	0.14	0.09	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.01	2.35	0.00	0.00	0.00	-0.00	0.00	0.00	1.00
Avg.	199	0.06	5.75	0.00	0.04	0.03	0.04	0.06	0.04	0.99
CL, PHOG	663	0.05	15.30	0.01	0.16	0.16	0.16	0.29	0.16	1.00
ACC, PHOG	886	0.11	19.84	0.01	0.16	0.16	0.16	0.28	0.16	1.00
CEDD, EH	224	0.03	7.08	0.00	0.16	0.16	0.16	0.27	0.16	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
JCD, Tamura	186	0.07	6.26	0.00	0.01	0.00	0.00	0.01	0.01	1.00
Avg.	398	0.12	10.01	0.08	0.06	0.05	0.06	0.09	0.06	1.00
5 h.p. ft.	1151	0.16	25.05	0.02	0.15	0.16	0.16	0.33	0.15	1.00
All 19 ft.	3778	1.12	76.66	0.05	0.11	0.11	0.13	0.21	0.11	1.00

(c) Caltech-256 dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
CEDD	144	0.06	0.04	0.00	0.66	0.64	0.62	0.68	0.66	0.95
FCTH	192	0.06	0.05	0.00	0.62	0.58	0.57	0.66	0.62	0.95
ACC	256	0.36	0.07	0.00	0.60	0.58	0.55	0.69	0.60	0.94
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.04	0.02	0.00	0.13	0.03	0.03	0.08	0.13	0.88
Avg.	199	0.29	0.05	0.00	0.43	0.39	0.38	0.50	0.43	0.92
ACC, EH	336	0.44	0.08	0.00	0.72	0.70	0.69	0.76	0.72	0.96
EH, JCD	248	0.17	0.06	0.00	0.71	0.70	0.67	0.72	0.71	0.96
CEDD, FCTH	336	0.11	0.09	0.00	0.68	0.68	0.65	0.74	0.68	0.95
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, Tamura	78	0.16	0.03	0.00	0.18	0.09	0.08	0.10	0.18	0.88
Avg.	398	0.58	0.10	0.00	0.51	0.46	0.46	0.59	0.51	0.93
5 h.p. ft.	1151	0.75	0.25	0.01	0.64	0.60	0.60	0.71	0.64	0.95
All 19 ft.	3778	5.50	0.81	0.02	0.63	0.58	0.60	0.70	0.63	0.95

(d) Kvasir dataset

Table 4.13: Execution times and performances for different combinations of global features using a **SGDClassifier** classifier (cont.)

4.3.13 Neural Network

The default NN configuration comes with one hidden layer of size 100. It also has quite a few parameters.

The training times for the NN algorithm are high, in fact, among the highest we have seen. This is especially true for the Caltech-256 dataset, the dataset with the most samples, ending in many learning iterations, and the UKBench dataset, with the most classes, causing the weigth matrix between the hidden layer and the output layer to be large, thus, require more computations.

The prediction times are all over low.

The prediction results some of the better we have seen, and follow right behind those of the Linear SVM and SVM Ploy 2. What is interesting, is that the NN scores poorly when using all 19 available features on the UKBench and Holidays dataset. It seams to overfit the low number of samples per class of these sets quite badly.

Pros

- Fast predictions
- Handles large classes well
- Very flexible

Cons

- Slow training
- Prone to overfit data
- Requires a large training set
- Many parameters to tune
- Hard to understand

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	0.14	54.61	0.10	0.79	0.78	0.79	0.80	0.79	1.00
JCD	168	0.03	181.44	0.10	0.72	0.70	0.72	0.73	0.72	1.00
FCTH	192	0.02	203.65	0.10	0.71	0.69	0.71	0.71	0.71	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.02	298.52	0.10	0.01	0.00	0.01	0.00	0.01	1.00
Avg.	199	0.11	137.88	0.10	0.38	0.35	0.38	0.37	0.38	1.00
ACC, EH	336	0.17	75.81	0.10	0.82	0.81	0.82	0.83	0.82	1.00
ACC, CEDD	400	0.17	83.70	0.10	0.82	0.80	0.82	0.83	0.82	1.00
ACC, SC	320	0.16	49.86	0.10	0.82	0.80	0.82	0.83	0.82	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, LL	124	0.05	158.40	0.10	0.04	0.03	0.04	0.03	0.04	1.00
Avg.	398	0.22	103.59	0.10	0.43	0.40	0.43	0.42	0.43	1.00
5 h.p. ft.	1151	0.29	65.22	0.10	0.40	0.37	0.40	0.39	0.40	1.00
All 19 ft.	3778	2.13	9.50	0.13	0.00	0.00	0.00	0.00	0.00	1.00

(a) UKBench dataset

Table 4.14: Execution times and performances for different combinations of global features using a **Neural Net** classifier

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
JCD	168	0.47	9.38	0.02	0.56	0.51	0.56	0.52	0.56	1.00
FCTH	192	0.31	9.43	0.02	0.55	0.50	0.55	0.51	0.55	1.00
ACC	256	2.17	5.16	0.03	0.55	0.50	0.55	0.50	0.55	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.15	1.08	0.02	0.01	0.00	0.00	0.00	0.01	0.99
Avg.	199	1.66	7.68	0.02	0.31	0.27	0.31	0.28	0.31	1.00
ACC, CEDD	400	2.48	5.05	0.02	0.61	0.56	0.61	0.57	0.61	1.00
CEDD, FCTH	336	0.62	9.97	0.02	0.58	0.54	0.58	0.55	0.58	1.00
ACC, SC	320	2.50	5.30	0.02	0.59	0.54	0.59	0.54	0.59	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor, LL	124	0.54	9.28	0.02	0.06	0.04	0.06	0.04	0.06	1.00
Avg.	398	3.31	7.68	0.03	0.34	0.30	0.34	0.31	0.34	1.00
5 h.p. ft.	1151	4.35	10.15	0.03	0.29	0.26	0.29	0.27	0.29	1.00
All 19 ft.	3778	31.48	6.91	0.04	0.02	0.00	0.01	0.00	0.02	0.99

(b) Holidays dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
EH	80	0.01	102.69	0.01	0.17	0.16	0.17	0.16	0.17	1.00
JCD	168	0.02	107.36	0.01	0.16	0.15	0.15	0.14	0.16	1.00
PHOG	630	0.04	86.18	0.02	0.14	0.14	0.14	0.15	0.14	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.01	39.57	0.01	0.05	0.01	0.04	0.01	0.05	0.98
Avg.	199	0.06	81.36	0.01	0.11	0.09	0.11	0.09	0.11	0.99
CL, EH	113	0.02	52.44	0.01	0.23	0.22	0.23	0.22	0.23	1.00
EH, JCD	248	0.03	112.40	0.01	0.21	0.21	0.21	0.22	0.21	1.00
CEDD, EH	224	0.03	110.66	0.01	0.21	0.21	0.21	0.21	0.21	1.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
FCH, Gabor	185	0.50	11.47	0.01	0.04	0.01	0.03	0.01	0.04	0.98
Avg.	398	0.12	85.48	0.05	0.13	0.12	0.13	0.12	0.13	0.99
5 h.p. ft.	1151	0.16	81.58	0.02	0.20	0.18	0.19	0.18	0.20	1.00
All 19 ft.	3778	1.12	45.33	0.05	0.07	0.03	0.07	0.02	0.07	0.98

(c) Caltech-256 dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	0.36	1.35	0.00	0.73	0.72	0.69	0.73	0.73	0.96
JCD	168	0.09	3.05	0.00	0.72	0.72	0.68	0.72	0.72	0.96
FCTH	192	0.06	2.31	0.00	0.71	0.70	0.67	0.71	0.71	0.96
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Gabor	60	0.04	1.36	0.00	0.30	0.17	0.23	0.12	0.30	0.90
Avg.	199	0.29	2.33	0.00	0.54	0.52	0.48	0.53	0.54	0.93
ACC, CEDD	400	0.42	1.43	0.00	0.75	0.75	0.71	0.75	0.75	0.96
ACC, FCTH	448	0.42	2.40	0.00	0.74	0.74	0.71	0.75	0.74	0.96
ACC, JCD	424	0.45	1.91	0.00	0.74	0.74	0.70	0.75	0.74	0.96
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
LL, Tamura	82	0.20	1.06	0.00	0.36	0.31	0.28	0.37	0.36	0.91
Avg.	398	0.58	2.35	0.00	0.63	0.62	0.58	0.65	0.63	0.95
5 h.p. ft.	1151	0.75	5.51	0.01	0.73	0.73	0.69	0.73	0.73	0.96
All 19 ft.	3778	5.50	4.57	0.04	0.69	0.67	0.66	0.74	0.69	0.96

(d) Kvasir dataset

Table 4.14: Execution times and performances for different combinations of global features using a **Neural Net** classifier (cont.)

4.3.14 Evaluating the Features

Throughout the feature extraction experiments, we can see that certain image features repeatedly performs very good on certain dataseet. In Tables A.1 to A.8, in the appendix, we present full lists of how well the different features performs on each dataset. We see similarities between the features preferred by the UKBench, Holidays and Kvasir dataset. The Caltech-256 dataset, however, has a noticeable different preference. It seams like the Caltech-256 dataset prefers more edge or gradient oriented features, while the three others prefers color oriented. Looking at the feature combinations that preformed best, the results for each dataset was a little more consistent, and it seamed to be a preference towards a mix of edge and color based features. We can also see that adding more features was more helpful for certain datasets.

All in all, which, and how many, features to use profoundly depends on the dataset.

The extraction times also plays a large role in the feature selection. The ACC feature preformed best as a single feature on three of the datasets. However, a slightly worse average scoring feature such as JCD or CEDD might be a better choice as they are extracted in a fraction of the time.

4.3.15 Summary of Feature Extraction Experiments

To conclude our findings from the feature extraction experiments we have a few interesting findings. In Table 4.15 we have compiled the highest scoring feature combination for each classifier according to F1 score.

Firstly, it is hard to draw an absolute conclusion of which classifier is the best due to their individual combinations of strengths and weakness. Looking at the classification scores alone, however, we have classifiers that stands out. The linear and the polynomial SVC are competing for the highest scores, with neural net and the SVC RBF following close.

As we have discussed, the classifiers perform very different in regard to timing and various algorithms performs better on distinct data. Thus, it is easy to imagine scenarios where, e.g., kNN or random forest would be a good choice. Our results from AdaBoots, however, have been awful with no clear advantages.

4.4 CNN Experiments

The other group of experiments are the CNN experiments. Here we will go through our setup and findings.

4.4.1 Configuration

As with the feature classification experiments, we have some choices to make:

- Which models will we use?

Classifier	Feature	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
kNN	ACC, EH	0.17	0.08	3.10	0.61	0.57	0.61	0.60	0.61	1.00
Decision Tree	FCTH	0.02	1.55	0.01	0.33	0.30	0.33	0.32	0.33	1.00
Random Forest	FCTH, OH	0.05	2.79	0.18	0.48	0.44	0.48	0.48	0.48	1.00
AdaBoost	JCD, PHOG	0.10	45.66	5.24	0.00	0.00	0.03	0.00	0.00	1.00
Linear SVC	ACC, CL	0.16	68.38	0.04	0.87	0.85	0.87	0.87	0.87	1.00
SVC Poly 2	ACC, CL	0.16	28.01	62.75	0.88	0.87	0.88	0.88	0.88	1.00
SVC RBF	ACC, FCTH	0.16	44.17	64.02	0.81	0.80	0.81	0.83	0.81	1.00
Naive Bayes	ACC, LBP	0.16	0.51	8.08	0.44	0.47	0.45	0.56	0.44	1.00
SGD	ACC, EH	0.17	27.88	0.04	0.39	0.38	0.39	0.42	0.39	1.00
Neural Net	ACC, EH	0.17	75.81	0.10	0.82	0.81	0.82	0.83	0.82	1.00

(a) UKBench dataset

Classifier	Feature	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
kNN	ACC, CL	2.43	0.01	0.35	0.33	0.28	0.33	0.29	0.33	1.00
Decision Tree	CEDD, FCTH	0.62	0.18	0.00	0.27	0.24	0.27	0.26	0.27	1.00
Random Forest	FCTH	0.31	0.08	0.03	0.38	0.34	0.38	0.35	0.38	1.00
AdaBoost	ACC, JCH	7.30	2.64	0.82	0.02	0.01	0.06	0.01	0.02	0.99
Linear SVC	ACC, LBP	2.48	8.14	0.01	0.70	0.66	0.70	0.67	0.70	1.00
SVC Poly 2	ACC, CL	2.43	0.79	1.59	0.66	0.62	0.66	0.62	0.66	1.00
SVC RBF	CL, JCD	0.73	0.74	1.36	0.23	0.19	0.24	0.22	0.23	0.99
Naive Bayes	CEDD, FCTH	0.62	0.07	0.85	0.25	0.20	0.25	0.22	0.25	1.00
SGD	ACC, EH	2.58	0.73	0.01	0.24	0.23	0.25	0.28	0.24	1.00
Neural Net	ACC, CEDD	2.48	5.05	0.02	0.61	0.56	0.61	0.57	0.61	1.00

(b) Holidays dataset

Classifier	Feature	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
kNN	CEDD, EH	0.03	0.31	5.80	0.20	0.18	0.19	0.22	0.20	1.00
Decision Tree	*All 19 ft.*	1.12	128.35	0.01	0.11	0.11	0.10	0.11	0.11	1.00
Random Forest	*All 19 ft.*	1.12	13.51	0.03	0.13	0.12	0.13	0.14	0.13	1.00
AdaBoost	JCD, LBP	0.03	0.27	2.64	0.08	0.07	0.08	0.08	0.08	1.00
Linear SVC	*All 19 ft.*	1.12	1428.96	0.04	0.29	0.29	0.28	0.29	0.29	1.00
SVC Poly 2	*5 h.p. ft.*	0.16	330.33	21.03	0.30	0.29	0.30	0.30	0.30	1.00
SVC RBF	EH, JCD	0.03	117.40	8.57	0.28	0.26	0.28	0.31	0.28	1.00
Naive Bayes	CL, EH	0.02	0.09	0.20	0.22	0.20	0.21	0.22	0.22	1.00
SGD	CL, PHOG	0.05	15.30	0.01	0.16	0.16	0.16	0.29	0.16	1.00
Neural Net	CL, EH	0.02	52.44	0.01	0.23	0.22	0.23	0.22	0.23	1.00

(c) Caltech-256 dataset

Classifier	Feature	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
kNN	ACC, EH	0.44	0.07	1.89	0.71	0.71	0.67	0.71	0.71	0.96
Decision Tree	*All 19 ft.*	5.50	3.99	0.01	0.66	0.66	0.62	0.66	0.66	0.95
Random Forest	ACC, CL	0.41	0.15	0.00	0.72	0.72	0.68	0.72	0.72	0.96
AdaBoost	FOH, Tamura	0.28	1.69	0.03	0.52	0.52	0.46	0.55	0.52	0.93
Linear SVC	*All 19 ft.*	5.50	35.09	0.02	0.77	0.76	0.74	0.79	0.77	0.97
SVC Poly 2	*All 19 ft.*	5.50	18.00	8.79	0.76	0.76	0.73	0.76	0.76	0.97
SVC RBF	CL, JCD	0.14	2.42	0.81	0.75	0.75	0.72	0.75	0.75	0.96
Naive Bayes	CL, EH	0.13	0.01	0.01	0.68	0.67	0.63	0.67	0.68	0.95
SGD	ACC, EH	0.44	0.08	0.00	0.72	0.70	0.69	0.76	0.72	0.96
Neural Net	ACC, CEDD	0.42	1.43	0.00	0.75	0.75	0.71	0.75	0.75	0.96

(d) Kvasir dataset

Table 4.15: Execution times and performances for the best performing feature combination for each classifier

- Which parameters will we use?
- Which top will we use?
- Which weight initialization will we use?
- Which input size should our models have
- What pre-processing should we use

We want a braod selection of CNN models. This was just what we implemented in Section 3.5.6, and we will experiment with all 14 of them.

As we saw in Section 2.8.1, CNN's have a good amount of parameters that have to be set.

The batch size was set to a default 64. However, some models was not able to run with a batch size of 64, as this is restricted by the amount of available GPU memory. The three DanseNet models and NASNetLarge was, thus, run with batch size of 32. Each model ran for a typical 100 epochs. The resulting parameters, such as activation function, loss function, optimization function and learning rate were set to common default values. They can be found in our Github repository [3], but we will not go into the details about them here.

Regarding the selection of model top, we have chosen to run our experiments with a common custom top. Running each model with one top limits the number of experiments we will have to run, and the common top makes for an easy comparison base.

When it comes to which weight initialization to use, we will start each model with both random weights, as well as weights pre-tranied on the imagenet dataset. It will be interesting to see how the two configurations compares. As to the choice of the imagenet weights, these are commonly used, and avaliable for all our models, except our custom ones.

The default input size of the input images are set to 150 by 150 pixels. However, the models has restrictions on the input size, thus, this size is altered to conform with the models.

We have decided not to use any advanced pre-processing beyond the models default scaling scheme. This is due to time restrictions, and the testing of advanced pre-processing was abandoned.

The total number of CNN experiments then becomes:

$$(14\ [random] + 12\ [imagenet]\ \text{classifiers}) \times 4\ \text{datasets} = 104\ \text{experiments}$$

4.4.2 Evaluating the Results

The evaluation of the experiments will be conducted the same way sa we did for the feature experiments, discussed in Section 4.3.2, where we focus on the relative performances.

The results will be presented in their entirety, grouped by dataset. In addition, we will present the accuracy development after each epoch.

4.4.3 CNN results

The results from the CNN experiments can be seen in Tables 4.16 to 4.19. The first we will note is how large the difference is between the results with different weight initialization. On the UKBench, Holidays and Caltech-256 dataset, the experiments with pre-trained weights scores distinctively outperform the others by a good margin. On the Kvasir dataset, however, the difference is much more subtle. This dataset has classes that do not resemble those of the imagenet dataset, from which the pre-trained weights come from.

As noted before, the bottom layers on the models with pre-trained weights are not trainable. We can imagine this being a larger restraint for the Kvasir dataset, which features are so different. It would be interesting to do experiments with all layers trainable, what is called *finetuning*, of the pre-trained weights.

When looking at which model gets the best classification scores, there is some clear tendencies. The DenseNet models are some of the best overall performers, while the NASNet models are among the worst. We also have models like the VGG ones, that fluctuates between being in the very top and rock bottom.

The narrower VGG models preform markedly better on the Kvasir data. We know that the deeper a model is, the more complex it can be, and the more abstraction it can tolerate. The Kvasir dataset is, in one way, a simpler problem, as the classes are less abstract. This may be the reason why the VGG models preform better here, as we can imagine the much deeper models to overfit the data.

Speaking of training time, CNN's are very slow. With some exceptions, as the NASNetLarge model which is the slowest to train for all experiments, the training times between models are quite constant. Also across the various datasets, the training time is very consistent, something that is a bit surprising.

The prediction times between the various models are very similar and depends mostly on the size of the input image. Difference between models is small, although noticable for the Caltech0256 dataset. We suspect we would have gotten larger differences with individual tuning of the batch size parameter.

In Figure 4.1 we can see how the accuracy develops after each epoch. The heavy black line is the average of the lot. What is interesting is that the models with pre-trained weights reach a plateau after just around 10–30 epochs while the models with randomized weights is still improving at epoch 100. This indicates that the models with randomized weights did not show their full potential, but at much more than 100 epochs, the training time causes the model to be more impractical.

4.4.4 Summary of the CNN Results

There is little variance of inbetween CNN model results across the different datasets. The overall good performing DenseNet models are good scoring

at all metrics for all tested configurations. Thus, it can seam like the choice of choosing a good CNN model is not too troublesome. Selecting one of the DenseNet models appers to be a safe bet.

4.5 Discussion

Although the in-between results of the CNN models was similar, this is definetly not the case when comparing CNN models to feature classifiers.

Compared the two approaches, the CNN models with pre-trained weights crushes the feature extraction experiments in terms of classification scores, especially for the Caltech-256 dataset, being so similar to the one used for pre-training, thus, can be seen to have an artificialy large training set. The models initialized with random weights, however, do not outperform the feature extraction classifiers, but rather places anonymously in the middle.

When comparing training times, we must remember to take feature extraction time into account. This was, as we remember, a very costly task. Using the average feature extraction times from Table 4.2, multiplying by number of training samples and adding the training time, the CNN models roughly end up being more computational heavy by a factor of about five or ten.

Compared to the combined preparation and prediction times for the feature extraction classifiers, the CNN models are faster by a factor of around five to ten.

It is worth noting that the LIRE extraction times was measured on the modest CPU from setup A, while the CNN experiment extracts features within the CNN, taking full advantage of am up-market GPU. If we were to run the LIRE feature extractions in parallel on the eight cores, we could imagine seeing a cut in extraction time by a factor upwards of eight, drastically cutting both the total training and the total prediction times, still on our modest CPU.

The CNN also may have performance improvements. The batch size, set to accommodate the larger models, could have limited the smaller models, which might could have run significantly faster on an optimized batch size. But, on the other hand, we aspired not to tune parameters, and our chosen setting seam like a very sensible choice.

The selection of our dataset was motivated to be as diverse as possible. However, in the grand scheme, the datasets represents a limited subsection of image classification problems, thus, bound to be application specific to comparable problems. The insights gotten are still fruitful to a heftier field.

4.6 Summary

Strongly based on first-rate classification performance, the three SVCs and neural net was the overall best feature classifiers for our tested applications. The linear SVC proved very low prediction times, but high training times for large datasets. The RBF SVC showed lower training times, but with

a trade-off of higher prediction times. The RBF SVC also provides more complex decision rules, are more prone to overfit and needs more data to excel. The SVC Poly 2 lays somewhere in between. The neural networks are comparable to the SVC RBF on both time consumption and their need for larger training sets.

AdaBoost and QDA has shown close to useless performances on our experiments, and we have not found suitable applications for them. Other classifiers, such as kNN and decision trees, has shown promising results on specific data, and has their use cases, although might not as a general go-to approach for our tested application space.

The CNN models with pre-trained weight was the best performing CNN's. Compared to the traditional feature extraction classifiers the CNN models perform on par or better. However, a small increase in prediction performance might not be worth the immense training overhead, thus the CNN models can only be said to be superior on their home soil, with large and abstract classes, preferably with weights pre-trained on a comparable task.

Figure 4.2 compiles this summary into an elementary flowchart.

Model	Train *h*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
VGG16	4.91	16.55	0.00	0.00	0.00	0.00	0.00	1.00
VGG19	5.13	17.03	0.00	0.00	0.00	0.00	0.00	1.00
ResNet50	5.48	17.33	0.00	0.00	0.00	0.00	0.00	1.00
large_net	4.29	15.77	0.00	0.00	0.00	0.00	0.00	1.00
small_net	4.14	15.52	0.00	0.00	0.00	0.00	0.00	1.00
NASNetLarge	11.36	21.85	0.00	0.00	0.00	0.00	0.00	1.00
NASNetMobile	4.69	15.95	0.00	0.00	0.00	0.00	0.00	1.00
InceptionV3	4.34	14.79	0.14	0.12	0.14	0.13	0.14	1.00
MobileNet	4.19	15.36	0.17	0.14	0.17	0.15	0.17	1.00
Xceptron	4.73	15.91	0.22	0.19	0.22	0.20	0.22	1.00
InceptionResNetV2	5.38	16.79	0.37	0.34	0.37	0.36	0.37	1.00
DenseNet121	7.32	22.34	0.59	0.55	0.59	0.59	0.59	1.00
DenseNet169	8.10	21.02	0.60	0.56	0.60	0.59	0.60	1.00
DenseNet201	9.81	23.31	0.60	0.57	0.60	0.60	0.60	1.00

(a) Randomized weights

Model	Train *h*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
NASNetMobile	4.81	18.46	0.30	0.26	0.30	0.29	0.30	1.00
NASNetLarge	19.99	74.27	0.43	0.40	0.43	0.43	0.43	1.00
InceptionV3	5.60	21.71	0.75	0.73	0.75	0.77	0.75	1.00
InceptionResNetV2	7.00	26.62	0.76	0.73	0.76	0.77	0.76	1.00
Xceptron	6.49	25.08	0.89	0.87	0.89	0.90	0.89	1.00
ResNet50	5.09	18.99	0.90	0.89	0.90	0.91	0.90	1.00
MobileNet	4.51	17.58	0.92	0.91	0.92	0.93	0.92	1.00
VGG19	5.51	20.84	0.93	0.92	0.93	0.94	0.93	1.00
VGG16	5.30	19.90	0.93	0.92	0.93	0.94	0.93	1.00
DenseNet121	5.27	20.47	0.94	0.93	0.94	0.94	0.94	1.00
DenseNet201	6.33	23.22	0.95	0.95	0.95	0.96	0.95	1.00
DenseNet169	5.51	21.54	0.96	0.95	0.96	0.96	0.96	1.00

(b) Pretrained weights

Table 4.16: Execution times and performances for different deep learning models and weights on the **UKbench** dataset.

Model	Train *h*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
NASNetLarge	7.40	211.60	0.00	0.00	-0.00	0.00	0.00	1.00
VGG16	6.29	193.83	0.01	0.00	0.00	0.00	0.01	0.99
VGG19	6.16	197.51	0.01	0.00	0.00	0.00	0.01	0.99
NASNetMobile	6.47	207.43	0.01	0.00	0.01	0.00	0.01	1.00
InceptionV3	6.42	207.52	0.05	0.04	0.05	0.04	0.05	1.00
MobileNet	6.23	196.48	0.10	0.08	0.10	0.09	0.10	1.00
large_net	6.32	184.66	0.12	0.09	0.12	0.09	0.12	1.00
small_net	6.33	183.07	0.15	0.11	0.14	0.11	0.15	1.00
ResNet50	6.55	194.22	0.27	0.24	0.27	0.26	0.27	1.00
DenseNet169	7.05	199.22	0.30	0.25	0.30	0.27	0.30	1.00
DenseNet201	7.18	203.01	0.33	0.28	0.33	0.28	0.33	1.00
Xceptron	6.30	193.34	0.32	0.28	0.32	0.29	0.32	1.00
InceptionResNetV2	6.58	206.27	0.32	0.28	0.32	0.29	0.32	1.00
DenseNet121	6.96	199.75	0.35	0.31	0.35	0.33	0.35	1.00

(a) Randomized weights

Model	Train *h*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
NASNetMobile	6.15	206.84	0.18	0.15	0.18	0.18	0.18	1.00
NASNetLarge	7.37	231.35	0.32	0.28	0.32	0.30	0.32	1.00
InceptionV3	6.11	190.69	0.51	0.47	0.51	0.48	0.51	1.00
InceptionResNetV2	6.47	211.46	0.53	0.49	0.53	0.50	0.53	1.00
ResNet50	6.51	223.46	0.61	0.57	0.61	0.59	0.61	1.00
Xceptron	6.68	208.58	0.67	0.62	0.67	0.64	0.67	1.00
DenseNet121	6.34	207.78	0.70	0.66	0.70	0.67	0.70	1.00
MobileNet	6.25	196.32	0.73	0.69	0.73	0.70	0.73	1.00
VGG19	6.13	189.35	0.75	0.71	0.75	0.71	0.75	1.00
VGG16	6.03	198.32	0.75	0.71	0.75	0.71	0.75	1.00
DenseNet201	6.83	230.36	0.75	0.71	0.75	0.73	0.75	1.00
DenseNet169	6.18	205.08	0.76	0.73	0.76	0.74	0.76	1.00

(b) Pretrained weights

Table 4.17: Execution times and performances for different deep learning models and weights on the **Holidays** dataset.

Model	Train *h*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
VGG16	9.45	7.71	0.00	0.00	0.00	0.00	0.00	1.00
small_net	4.84	5.76	0.00	0.00	0.00	0.00	0.00	1.00
VGG19	10.57	7.76	0.01	0.00	0.00	0.00	0.01	0.99
NASNetMobile	7.86	7.47	0.05	0.04	0.05	0.12	0.05	0.99
NASNetLarge	31.19	16.59	0.17	0.16	0.16	0.20	0.17	1.00
large_net	5.31	6.15	0.24	0.23	0.23	0.25	0.24	1.00
MobileNet	6.92	7.25	0.24	0.24	0.24	0.27	0.24	1.00
ResNet50	13.61	10.07	0.25	0.25	0.25	0.26	0.25	1.00
InceptionV3	7.76	7.72	0.26	0.26	0.26	0.29	0.26	1.00
Xceptron	9.32	7.13	0.27	0.27	0.27	0.32	0.27	1.00
DenseNet201	26.75	17.70	0.28	0.27	0.27	0.29	0.28	1.00
DenseNet169	21.36	13.92	0.28	0.28	0.27	0.31	0.28	1.00
InceptionResNetV2	11.65	7.92	0.30	0.29	0.29	0.36	0.30	1.00
DenseNet121	17.93	12.92	0.32	0.31	0.31	0.33	0.32	1.00

(a) Randomized weights

Model	Train *h*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
NASNetMobile	7.11	8.94	0.48	0.50	0.48	0.65	0.48	1.00
NASNetLarge	53.58	68.04	0.73	0.74	0.73	0.78	0.73	1.00
VGG16	8.80	11.83	0.74	0.74	0.74	0.77	0.74	1.00
VGG19	9.81	12.49	0.75	0.75	0.75	0.77	0.75	1.00
ResNet50	8.61	11.44	0.77	0.78	0.77	0.81	0.77	1.00
MobileNet	6.51	8.54	0.77	0.78	0.77	0.80	0.77	1.00
DenseNet121	8.79	11.56	0.80	0.80	0.80	0.82	0.80	1.00
InceptionV3	10.82	13.68	0.81	0.81	0.81	0.84	0.81	1.00
DenseNet169	9.60	12.87	0.81	0.81	0.81	0.83	0.81	1.00
DenseNet201	12.55	16.91	0.82	0.82	0.82	0.84	0.82	1.00
Xceptron	16.01	20.55	0.84	0.84	0.84	0.85	0.84	1.00
InceptionResNetV2	19.87	25.44	0.84	0.84	0.84	0.85	0.84	1.00

(b) Pretrained weights

Table 4.18: Execution times and performances for different deep learning models and weights on the **Caltech-256** dataset.

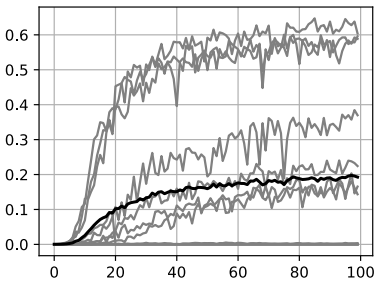
Model	Train *h*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
VGG16	7.89	35.74	0.12	0.03	0.00	0.02	0.12	0.88
NASNetMobile	8.23	36.07	0.12	0.03	0.00	0.02	0.12	0.88
VGG19	8.49	37.45	0.12	0.03	0.00	0.02	0.12	0.88
small_net	7.22	34.16	0.12	0.03	0.00	0.02	0.12	0.88
NASNetLarge	11.30	43.32	0.24	0.11	0.18	0.10	0.24	0.89
MobileNet	6.64	32.97	0.68	0.66	0.64	0.70	0.68	0.95
large_net	7.36	34.35	0.68	0.68	0.63	0.68	0.68	0.95
DenseNet201	10.44	41.66	0.71	0.70	0.68	0.76	0.71	0.96
DenseNet169	9.71	38.79	0.74	0.72	0.71	0.77	0.74	0.96
DenseNet121	9.14	37.58	0.73	0.73	0.70	0.77	0.73	0.96
ResNet50	8.42	37.85	0.75	0.74	0.72	0.76	0.75	0.96
Xceptron	7.69	31.33	0.81	0.81	0.78	0.81	0.81	0.97
InceptionV3	8.19	37.13	0.81	0.81	0.79	0.82	0.81	0.97
InceptionResNetV2	7.93	35.06	0.83	0.83	0.80	0.83	0.83	0.98

(a) Randomized weights

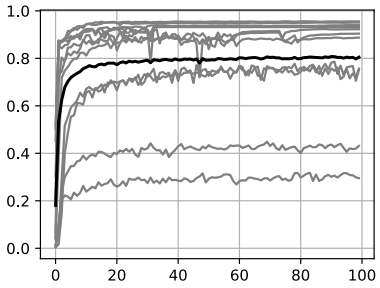
Model	Train *h*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
NASNetMobile	8.33	37.36	0.41	0.33	0.35	0.60	0.41	0.92
NASNetLarge	14.46	68.77	0.40	0.33	0.34	0.51	0.40	0.91
InceptionV3	8.97	42.94	0.51	0.45	0.46	0.65	0.51	0.93
MobileNet	8.01	38.60	0.49	0.46	0.45	0.72	0.49	0.93
ResNet50	8.41	38.62	0.57	0.53	0.54	0.76	0.57	0.94
DenseNet121	8.79	41.07	0.66	0.61	0.63	0.73	0.66	0.95
Xceptron	9.58	47.17	0.63	0.63	0.59	0.69	0.63	0.95
InceptionResNetV2	10.31	49.82	0.67	0.64	0.63	0.70	0.67	0.95
DenseNet169	8.70	41.75	0.69	0.68	0.66	0.76	0.69	0.96
DenseNet201	8.94	42.52	0.73	0.72	0.69	0.75	0.73	0.96
VGG19	8.35	39.51	0.85	0.85	0.83	0.85	0.85	0.98
VGG16	8.55	40.96	0.86	0.86	0.84	0.86	0.86	0.98

(b) Pretrained weights

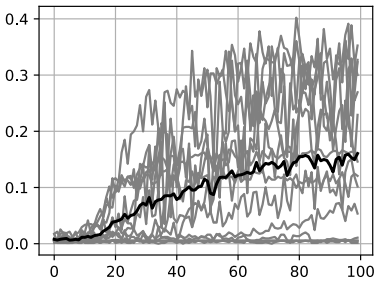
Table 4.19: Execution times and performances for different deep learning models and weights on the **Kvasir** dataset.



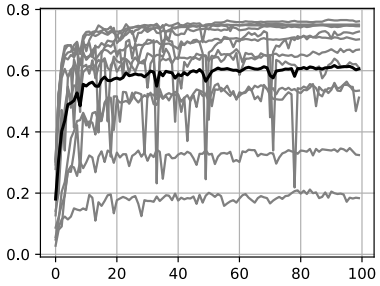
(a) UKBench dataset with randomized weigths



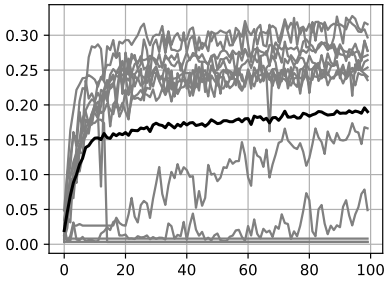
(b) UKBench dataset with pre-trained weigths



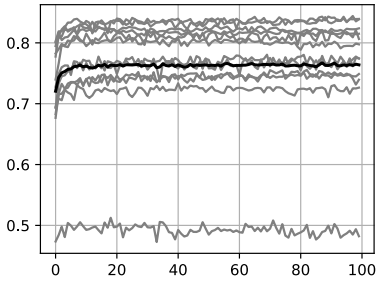
(c) Holidays dataset with randomized weigths



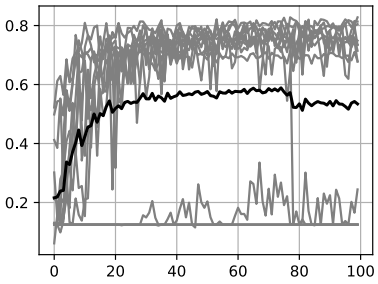
(d) Holidays dataset with pre-trained weigths



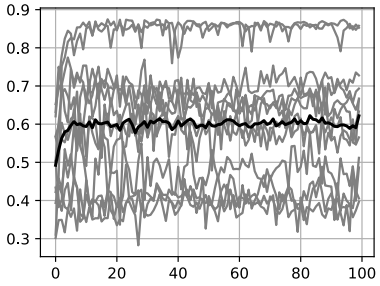
(e) Caltech-256 dataset with randomized weigths



(f) Caltech-256 dataset with pre-trained weigths



(g) Kvasir dataset with randomized weigths



(h) Kvasir dataset with pre-trained weigths

Figure 4.1: Accuracy development for various datasets and weight initialization

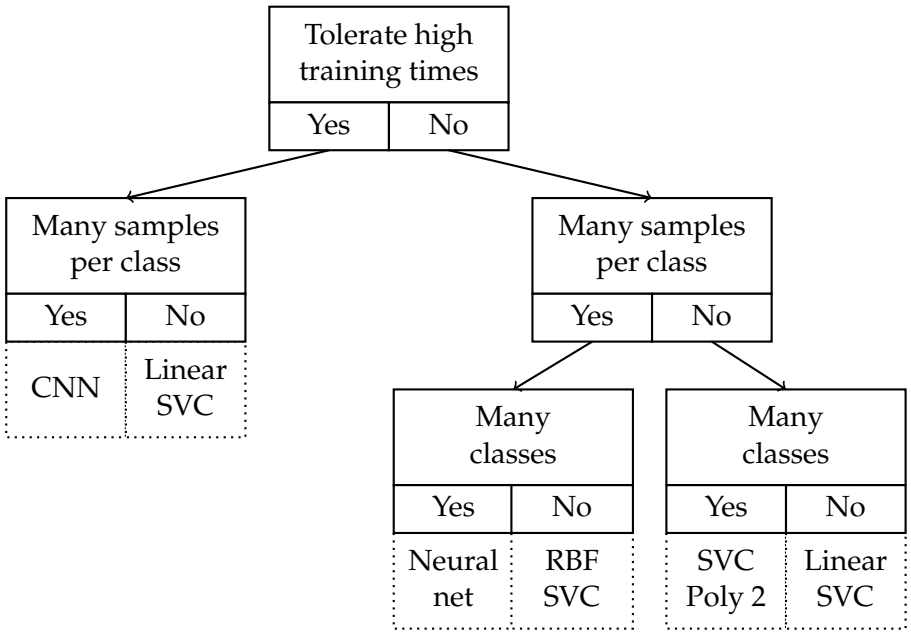


Figure 4.2: Elementary flowchart for choosing classifier

Chapter 5

Conclusion

In this thesis we wanted to compare deep learning image classification methods to traditional methods based on feature extraction, and explore the differentiating factors. This was tackled by (1) implementing an automated testing pipeline, (2) running a large set of experiments with diverse configurations, (3) using the results to analyse the classifiers, and (4) coming up with a verdict on which classifier to use given various problems.

We have implemented an automatic testing pipeline for multiple classifiers with different approaches, abstracted for ease of use, flexible, easily expandable, and presented online [3]. Furthermore, we have run and analyzed a vast set of classification experiments, culminating in a conclusion on which classification approach to use, given a specific tasks, nicely visualized as a flowchart.

To conclude, CNN approaches are able to outperform traditional image classes in terms of classification scores, but only if we either have suitable pre-trained weights available or an enormous dataset, and, furthermore, has drawbacks such large processing times and a black-box approach, making them a suboptimal choice for a range of problems.

5.1 Futhure work

We have identified two paths to further extend this thesis.

The first one will be to continue development of the automatic classification pipeline with more classifiers, more features, or performance optimizations. An especially interesting feature that we would like to be implemented, is the possibility to extract CNN features from pre-trained models and classify them by the traditional machine learning algotithms.

The second approach is to extend to the classifier choosing scheme by either introducing new datasets, and thus widen the direct applications of our conclusion, or testing new classifier configurations, such as finetuning of CNN's or the affect of advanced pre-processing.

Bibliography

[1] Martın Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "TensorFlow: A System for Large-Scale Machine Learning." In: *OSDI*. Vol. 16. 2016, pp. 265–283.

[2] Apple. *A11 Bionic*. 2017. URL: https://www.apple.com/iphone-8/#a11 (visited on 02/26/2018).

[3] Edvard Johannesen Bakken. *Image Classifier*. 2018. URL: https://github.com/eBakken/image_classifier (visited on 05/16/2018).

[4] Sabri Boughorbel, Fethi Jarray, and Mohammed El-Anbari. "Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric". In: *PloS one* 12.6 (2017), e0177678.

[5] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[6] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[7] Rich Caruana and Alexandru Niculescu-Mizil. "An empirical comparison of supervised learning algorithms". In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 161–168.

[8] Christine Chin and David E. Brown. "Learning in science: A comparison of deep and surface approaches". In: *Journal of research in science teaching* 37 (2 2000), pp. 109–138.

[9] François Chollet et al. *Keras*. https://github.com/fchollet/keras. 2015.

[10] François Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *arXiv preprint* (2016).

[11] Douglas E Comer, David Gries, Michael C Mulder, Allen Tucker, A Joe Turner, Paul R Young, and Peter J Denning. "Computing as a discipline". In: *Communications of the ACM* 32.1 (1989), pp. 9–23.

[12] Thomas Cover and Peter Hart. "Nearest neighbor pattern classification". In: *IEEE transactions on information theory* 13.1 (1967), pp. 21–27.

[13] Janez Demšar. "Statistical comparisons of classifiers over multiple data sets". In: *Journal of Machine learning research* 7.Jan (2006), pp. 1–30.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 248–255.

[15] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. "Decaf: A deep convolutional activation feature for generic visual recognition". In: *International conference on machine learning*. 2014, pp. 647–655.

[16] Forbes. *Why deep learning is suddenly changing your life*. 2016. URL: http://fortune.com/ai-artificial-intelligence-deep-machine-learning/ (visited on 02/26/2018).

[17] Yoav Freund and Robert E Schapire. "A desicion-theoretic generalization of on-line learning and an application to boosting". In: *European conference on computational learning theory*. Springer. 1995, pp. 23–37.

[18] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. Springer series in statistics New York, 2001.

[19] Dario Garcia-Gasulla, Armand Vilalta, Ferran Parés, Jonatan Moreno, Eduard Ayguadé, Jesus Labarta, Ulises Cortés, and Toyotaro Suzumura. "An Out-of-the-box Full-network Embedding for Convolutional Neural Networks". In: *arXiv preprint arXiv:1705.07706* (2017).

[20] Gregory Griffin, Alex Holub, and Pietro Perona. "Caltech-256 object category dataset". In: (2007).

[21] HDF Group et al. "Hierarchical data format, version 5". In: (2014).

[22] Simon Haykin and Neural Network. "A comprehensive foundation". In: *Neural networks* 2.2004 (2004), p. 41.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[24] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[25] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. Vol. 1. 2. 2017, p. 3.

[26] John D Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in science & engineering* 9.3 (2007), pp. 90–95.

[27] Google Inc. *Machine learning - Explore - Google Trends*. URL: https://trends.google.com/trends/explore?q=machine%20learning (visited on 03/02/2017).

[28] Herve Jegou, Matthijs Douze, and Cordelia Schmid. "Hamming embedding and weak geometric consistency for large scale image search". In: *European conference on computer vision*. Springer. 2008, pp. 304–317.

[29] Eric Jones, Travis Oliphant, and Pearu Peterson. "{SciPy}: open source scientific tools for {Python}". In: (2014).

[30] Keras. *Applications*. URL: https://keras.io/applications/ (visited on 04/26/2018).

[31] Ross D. King, Cao Feng, and Alistair Sutherland. "Statlog: comparison of classification algorithms on large real-world problems". In: *Applied Artificial Intelligence an International Journal* 9.3 (1995), pp. 289–333.

[32] UNC Vision Lab. *Large Scale Visual Recognition Challenge 2014 (ILSVRC2014)*. 2014. URL: http://image-net.org/challenges/LSVRC/2014/results (visited on 02/26/2018).

[33] UNC Vision Lab. *Large Scale Visual Recognition Challenge 2015 (ILSVRC2015)*. 2015. URL: http://image-net.org/challenges/LSVRC/2015/results (visited on 02/26/2018).

[34] UNC Vision Lab. *Large Scale Visual Recognition Challenge 2017 (ILSVRC2017)*. 2017. URL: http://image-net.org/challenges/LSVRC/2017/results (visited on 02/26/2018).

[35] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4 (1989), pp. 541–551.

[36] Yann LeCun, LD Jackel, Corinna Cortes, et al. "Learning algorithms for classification: A comparison on handwritten digit recognition". In: *Neural networks: the statistical mechanics perspective* 261 (1995).

[37] Guillaume Lemaıtre, Fernando Nogueira, and Christos K Aridas. "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning". In: *Journal of Machine Learning Research* 18.17 (2017), pp. 1–5.

[38] Mathias Lux. "LIRE: Open Source Image Retrieval in Java". In: *Proc. of ACM MM* (2013).

[39] Mathias Lux and Oge Marques. "Visual information retrieval using java and lire". In: *Synthesis Lectures on Information Concepts, Retrieval, and Services* 5.1 (2013), pp. 1–112.

[40] Raphaël Marée, Pierre Geurts, Giorgio Visimberga, Justus Piater, and Louis Wehenkel. "A comparison of generic machine learning algorithms for image classification". In: *Research and Development in Intelligent Systems XX*. Springer, 2004, pp. 169–182.

[41] Anh Nguyen, Jason Yosinski, and Jeff Clune. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images." In: *arXiv preprint arXiv* 1412 (1897 2014).

[42] David Nister and Henrik Stewenius. "Scalable recognition with a vocabulary tree". In: *Computer vision and pattern recognition, 2006 IEEE computer society conference on*. Vol. 2. Ieee. 2006, pp. 2161–2168.

[43] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[44] Travis E Oliphant. "Python for scientific computing". In: *Computing in Science & Engineering* 9.3 (2007).

[45] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[46] Konstantin Pogorelov, Kristin Ranheim Randel, Carsten Griwodz, Sigrun Losada Eskeland, Thomas de Lange, Dag Johansen, Concetto Spampinato, Duc-Tien Dang-Nguyen, Mathias Lux, Peter Thelin Schmidt, Michael Riegler, and Pål Halvorsen. "KVASIR: A Multi-Class Image Dataset for Computer Aided Gastrointestinal Disease Detection". In: *Proceedings of the 8th ACM on Multimedia Systems Conference*. MMSys'17. Taipei, Taiwan: ACM, 2017, pp. 164–169. ISBN: 978-1-4503-5002-0. DOI: 10.1145/3083187.3083212. URL: http://doi.acm.org/10.1145/3083187.3083212.

[47] Konstantin Pogorelov, Kristin Ranheim Randel, Carsten Griwodz, Sigrun Losada Eskeland, Thomas de Lange, Dag Johansen, Concetto Spampinato, Duc-Tien Dang-Nguyen, Mathias Lux, Peter Thelin Schmidt, et al. "Kvasir: a multi-class image dataset for computer aided gastrointestinal disease detection". In: *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM. 2017, pp. 164–169.

[48] Oxford University Press. *Overfitting*. 2018. URL: https://en.oxforddictionaries.com/definition/overfitting (visited on 04/25/2018).

[49] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. "CNN features off-the-shelf: an astounding baseline for recognition". In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*. IEEE. 2014, pp. 512–519.

[50] MIT Technology Review. *10 Breakthrough Technologies 2014*. 2014. URL: https://www.technologyreview.com/lists/technologies/2014/ (visited on 02/26/2018).

[51] MIT Technology Review. *10 Breakthrough Technologies 2015*. 2015. URL: https://www.technologyreview.com/lists/technologies/2015/ (visited on 02/26/2018).

[52] MIT Technology Review. *10 Breakthrough Technologies 2016*. 2016. URL: https://www.technologyreview.com/lists/technologies/2016/ (visited on 02/26/2018).

[53] MIT Technology Review. *10 Breakthrough Technologies 2017*. 2017. URL: https://www.technologyreview.com/lists/technologies/2017/ (visited on 02/26/2018).

[54] Guido Rossum. *Python Reference Manual*. Tech. rep. Amsterdam, The Netherlands, The Netherlands, 1995.

[55] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[56] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[57] Alex J Smola and Bernhard Schölkopf. "A tutorial on support vector regression". In: *Statistics and computing* 14.3 (2004), pp. 199–222.

[58] Donald F. Specht. "Probabilistic neural networks". In: *Neural networks* 3 (1 1990), pp. 109–118.

[59] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. "Inception-v4, inception-resnet and the impact of residual connections on learning." In: *AAAI*. Vol. 4. 2017, p. 12.

[60] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2818–2826.

[61] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. "Going deeper with convolutions". In: Cvpr. 2015.

[62] Jason Wang and Luis Perez. *The effectiveness of data augmentation in image classification using deep learning*. Tech. rep. Technical report, 2017.

[63] Ren Wu, Shengen Yan, Yi Shan, Qingqing Dang, and Gang Sun. "Deep image: Scaling up image recognition". In: *arXiv preprint arXiv:1501.02876* 7.8 (2015).

[64] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. "How transferable are features in deep neural networks?" In: *Advances in neural information processing systems*. 2014, pp. 3320–3328.

[65] Harry Zhang. "The optimality of naive Bayes". In: *AA* 1.2 (2004), p. 3.

[66] Tong Zhang. "Solving large scale linear prediction problems using stochastic gradient descent algorithms". In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 116.

[67] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. "Learning transferable architectures for scalable image recognition". In: *arXiv preprint arXiv:1707.07012* (2017).

Appendices

Appendix A

Feature Combination Averages

Tables A.1 to A.8 shows the performances of the various feature combinations on the different datasets, each feature combination averaged over our ten feature classifiers.

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	0.14	23.91	13.83	0.49	0.48	0.50	0.51	0.49	1.00
JCD	168	0.03	30.60	13.33	0.44	0.42	0.44	0.44	0.44	1.00
FCTH	192	0.02	32.31	13.35	0.43	0.41	0.43	0.43	0.43	1.00
CEDD	144	0.02	33.91	13.27	0.41	0.39	0.41	0.41	0.41	1.00
JCH	192	0.32	37.62	13.04	0.25	0.23	0.25	0.24	0.25	1.00
CL	33	0.02	27.73	13.04	0.23	0.21	0.24	0.23	0.23	1.00
SCH	64	0.02	18.61	14.02	0.23	0.21	0.23	0.23	0.23	1.00
OH	64	0.03	18.75	14.78	0.21	0.19	0.21	0.20	0.21	1.00
FOH	576	0.06	30.24	14.78	0.20	0.18	0.21	0.19	0.20	1.00
SC	64	0.02	29.95	14.79	0.17	0.16	0.18	0.17	0.17	1.00
FCH	125	0.94	33.79	14.09	0.17	0.15	0.17	0.16	0.17	1.00
LBP	256	0.02	42.60	13.55	0.16	0.14	0.16	0.15	0.16	1.00
PHOG	630	0.07	48.83	15.21	0.13	0.12	0.13	0.13	0.13	1.00
RILBP	36	0.04	36.01	13.99	0.14	0.12	0.14	0.12	0.14	1.00
EH	80	0.03	26.32	13.17	0.13	0.12	0.13	0.13	0.13	1.00
BPP	756	0.23	50.82	15.43	0.13	0.12	0.13	0.12	0.13	1.00
Tamura	18	0.06	32.03	12.78	0.03	0.02	0.03	0.02	0.03	1.00
LL	64	0.03	75.37	13.29	0.02	0.01	0.02	0.01	0.02	1.00
Gabor	60	0.02	57.61	13.19	0.01	0.01	0.01	0.01	0.01	1.00

Table A.1: Average performances over different feature combinations on the **UKBench** dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC, EH	336	0.17	29.36	14.11	0.53	0.51	0.53	0.55	0.53	1.00
ACC, CEDD	400	0.17	31.49	14.22	0.53	0.51	0.53	0.54	0.53	1.00
ACC, CL	289	0.16	28.71	13.87	0.52	0.51	0.53	0.54	0.52	1.00
ACC, JCD	424	0.17	32.89	14.28	0.52	0.51	0.52	0.54	0.52	1.00
ACC, FCTH	448	0.16	31.70	14.35	0.52	0.51	0.52	0.54	0.52	1.00
ACC, SC	320	0.16	27.96	13.95	0.52	0.51	0.52	0.53	0.52	1.00
ACC, Gabor	316	0.16	47.20	15.72	0.51	0.49	0.51	0.52	0.51	1.00
ACC, RILBP	292	0.18	31.99	13.74	0.47	0.46	0.48	0.48	0.47	1.00
ACC, LBP	512	0.16	40.75	14.44	0.47	0.45	0.47	0.48	0.47	1.00
CEDD, FCTH	336	0.05	35.81	13.84	0.47	0.45	0.47	0.48	0.47	1.00
ACC, FOH	832	0.20	34.53	17.69	0.46	0.44	0.46	0.47	0.46	1.00
FCTH, JCD	360	0.06	28.45	13.87	0.46	0.44	0.46	0.46	0.46	1.00
ACC, BPP	1012	0.37	49.19	17.57	0.45	0.43	0.45	0.46	0.45	1.00
CEDD, JCD	312	0.06	27.42	13.77	0.44	0.42	0.44	0.44	0.44	1.00
ACC, OH	320	0.17	25.59	15.26	0.43	0.41	0.43	0.44	0.43	1.00
ACC, LL	320	0.18	29.54	14.04	0.41	0.39	0.41	0.41	0.41	1.00
ACC, SCH	320	0.16	29.78	15.05	0.38	0.36	0.38	0.39	0.38	1.00
CEDD, CL	177	0.04	33.48	13.39	0.38	0.35	0.38	0.38	0.38	1.00
CL, JCD	201	0.05	34.37	13.53	0.37	0.35	0.38	0.37	0.37	1.00
CEDD, SC	208	0.05	30.19	13.44	0.37	0.35	0.38	0.37	0.37	1.00
JCD, SC	232	0.06	33.61	13.48	0.37	0.35	0.37	0.37	0.37	1.00
Gabor, JCD	228	0.05	56.79	13.57	0.36	0.35	0.37	0.37	0.36	1.00
CL, FCTH	225	0.04	33.41	13.55	0.36	0.34	0.36	0.36	0.36	1.00
CEDD, Gabor	204	0.04	51.39	13.53	0.36	0.34	0.36	0.36	0.36	1.00
FCTH, SC	256	0.05	30.84	15.74	0.36	0.34	0.36	0.36	0.36	1.00
CL, SC	97	0.04	28.16	13.18	0.33	0.31	0.33	0.33	0.33	1.00
FCTH, Gabor	252	0.04	56.83	13.60	0.33	0.31	0.33	0.33	0.33	1.00
CEDD, EH	224	0.05	28.16	13.58	0.33	0.31	0.33	0.33	0.33	1.00
ACC, JCH	448	0.46	54.17	14.41	0.33	0.30	0.33	0.32	0.33	1.00
EH, JCD	248	0.06	28.91	13.70	0.32	0.30	0.32	0.32	0.32	1.00
EH, FCTH	272	0.05	28.74	13.75	0.32	0.30	0.32	0.32	0.32	1.00
JCH, SCH	256	0.34	42.23	13.20	0.32	0.29	0.32	0.31	0.32	1.00
ACC, FCH	381	1.08	44.67	15.59	0.31	0.29	0.31	0.32	0.31	1.00
ACC, PHOG	886	0.21	53.36	17.04	0.30	0.29	0.31	0.32	0.30	1.00
FOH, SC	640	0.08	31.66	16.08	0.31	0.29	0.31	0.31	0.31	1.00
CL, EH	113	0.04	33.08	13.25	0.30	0.28	0.30	0.30	0.30	1.00
OH, SC	128	0.05	21.39	13.68	0.30	0.28	0.30	0.30	0.30	1.00
EH, SC	144	0.05	28.32	13.26	0.29	0.27	0.30	0.29	0.29	1.00
SC, SCH	128	0.05	24.31	14.52	0.29	0.27	0.29	0.29	0.29	1.00
JCH, OH	256	0.35	39.81	13.31	0.29	0.27	0.29	0.28	0.29	1.00
FCH, JCH	317	1.25	55.56	13.57	0.29	0.27	0.29	0.28	0.29	1.00
LBP, SC	320	0.04	40.09	13.78	0.29	0.27	0.29	0.28	0.29	1.00
FCTH, LBP	448	0.04	45.80	14.06	0.29	0.27	0.29	0.28	0.29	1.00
CEDD, LBP	400	0.05	45.45	13.96	0.28	0.26	0.29	0.28	0.28	1.00
JCD, LBP	424	0.05	49.82	13.99	0.28	0.26	0.29	0.28	0.28	1.00
RILBP, SCH	100	0.06	20.07	14.53	0.28	0.26	0.28	0.28	0.28	1.00
LBP, SCH	320	0.04	33.69	15.28	0.28	0.26	0.28	0.28	0.28	1.00
OH, RILBP	100	0.07	20.92	14.49	0.28	0.26	0.28	0.28	0.28	1.00
LBP, OH	320	0.05	32.81	15.69	0.28	0.26	0.28	0.28	0.28	1.00
FOH, OH	640	0.09	29.02	16.24	0.28	0.26	0.28	0.27	0.28	1.00
CL, LBP	289	0.04	40.03	13.67	0.28	0.26	0.28	0.27	0.28	1.00
CEDD, OH	208	0.05	22.47	13.89	0.27	0.26	0.28	0.28	0.27	1.00
FOH, JCH	768	0.38	51.33	16.19	0.28	0.26	0.28	0.27	0.28	1.00
FOH, SCH	640	0.08	28.98	16.49	0.28	0.26	0.28	0.28	0.28	1.00
CL, FOH	609	0.08	35.96	16.35	0.27	0.25	0.28	0.27	0.27	1.00
FCTH, FOH	768	0.08	33.71	15.28	0.27	0.25	0.28	0.27	0.27	1.00
CEDD, FOH	720	0.08	34.18	15.15	0.27	0.25	0.27	0.27	0.27	1.00
CEDD, RILBP	180	0.07	33.44	13.43	0.27	0.25	0.27	0.27	0.27	1.00
RILBP, SC	100	0.06	27.78	13.21	0.27	0.25	0.27	0.27	0.27	1.00
CL, OH	97	0.05	22.49	14.76	0.27	0.25	0.28	0.27	0.27	1.00

Table A.2: Average performances over different feature combinations on the **UKBench** dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
FOH, JCD	744	0.09	35.31	15.24	0.27	0.25	0.27	0.27	0.27	1.00
CL, RILBP	69	0.06	28.55	13.23	0.27	0.25	0.27	0.27	0.27	1.00
JCD, OH	232	0.06	23.85	14.92	0.27	0.25	0.27	0.27	0.27	1.00
CEDD, JCH	336	0.34	42.11	13.79	0.27	0.25	0.28	0.26	0.27	1.00
JCD, JCH	360	0.35	43.91	13.90	0.27	0.25	0.27	0.26	0.27	1.00
JCH, SC	256	0.34	41.54	13.37	0.27	0.25	0.27	0.26	0.27	1.00
FCTH, RILBP	228	0.06	33.49	13.55	0.26	0.24	0.27	0.26	0.26	1.00
FCTH, JCH	384	0.34	44.70	13.99	0.27	0.24	0.27	0.26	0.27	1.00
CL, SCH	97	0.04	22.75	14.60	0.26	0.24	0.27	0.26	0.26	1.00
FCTH, OH	256	0.05	23.01	13.98	0.26	0.24	0.26	0.26	0.26	1.00
FOH, LBP	832	0.08	45.25	16.36	0.26	0.24	0.27	0.26	0.26	1.00
FOH, RILBP	612	0.10	33.57	14.69	0.26	0.24	0.27	0.26	0.26	1.00
OH, SCH	128	0.05	19.50	14.20	0.26	0.24	0.26	0.26	0.26	1.00
JCD, RILBP	204	0.07	34.76	13.44	0.26	0.24	0.26	0.26	0.26	1.00
CEDD, SCH	208	0.05	24.09	16.28	0.26	0.24	0.26	0.26	0.26	1.00
FCTH, SCH	256	0.05	23.97	15.68	0.26	0.24	0.26	0.26	0.26	1.00
JCH, PHOG	822	0.38	81.30	17.73	0.26	0.24	0.26	0.25	0.26	1.00
JCD, SCH	232	0.06	22.29	16.03	0.26	0.24	0.26	0.26	0.26	1.00
CL, JCH	225	0.34	41.77	13.17	0.26	0.24	0.26	0.25	0.26	1.00
BPP, SCH	820	0.25	44.52	16.22	0.26	0.24	0.26	0.25	0.26	1.00
JCH, RILBP	228	0.36	40.96	13.22	0.26	0.23	0.26	0.25	0.26	1.00
JCH, LBP	448	0.34	53.44	14.40	0.26	0.23	0.26	0.24	0.26	1.00
Gabor, JCH	252	0.34	61.60	13.32	0.25	0.23	0.26	0.24	0.25	1.00
BPP, FOH	1332	0.29	58.21	20.27	0.25	0.23	0.25	0.25	0.25	1.00
BPP, OH	820	0.26	45.51	17.52	0.25	0.23	0.25	0.25	0.25	1.00
JCH, LL	256	0.35	45.65	13.35	0.25	0.23	0.25	0.24	0.25	1.00
BPP, JCH	948	0.55	74.26	18.38	0.25	0.23	0.25	0.24	0.25	1.00
EH, JCH	272	0.34	41.55	13.42	0.25	0.22	0.25	0.24	0.25	1.00
JCH, Tamura	210	0.38	44.76	14.78	0.24	0.22	0.25	0.23	0.24	1.00
BPP, SC	820	0.25	50.60	17.60	0.24	0.22	0.24	0.24	0.24	1.00
EH, OH	144	0.05	26.71	14.71	0.24	0.22	0.24	0.24	0.24	1.00
EH, SCH	144	0.05	22.57	15.58	0.24	0.22	0.24	0.24	0.24	1.00
PHOG, SCH	694	0.09	53.62	17.04	0.24	0.22	0.24	0.24	0.24	1.00
OH, PHOG	694	0.09	48.90	15.30	0.23	0.21	0.23	0.23	0.23	1.00
EH, LBP	336	0.05	44.48	13.81	0.23	0.21	0.23	0.22	0.23	1.00
EH, FOH	656	0.08	35.80	16.15	0.23	0.21	0.23	0.22	0.23	1.00
FOH, PHOG	1206	0.12	58.67	18.22	0.22	0.20	0.22	0.22	0.22	1.00
CL, Gabor	93	0.04	59.17	13.23	0.22	0.20	0.23	0.22	0.22	1.00
ACC, Tamura	274	0.21	39.56	14.57	0.21	0.20	0.21	0.22	0.21	1.00
FCH, LBP	381	0.96	55.64	15.68	0.22	0.20	0.22	0.21	0.22	1.00
FCH, RILBP	161	0.98	39.31	14.52	0.22	0.20	0.22	0.21	0.22	1.00
Gabor, SCH	124	0.04	42.27	15.21	0.22	0.20	0.22	0.21	0.22	1.00
BPP, CL	789	0.25	54.29	15.59	0.21	0.19	0.21	0.21	0.21	1.00
FCH, OH	189	0.96	33.86	14.71	0.21	0.19	0.21	0.21	0.21	1.00
FCH, SC	189	0.96	38.67	14.95	0.21	0.19	0.21	0.20	0.21	1.00
BPP, LBP	1012	0.25	65.23	17.27	0.21	0.19	0.21	0.20	0.21	1.00
PHOG, SC	694	0.09	53.02	15.33	0.20	0.19	0.20	0.21	0.20	1.00
EH, RILBP	116	0.07	35.78	13.26	0.21	0.19	0.21	0.20	0.21	1.00
FCH, SCH	189	0.96	33.30	14.10	0.21	0.19	0.21	0.20	0.21	1.00
FCH, FOH	701	0.99	45.57	15.68	0.21	0.19	0.21	0.20	0.21	1.00
Gabor, SC	124	0.04	48.36	13.20	0.21	0.19	0.21	0.20	0.21	1.00
FCH, JCD	293	0.97	41.66	15.49	0.20	0.18	0.20	0.20	0.20	1.00
Gabor, OH	124	0.05	38.23	14.65	0.20	0.18	0.21	0.20	0.20	1.00
CEDD, FCH	269	0.96	42.08	15.38	0.20	0.18	0.20	0.20	0.20	1.00
FCTH, FCH	317	0.96	43.47	15.47	0.20	0.18	0.20	0.19	0.20	1.00
FOH, Gabor	636	0.08	48.77	16.03	0.20	0.18	0.20	0.19	0.20	1.00
FCH, PHOG	755	1.00	78.25	16.03	0.19	0.18	0.20	0.19	0.19	1.00
BPP, CEDD	900	0.25	51.25	17.06	0.19	0.18	0.20	0.19	0.19	1.00
BPP, RILBP	792	0.27	52.47	15.17	0.19	0.18	0.19	0.19	0.19	1.00
BPP, FCH	881	1.17	62.54	17.07	0.19	0.17	0.19	0.19	0.19	1.00

Table A.2: Average performances over different feature combinations on the **UKBench** dataset (cont.)

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
BPP, JCD	924	0.26	52.32	17.08	0.19	0.17	0.20	0.19	0.19	1.00
LL, SCH	128	0.06	23.16	15.22	0.19	0.17	0.19	0.19	0.19	1.00
LBP, PHOG	886	0.09	64.43	16.93	0.19	0.17	0.19	0.19	0.19	1.00
PHOG, RILBP	666	0.11	58.99	15.08	0.18	0.17	0.18	0.18	0.18	1.00
BPP, PHOG	1386	0.30	77.70	19.29	0.18	0.17	0.18	0.18	0.18	1.00
BPP, FCTH	948	0.25	51.62	17.16	0.18	0.17	0.19	0.18	0.18	1.00
SCH, Tamura	82	0.09	24.05	13.02	0.18	0.16	0.18	0.18	0.18	1.00
CL, PHOG	663	0.08	52.78	15.24	0.18	0.16	0.18	0.18	0.18	1.00
Gabor, LBP	316	0.04	63.99	15.44	0.18	0.16	0.18	0.17	0.18	1.00
CL, FCH	158	0.96	42.55	14.59	0.18	0.16	0.18	0.17	0.18	1.00
FCH, Gabor	185	0.95	60.94	14.92	0.18	0.16	0.18	0.17	0.18	1.00
EH, FCH	205	0.96	39.02	15.11	0.17	0.15	0.17	0.16	0.17	1.00
FCH, Tamura	143	1.00	39.14	15.20	0.17	0.15	0.17	0.16	0.17	1.00
JCD, PHOG	798	0.10	50.76	15.68	0.16	0.15	0.17	0.17	0.16	1.00
LL, OH	128	0.06	24.64	15.22	0.17	0.15	0.17	0.16	0.17	1.00
EH, Gabor	140	0.04	40.28	13.36	0.16	0.15	0.16	0.16	0.16	1.00
CEDD, PHOG	774	0.09	49.55	15.63	0.16	0.15	0.16	0.16	0.16	1.00
BPP, EH	836	0.26	46.45	16.75	0.16	0.14	0.16	0.16	0.16	1.00
Gabor, RILBP	96	0.06	56.00	13.21	0.16	0.14	0.16	0.15	0.16	1.00
FCTH, PHOG	822	0.09	49.73	16.72	0.16	0.14	0.16	0.16	0.16	1.00
FCH, LL	189	0.97	39.50	14.86	0.16	0.14	0.16	0.15	0.16	1.00
LBP, RILBP	292	0.06	50.21	13.74	0.16	0.14	0.16	0.15	0.16	1.00
FOH, LL	640	0.09	30.46	16.48	0.16	0.14	0.16	0.15	0.16	1.00
OH, Tamura	82	0.09	25.45	14.32	0.15	0.14	0.15	0.15	0.15	1.00
CEDD, LL	208	0.06	26.53	13.66	0.15	0.13	0.16	0.14	0.15	1.00
JCD, LL	232	0.07	28.51	13.71	0.15	0.13	0.15	0.14	0.15	1.00
EH, PHOG	710	0.09	47.44	15.40	0.14	0.13	0.14	0.14	0.14	1.00
Gabor, PHOG	690	0.08	57.39	15.37	0.14	0.13	0.14	0.14	0.14	1.00
LBP, LL	320	0.06	41.61	13.97	0.14	0.12	0.15	0.13	0.14	1.00
LL, PHOG	694	0.10	46.29	15.43	0.13	0.12	0.13	0.13	0.13	1.00
LL, SC	128	0.06	27.50	15.06	0.14	0.12	0.14	0.12	0.14	1.00
CL, LL	97	0.05	28.53	13.27	0.13	0.12	0.14	0.12	0.13	1.00
BPP, Gabor	816	0.25	56.86	15.66	0.13	0.12	0.13	0.13	0.13	1.00
FCTH, LL	256	0.06	26.06	13.81	0.14	0.11	0.14	0.12	0.14	1.00
FOH, Tamura	594	0.12	41.81	14.92	0.12	0.11	0.13	0.12	0.12	1.00
BPP, LL	820	0.26	44.65	16.04	0.12	0.11	0.12	0.12	0.12	1.00
PHOG, Tamura	648	0.13	78.20	15.27	0.11	0.10	0.12	0.11	0.11	1.00
JCD, Tamura	186	0.10	41.83	14.52	0.10	0.09	0.10	0.10	0.10	1.00
FCTH, Tamura	210	0.09	36.40	14.43	0.10	0.09	0.10	0.10	0.10	1.00
CEDD, Tamura	162	0.09	36.77	14.51	0.10	0.09	0.10	0.10	0.10	1.00
LL, RILBP	100	0.08	26.20	13.28	0.11	0.09	0.11	0.09	0.11	1.00
SC, Tamura	82	0.09	35.17	14.30	0.08	0.07	0.08	0.08	0.08	1.00
EH, LL	144	0.06	28.70	13.43	0.08	0.07	0.08	0.08	0.08	1.00
BPP, Tamura	774	0.29	70.54	15.46	0.08	0.07	0.08	0.08	0.08	1.00
LBP, Tamura	274	0.08	54.61	14.62	0.08	0.07	0.08	0.07	0.08	1.00
RILBP, Tamura	54	0.10	24.22	13.70	0.08	0.07	0.08	0.07	0.08	1.00
CL, Tamura	51	0.08	28.74	13.87	0.06	0.05	0.06	0.06	0.06	1.00
Gabor, Tamura	78	0.08	65.37	14.28	0.04	0.03	0.04	0.03	0.04	1.00
LL, Tamura	82	0.10	32.18	14.32	0.04	0.03	0.04	0.03	0.04	1.00
EH, Tamura	98	0.09	46.37	14.43	0.03	0.03	0.03	0.03	0.03	1.00
Gabor, LL	124	0.05	51.34	13.40	0.03	0.02	0.03	0.02	0.03	1.00

Table A.2: Average performances over different feature combinations on the **UKBench** dataset (cont.)

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	2.17	1.19	0.49	0.29	0.26	0.30	0.27	0.29	1.00
CEDD	144	0.31	1.34	0.44	0.28	0.24	0.28	0.24	0.28	1.00
JCD	168	0.47	1.36	0.43	0.27	0.23	0.28	0.24	0.27	1.00
FCTH	192	0.31	1.34	0.42	0.26	0.22	0.26	0.22	0.26	1.00
JCH	192	5.13	2.03	0.46	0.19	0.17	0.20	0.17	0.19	1.00
CL	33	0.26	1.32	0.37	0.18	0.15	0.19	0.16	0.18	1.00
OH	64	0.43	1.04	0.47	0.17	0.14	0.17	0.14	0.17	1.00
SCH	64	0.42	1.06	0.38	0.16	0.14	0.17	0.14	0.16	1.00
FOH	576	0.91	1.49	0.67	0.15	0.12	0.15	0.13	0.15	1.00
LBP	256	0.30	2.20	0.50	0.14	0.12	0.14	0.12	0.14	1.00
SC	64	0.33	1.44	0.37	0.14	0.12	0.14	0.12	0.14	1.00
BPP	756	3.51	2.60	0.82	0.13	0.10	0.12	0.11	0.13	1.00
FCH	125	13.91	1.35	0.46	0.11	0.09	0.11	0.09	0.11	1.00
EH	80	0.40	1.33	0.39	0.11	0.09	0.11	0.09	0.11	1.00
PHOG	630	1.05	2.68	0.76	0.11	0.09	0.11	0.09	0.11	1.00
RILBP	36	0.60	1.43	0.38	0.10	0.09	0.11	0.09	0.10	1.00
Tamura	18	0.42	1.08	0.35	0.05	0.03	0.04	0.03	0.05	1.00
LL	64	0.40	1.51	0.40	0.02	0.01	0.02	0.01	0.02	1.00
Gabor	60	0.15	1.85	0.39	0.02	0.01	0.02	0.01	0.02	1.00

Table A.3: Average performances over different feature combinations on the **Holidays** dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC, CEDD	400	2.48	1.31	0.58	0.32	0.29	0.33	0.30	0.32	1.00
ACC, FCTH	448	2.49	1.36	0.61	0.32	0.29	0.33	0.29	0.32	1.00
ACC, JCD	424	2.64	1.34	0.60	0.31	0.29	0.33	0.30	0.31	1.00
ACC, CL	289	2.43	1.48	0.52	0.30	0.27	0.31	0.28	0.30	1.00
ACC, SC	320	2.50	1.38	0.53	0.30	0.27	0.31	0.28	0.30	1.00
ACC, EH	336	2.58	1.19	0.55	0.29	0.26	0.30	0.27	0.29	1.00
ACC, Gabor	316	2.32	1.75	0.53	0.29	0.26	0.30	0.27	0.29	1.00
ACC, LBP	512	2.48	1.89	0.66	0.28	0.25	0.29	0.26	0.28	1.00
CEDD, FCTH	336	0.62	1.56	0.53	0.29	0.25	0.29	0.26	0.29	1.00
ACC, RILBP	292	2.77	1.61	0.51	0.28	0.25	0.29	0.25	0.28	1.00
ACC, BPP	1012	5.68	2.14	0.99	0.27	0.24	0.29	0.25	0.27	1.00
CEDD, CL	177	0.57	1.50	0.44	0.27	0.23	0.27	0.24	0.27	1.00
ACC, FOH	832	3.08	1.78	0.83	0.26	0.23	0.27	0.24	0.26	1.00
CEDD, JCD	312	0.78	1.55	0.52	0.28	0.23	0.28	0.24	0.28	1.00
CEDD, SC	208	0.64	1.57	0.45	0.26	0.23	0.26	0.24	0.26	1.00
FCTH, JCD	360	0.78	1.60	0.54	0.27	0.23	0.27	0.24	0.27	1.00
CL, JCD	201	0.73	1.60	0.46	0.27	0.23	0.27	0.24	0.27	1.00
CEDD, Gabor	204	0.46	2.02	0.46	0.27	0.23	0.27	0.23	0.27	1.00
JCD, SC	232	0.80	1.62	0.46	0.26	0.23	0.26	0.24	0.26	1.00
Gabor, JCD	228	0.62	2.10	0.47	0.26	0.22	0.26	0.23	0.26	1.00
FCTH, SC	256	0.64	1.48	0.47	0.26	0.22	0.26	0.23	0.26	1.00
CL, FCTH	225	0.57	1.54	0.47	0.26	0.22	0.26	0.23	0.26	1.00
ACC, OH	320	2.60	1.41	0.53	0.25	0.22	0.26	0.23	0.25	1.00
EH, JCD	248	0.87	1.34	0.49	0.25	0.21	0.25	0.22	0.25	1.00
CEDD, LBP	400	0.61	2.14	0.57	0.24	0.21	0.24	0.22	0.24	1.00
CEDD, EH	224	0.71	1.22	0.47	0.24	0.21	0.25	0.22	0.24	1.00
JCD, LBP	424	0.77	2.33	0.59	0.24	0.21	0.24	0.22	0.24	1.00
FCTH, Gabor	252	0.46	1.94	0.48	0.24	0.21	0.24	0.21	0.24	1.00
JCH, SCH	256	5.55	1.99	0.47	0.23	0.20	0.23	0.21	0.23	1.00
FCTH, LBP	448	0.62	2.27	0.59	0.23	0.20	0.23	0.21	0.23	1.00
EH, FCTH	272	0.72	1.17	0.50	0.23	0.20	0.23	0.21	0.23	1.00
ACC, SCH	320	2.60	1.39	0.55	0.22	0.20	0.23	0.21	0.22	1.00
JCD, RILBP	204	1.07	1.70	0.45	0.22	0.19	0.22	0.21	0.22	1.00
LBP, SC	320	0.63	1.88	0.53	0.22	0.19	0.22	0.20	0.22	1.00
ACC, JCH	448	7.30	2.50	0.64	0.22	0.19	0.22	0.20	0.22	1.00
CL, LBP	289	0.56	2.04	0.51	0.22	0.19	0.22	0.19	0.22	1.00
LBP, OH	320	0.73	1.29	0.53	0.22	0.19	0.22	0.20	0.22	1.00
FOH, JCH	768	6.04	2.42	0.85	0.22	0.19	0.22	0.19	0.22	1.00
CL, SC	97	0.59	1.35	0.40	0.21	0.19	0.22	0.20	0.21	1.00
CEDD, RILBP	180	0.91	1.65	0.44	0.21	0.19	0.22	0.20	0.21	1.00
JCH, OH	256	5.56	1.76	0.48	0.21	0.19	0.22	0.19	0.21	1.00
CL, EH	113	0.66	1.58	0.41	0.22	0.18	0.22	0.19	0.22	1.00
OH, SC	128	0.76	1.12	0.42	0.22	0.18	0.22	0.19	0.22	1.00
JCD, JCH	360	5.60	2.29	0.56	0.21	0.18	0.21	0.19	0.21	1.00
OH, RILBP	100	1.03	0.92	0.44	0.21	0.18	0.22	0.19	0.21	1.00
FOH, SC	640	1.24	1.70	0.70	0.21	0.18	0.22	0.19	0.21	1.00
FCH, JCH	317	19.04	2.74	0.52	0.21	0.18	0.21	0.19	0.21	1.00
FCTH, JCH	384	5.44	2.27	0.57	0.21	0.18	0.21	0.19	0.21	1.00
JCD, OH	232	0.90	1.31	0.47	0.20	0.18	0.21	0.19	0.20	1.00
JCH, SC	256	5.46	2.24	0.48	0.20	0.17	0.20	0.18	0.20	1.00
RILBP, SC	100	0.92	1.38	0.40	0.20	0.17	0.21	0.18	0.20	1.00
ACC, LL	320	2.57	1.28	0.54	0.20	0.17	0.21	0.18	0.20	1.00
JCH, RILBP	228	5.73	2.11	0.46	0.20	0.17	0.20	0.18	0.20	1.00
CEDD, OH	208	0.74	1.14	0.46	0.20	0.17	0.21	0.18	0.20	1.00
CEDD, JCH	336	5.44	2.12	0.54	0.20	0.17	0.20	0.18	0.20	1.00
CL, JCH	225	5.39	1.95	0.46	0.20	0.17	0.20	0.18	0.20	1.00
FOH, OH	640	1.34	1.32	0.71	0.20	0.17	0.21	0.18	0.20	1.00
JCH, Tamura	210	5.55	2.23	0.44	0.20	0.17	0.20	0.18	0.20	1.00
FCTH, RILBP	228	0.91	1.64	0.46	0.20	0.17	0.20	0.18	0.20	1.00
JCH, LBP	448	5.43	2.53	0.64	0.20	0.17	0.20	0.17	0.20	1.00

Table A.4: Average performances over different feature combinations on the **Holidays** dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC, PHOG	886	3.22	2.74	0.94	0.20	0.17	0.20	0.18	0.20	1.00
OH, SCH	128	0.85	0.79	0.44	0.20	0.17	0.21	0.18	0.20	1.00
JCH, LL	256	5.52	2.03	0.48	0.20	0.17	0.20	0.18	0.20	1.00
FCTH, OH	256	0.74	1.47	0.48	0.19	0.17	0.20	0.18	0.19	1.00
RILBP, SCH	100	1.02	0.96	0.42	0.20	0.17	0.20	0.18	0.20	1.00
JCH, PHOG	822	6.18	4.19	0.94	0.20	0.17	0.20	0.17	0.20	1.00
BPP, JCH	948	8.64	3.23	1.00	0.19	0.17	0.19	0.17	0.19	1.00
LBP, SCH	320	0.73	1.52	0.55	0.19	0.17	0.20	0.18	0.19	1.00
EH, JCH	272	5.53	2.27	0.49	0.20	0.17	0.20	0.17	0.20	1.00
FOH, RILBP	612	1.51	1.64	0.70	0.19	0.17	0.20	0.17	0.19	1.00
CL, FOH	609	1.17	1.79	0.68	0.20	0.17	0.20	0.17	0.20	1.00
Gabor, JCH	252	5.28	2.63	0.48	0.19	0.17	0.19	0.17	0.19	1.00
SC, SCH	128	0.75	1.03	0.45	0.20	0.17	0.20	0.18	0.20	1.00
FOH, SCH	640	1.33	1.40	0.70	0.19	0.17	0.20	0.17	0.19	1.00
CL, OH	97	0.69	1.09	0.41	0.19	0.17	0.20	0.18	0.19	1.00
EH, SC	144	0.73	1.31	0.42	0.19	0.16	0.20	0.17	0.19	1.00
FOH, LBP	832	1.21	2.14	0.83	0.19	0.16	0.20	0.18	0.19	1.00
CL, RILBP	69	0.86	1.53	0.39	0.19	0.16	0.20	0.17	0.19	1.00
CEDD, FOH	720	1.22	1.54	0.75	0.19	0.16	0.20	0.17	0.19	1.00
FOH, JCD	744	1.38	1.56	0.76	0.19	0.16	0.20	0.17	0.19	1.00
FCTH, FOH	768	1.22	1.78	0.77	0.19	0.16	0.19	0.17	0.19	1.00
CL, Gabor	93	0.41	1.93	0.40	0.18	0.16	0.19	0.17	0.18	1.00
BPP, OH	820	3.94	1.84	0.85	0.19	0.16	0.19	0.16	0.19	1.00
CL, SCH	97	0.68	1.12	0.42	0.18	0.16	0.18	0.17	0.18	1.00
BPP, FOH	1332	4.42	2.46	1.15	0.18	0.16	0.19	0.16	0.18	1.00
FCTH, SCH	256	0.74	1.15	0.52	0.18	0.15	0.19	0.17	0.18	1.00
EH, OH	144	0.83	1.15	0.43	0.18	0.15	0.18	0.16	0.18	1.00
JCD, SCH	232	0.89	1.20	0.53	0.18	0.15	0.19	0.16	0.18	1.00
PHOG, SCH	694	1.47	2.71	0.78	0.18	0.15	0.18	0.16	0.18	1.00
BPP, SCH	820	3.93	1.99	0.84	0.18	0.15	0.18	0.16	0.18	1.00
BPP, SC	820	3.83	2.20	0.83	0.18	0.15	0.18	0.16	0.18	1.00
EH, LBP	336	0.71	2.11	0.54	0.18	0.15	0.18	0.15	0.18	1.00
OH, PHOG	694	1.48	2.26	0.80	0.18	0.15	0.18	0.16	0.18	1.00
CEDD, SCH	208	0.73	1.15	0.52	0.18	0.15	0.18	0.16	0.18	1.00
BPP, CEDD	900	3.82	2.30	0.91	0.18	0.15	0.18	0.15	0.18	1.00
BPP, JCD	924	3.98	2.26	0.93	0.18	0.15	0.18	0.15	0.18	1.00
BPP, CL	789	3.77	2.50	0.84	0.17	0.14	0.18	0.15	0.17	1.00
ACC, FCH	381	16.08	2.49	0.61	0.16	0.14	0.16	0.15	0.16	1.00
EH, SCH	144	0.83	1.16	0.47	0.17	0.14	0.17	0.15	0.17	1.00
ACC, Tamura	274	2.60	1.82	0.51	0.16	0.14	0.17	0.15	0.16	1.00
SCH, Tamura	82	0.85	0.84	0.38	0.17	0.14	0.17	0.15	0.17	1.00
BPP, FCTH	948	3.82	2.34	0.94	0.17	0.14	0.17	0.15	0.17	1.00
EH, FOH	656	1.31	1.43	0.71	0.16	0.14	0.17	0.14	0.16	1.00
FOH, PHOG	1206	1.96	2.92	1.11	0.16	0.14	0.16	0.14	0.16	1.00
Gabor, OH	124	0.58	1.39	0.42	0.16	0.13	0.16	0.14	0.16	1.00
BPP, LBP	1012	3.81	3.07	0.97	0.16	0.13	0.16	0.14	0.16	1.00
PHOG, SC	694	1.38	2.60	0.80	0.16	0.13	0.16	0.14	0.16	1.00
Gabor, SCH	124	0.57	1.66	0.45	0.16	0.13	0.16	0.14	0.16	1.00
Gabor, SC	124	0.48	1.99	0.41	0.16	0.13	0.16	0.14	0.16	1.00
FCH, OH	189	14.34	1.87	0.48	0.15	0.13	0.15	0.14	0.15	1.00
BPP, RILBP	792	4.10	2.75	0.82	0.16	0.13	0.16	0.14	0.16	1.00
Gabor, LBP	316	0.45	2.81	0.52	0.15	0.13	0.15	0.13	0.15	1.00
EH, RILBP	116	1.00	1.70	0.41	0.15	0.13	0.15	0.13	0.15	1.00
OH, Tamura	82	0.85	0.97	0.41	0.15	0.13	0.15	0.13	0.15	1.00
LBP, PHOG	886	1.35	3.55	0.93	0.15	0.13	0.15	0.13	0.15	1.00
FCH, RILBP	161	14.50	2.00	0.47	0.15	0.12	0.15	0.13	0.15	1.00
BPP, PHOG	1386	4.56	3.88	1.24	0.15	0.12	0.15	0.12	0.15	1.00
PHOG, RILBP	666	1.65	2.96	0.78	0.15	0.12	0.15	0.13	0.15	1.00
FCH, Tamura	143	14.33	1.98	0.42	0.14	0.12	0.14	0.13	0.14	1.00
BPP, EH	836	3.91	2.22	0.87	0.15	0.12	0.15	0.13	0.15	1.00

Table A.4: Average performances over different feature combinations on the **Holidays** dataset (cont.)

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
FCH, PHOG	755	14.96	4.03	0.87	0.14	0.12	0.14	0.13	0.14	1.00
FCH, SCH	189	14.33	1.77	0.46	0.14	0.12	0.14	0.12	0.14	1.00
FOH, Gabor	636	1.06	2.05	0.69	0.14	0.12	0.15	0.12	0.14	1.00
CL, PHOG	663	1.31	2.95	0.79	0.14	0.12	0.15	0.12	0.14	1.00
LL, SCH	128	0.82	1.11	0.45	0.14	0.11	0.14	0.12	0.14	1.00
FCH, SC	189	14.24	2.06	0.49	0.14	0.11	0.14	0.12	0.14	1.00
LL, OH	128	0.82	1.17	0.43	0.14	0.11	0.14	0.12	0.14	1.00
FCH, JCD	293	14.38	2.09	0.56	0.14	0.11	0.14	0.12	0.14	1.00
CEDD, PHOG	774	1.36	2.85	0.86	0.14	0.11	0.14	0.12	0.14	1.00
FCH, LBP	381	14.21	2.60	0.61	0.13	0.11	0.13	0.12	0.13	1.00
FCH, FOH	701	14.82	2.29	0.78	0.13	0.11	0.14	0.12	0.13	1.00
JCD, PHOG	798	1.52	2.84	0.87	0.14	0.11	0.14	0.12	0.14	1.00
FCTH, FCH	317	14.22	2.02	0.56	0.13	0.11	0.13	0.11	0.13	1.00
FCTH, PHOG	822	1.36	2.87	0.88	0.13	0.11	0.14	0.11	0.13	1.00
CEDD, FCH	269	14.22	1.70	0.54	0.13	0.11	0.13	0.11	0.13	1.00
LBP, RILBP	292	0.90	2.36	0.51	0.13	0.10	0.13	0.11	0.13	1.00
BPP, FCH	881	17.41	3.18	0.91	0.13	0.10	0.13	0.11	0.13	1.00
BPP, Gabor	816	3.66	2.65	0.86	0.12	0.10	0.12	0.11	0.12	1.00
FOH, Tamura	594	1.33	1.83	0.67	0.12	0.10	0.12	0.10	0.12	1.00
EH, Gabor	140	0.55	1.75	0.43	0.12	0.10	0.12	0.10	0.12	1.00
Gabor, RILBP	96	0.75	2.08	0.40	0.11	0.09	0.12	0.10	0.11	1.00
EH, PHOG	710	1.45	2.62	0.81	0.12	0.09	0.12	0.10	0.12	1.00
Gabor, PHOG	690	1.20	2.95	0.80	0.11	0.09	0.12	0.10	0.11	1.00
CL, FCH	158	14.17	1.75	0.47	0.11	0.09	0.12	0.10	0.11	1.00
FCH, LL	189	14.30	2.04	0.49	0.11	0.09	0.11	0.10	0.11	1.00
FCH, Gabor	185	14.06	2.24	0.49	0.11	0.09	0.11	0.10	0.11	1.00
PHOG, Tamura	648	1.47	4.01	0.75	0.11	0.09	0.11	0.09	0.11	1.00
LBP, LL	320	0.70	1.98	0.53	0.11	0.09	0.11	0.09	0.11	1.00
JCD, LL	232	0.86	1.36	0.47	0.11	0.09	0.12	0.09	0.11	1.00
CEDD, Tamura	162	0.73	1.69	0.45	0.10	0.09	0.10	0.09	0.10	1.00
JCD, Tamura	186	0.89	1.54	0.46	0.10	0.09	0.10	0.09	0.10	1.00
EH, FCH	205	14.31	1.67	0.50	0.10	0.08	0.10	0.09	0.10	1.00
CEDD, LL	208	0.71	1.23	0.46	0.11	0.08	0.11	0.09	0.11	1.00
LL, PHOG	694	1.45	2.16	0.80	0.11	0.08	0.11	0.09	0.11	1.00
FOH, LL	640	1.30	1.58	0.71	0.10	0.08	0.11	0.08	0.10	1.00
LBP, Tamura	274	0.73	2.08	0.51	0.10	0.08	0.09	0.08	0.10	1.00
LL, SC	128	0.72	1.21	0.42	0.10	0.08	0.11	0.08	0.10	1.00
FCTH, Tamura	210	0.74	1.66	0.47	0.10	0.08	0.09	0.08	0.10	1.00
SC, Tamura	82	0.75	1.47	0.41	0.10	0.08	0.10	0.08	0.10	1.00
CL, LL	97	0.65	1.41	0.40	0.10	0.07	0.10	0.08	0.10	1.00
FCTH, LL	256	0.71	1.51	0.48	0.10	0.07	0.10	0.07	0.10	1.00
BPP, LL	820	3.90	2.09	0.86	0.09	0.07	0.09	0.07	0.09	1.00
BPP, Tamura	774	3.93	2.90	0.79	0.08	0.07	0.08	0.07	0.08	1.00
RILBP, Tamura	54	1.02	1.35	0.39	0.08	0.07	0.08	0.07	0.08	1.00
LL, RILBP	100	0.99	1.47	0.41	0.08	0.06	0.08	0.07	0.08	1.00
CL, Tamura	51	0.68	1.50	0.39	0.08	0.06	0.08	0.06	0.08	1.00
EH, LL	144	0.80	1.17	0.43	0.07	0.05	0.07	0.05	0.07	1.00
LL, Tamura	82	0.82	1.67	0.41	0.06	0.04	0.05	0.05	0.06	1.00
Gabor, Tamura	78	0.57	1.89	0.41	0.05	0.04	0.05	0.04	0.05	1.00
EH, Tamura	98	0.83	1.63	0.42	0.05	0.04	0.05	0.04	0.05	1.00
Gabor, LL	124	0.54	1.90	0.42	0.04	0.03	0.04	0.03	0.04	1.00

Table A.4: Average performances over different feature combinations on the **Holidays** dataset (cont.)

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
EH	80	0.01	79.37	1.51	0.13	0.12	0.13	0.14	0.13	0.99
PHOG	630	0.04	107.89	4.58	0.12	0.11	0.12	0.13	0.12	0.99
JCD	168	0.02	37.74	1.87	0.11	0.09	0.11	0.09	0.11	0.99
CEDD	144	0.01	37.37	1.78	0.11	0.08	0.10	0.09	0.11	0.99
CL	33	0.01	69.07	1.15	0.10	0.08	0.10	0.10	0.10	0.99
BPP	756	0.11	111.59	5.21	0.08	0.08	0.08	0.10	0.08	0.99
FCTH	192	0.01	40.71	1.91	0.09	0.07	0.09	0.08	0.09	0.99
JCH	192	0.16	106.72	2.01	0.08	0.07	0.08	0.09	0.08	0.99
ACC	256	0.07	82.89	2.42	0.07	0.06	0.07	0.08	0.07	0.99
LBP	256	0.01	133.58	1.95	0.07	0.06	0.07	0.07	0.07	0.99
Tamura	18	0.05	547.86	1.09	0.07	0.05	0.06	0.06	0.07	0.99
RILBP	36	0.02	303.96	1.48	0.06	0.04	0.06	0.05	0.06	0.99
SCH	64	0.01	89.23	1.42	0.06	0.04	0.05	0.05	0.06	0.99
OH	64	0.01	79.73	2.17	0.05	0.04	0.05	0.04	0.05	0.99
FCH	125	0.49	150.76	1.67	0.05	0.04	0.05	0.05	0.05	0.99
FOH	576	0.03	114.59	3.71	0.05	0.03	0.05	0.04	0.05	0.99
LL	64	0.02	760.97	1.55	0.04	0.03	0.03	0.03	0.04	0.99
SC	64	0.01	164.54	1.24	0.04	0.03	0.03	0.03	0.04	0.99
Gabor	60	0.01	133.01	0.73	0.03	0.01	0.02	0.01	0.03	0.99

Table A.5: Average performances over different feature combinations on the **Caltech-256** dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
EH, JCD	248	0.03	49.21	2.39	0.16	0.15	0.16	0.18	0.16	1.00
CEDD, EH	224	0.03	47.09	2.27	0.16	0.15	0.16	0.18	0.16	0.99
EH, FCTH	272	0.03	54.68	2.51	0.16	0.14	0.16	0.17	0.16	1.00
CL, EH	113	0.02	60.10	1.60	0.15	0.13	0.15	0.17	0.15	0.99
CL, PHOG	663	0.05	94.14	4.71	0.14	0.13	0.14	0.16	0.14	0.99
ACC, PHOG	886	0.11	113.19	5.95	0.14	0.13	0.14	0.16	0.14	0.99
EH, PHOG	710	0.05	98.31	5.00	0.13	0.12	0.13	0.15	0.13	0.99
CEDD, PHOG	774	0.05	106.96	5.34	0.13	0.12	0.13	0.15	0.13	0.99
PHOG, SC	694	0.05	100.31	4.88	0.13	0.12	0.13	0.14	0.13	0.99
JCD, PHOG	798	0.05	109.46	5.47	0.13	0.12	0.13	0.15	0.13	0.99
PHOG, RILBP	666	0.06	108.08	4.11	0.13	0.12	0.13	0.14	0.13	0.99
Gabor, PHOG	690	0.05	102.58	4.88	0.13	0.12	0.12	0.14	0.13	0.99
BPP, PHOG	1386	0.15	168.98	8.63	0.12	0.12	0.12	0.15	0.12	0.99
FCTH, PHOG	822	0.05	108.32	5.60	0.13	0.12	0.12	0.14	0.13	0.99
PHOG, SCH	694	0.05	110.02	4.65	0.12	0.11	0.12	0.14	0.12	0.99
JCH, PHOG	822	0.20	151.81	5.51	0.12	0.11	0.12	0.14	0.12	0.99
FOH, PHOG	1206	0.07	147.38	7.66	0.12	0.11	0.12	0.14	0.12	0.99
OH, PHOG	694	0.05	105.20	4.79	0.12	0.11	0.12	0.14	0.12	0.99
LBP, PHOG	886	0.05	127.18	5.57	0.12	0.11	0.12	0.14	0.12	0.99
LL, PHOG	694	0.06	112.20	4.82	0.12	0.11	0.12	0.13	0.12	0.99
EH, Gabor	140	0.03	127.92	1.65	0.13	0.11	0.13	0.13	0.13	0.99
EH, LBP	336	0.03	115.07	2.40	0.12	0.10	0.12	0.13	0.12	0.99
FCH, PHOG	755	0.52	154.72	4.60	0.11	0.10	0.11	0.13	0.11	0.99
EH, SC	144	0.03	102.05	1.60	0.11	0.10	0.11	0.13	0.11	0.99
ACC, BPP	1012	0.18	71.55	4.84	0.11	0.10	0.11	0.13	0.11	0.99
PHOG, Tamura	648	0.09	205.96	3.80	0.11	0.10	0.11	0.12	0.11	0.99
CL, JCD	201	0.03	65.79	1.90	0.12	0.10	0.11	0.14	0.12	0.99
ACC, EH	336	0.09	64.00	2.91	0.12	0.10	0.11	0.13	0.12	0.99
CEDD, CL	177	0.02	66.80	1.80	0.12	0.10	0.11	0.13	0.12	0.99
JCD, LBP	424	0.03	124.41	2.90	0.11	0.10	0.11	0.12	0.11	1.00
BPP, JCD	924	0.13	116.38	6.10	0.11	0.10	0.10	0.13	0.11	0.99
EH, RILBP	116	0.04	80.78	1.42	0.11	0.10	0.11	0.13	0.11	0.99
CL, FCTH	225	0.02	72.01	2.01	0.11	0.10	0.11	0.12	0.11	0.99
BPP, CEDD	900	0.13	113.40	5.98	0.10	0.10	0.10	0.13	0.10	0.99
CEDD, LBP	400	0.02	121.54	2.57	0.11	0.09	0.11	0.11	0.11	0.99
BPP, CL	789	0.12	111.80	5.36	0.10	0.09	0.10	0.13	0.10	0.99
BPP, FCTH	948	0.13	117.37	6.23	0.10	0.09	0.10	0.12	0.10	0.99
BPP, EH	836	0.13	101.57	5.93	0.10	0.09	0.10	0.11	0.10	0.99
FCTH, LBP	448	0.02	130.94	2.74	0.11	0.09	0.11	0.11	0.11	0.99
CEDD, FCTH	336	0.03	45.77	2.81	0.11	0.09	0.11	0.10	0.11	0.99
CL, LBP	289	0.02	134.45	2.20	0.10	0.09	0.10	0.12	0.10	0.99
EH, JCH	272	0.18	106.82	2.43	0.10	0.09	0.10	0.11	0.10	0.99
FCTH, JCD	360	0.03	49.55	2.81	0.11	0.09	0.11	0.10	0.11	0.99
BPP, LBP	1012	0.12	150.28	5.74	0.10	0.09	0.09	0.11	0.10	0.99
CL, JCH	225	0.17	113.70	2.18	0.10	0.09	0.10	0.11	0.10	0.99
EH, FOH	656	0.05	124.97	4.20	0.10	0.09	0.10	0.11	0.10	0.99
CEDD, JCD	312	0.03	33.78	2.91	0.11	0.09	0.10	0.10	0.11	0.99
BPP, JCH	948	0.28	172.21	6.02	0.10	0.08	0.09	0.11	0.10	0.99
EH, LL	144	0.04	109.38	1.71	0.09	0.08	0.09	0.11	0.09	0.99
CL, RILBP	69	0.03	94.83	1.45	0.10	0.08	0.10	0.10	0.10	0.99
ACC, LBP	512	0.08	115.11	3.33	0.10	0.08	0.09	0.11	0.10	0.99
ACC, JCH	448	0.23	116.83	3.48	0.10	0.08	0.10	0.10	0.10	0.99
CEDD, JCH	336	0.17	118.41	2.83	0.10	0.08	0.09	0.10	0.10	0.99
ACC, CL	289	0.08	79.37	2.63	0.10	0.08	0.09	0.11	0.10	0.99
BPP, SCH	820	0.12	138.74	5.16	0.09	0.08	0.09	0.10	0.09	0.99
ACC, JCD	424	0.09	76.99	3.40	0.10	0.08	0.09	0.10	0.10	0.99
EH, SCH	144	0.03	51.52	2.20	0.09	0.08	0.09	0.10	0.09	0.99
CEDD, RILBP	180	0.03	86.70	1.62	0.10	0.08	0.09	0.10	0.10	0.99
CL, Tamura	51	0.06	127.99	1.30	0.10	0.08	0.10	0.09	0.10	0.99
Gabor, JCD	228	0.03	100.55	1.89	0.10	0.08	0.09	0.10	0.10	0.99

Table A.6: Average performances over different feature combinations on the **Caltech-256** dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
BPP, FOH	1332	0.14	172.09	7.85	0.09	0.08	0.08	0.10	0.09	0.99
ACC, CEDD	400	0.08	74.05	3.25	0.09	0.08	0.09	0.10	0.09	0.99
EH, OH	144	0.03	62.17	1.73	0.09	0.08	0.09	0.11	0.09	0.99
JCH, SCH	256	0.17	92.67	2.33	0.10	0.08	0.09	0.10	0.10	0.99
BPP, RILBP	792	0.14	133.67	4.29	0.09	0.08	0.09	0.10	0.09	0.99
BPP, Gabor	816	0.13	128.06	5.51	0.09	0.08	0.08	0.10	0.09	0.99
BPP, Tamura	774	0.16	256.71	4.11	0.09	0.08	0.09	0.10	0.09	0.99
BPP, SC	820	0.12	122.80	5.25	0.09	0.08	0.08	0.10	0.09	0.99
JCH, Tamura	210	0.21	120.81	1.99	0.09	0.08	0.09	0.10	0.09	0.99
BPP, OH	820	0.13	123.80	5.21	0.08	0.08	0.08	0.10	0.08	0.99
JCH, OH	256	0.18	97.63	2.35	0.09	0.08	0.09	0.10	0.09	0.99
JCD, JCH	360	0.18	121.63	2.97	0.09	0.08	0.09	0.10	0.09	0.99
CEDD, Gabor	204	0.03	99.06	1.83	0.09	0.08	0.09	0.09	0.09	0.99
JCD, RILBP	204	0.04	97.09	1.72	0.09	0.08	0.09	0.09	0.09	0.99
ACC, Tamura	274	0.12	150.46	2.29	0.09	0.08	0.08	0.09	0.09	0.99
FCTH, RILBP	228	0.03	89.96	1.77	0.09	0.07	0.09	0.09	0.09	0.99
ACC, FCTH	448	0.08	82.44	3.50	0.09	0.07	0.08	0.09	0.09	0.99
JCH, RILBP	228	0.18	120.66	2.17	0.09	0.07	0.09	0.09	0.09	0.99
CL, Gabor	93	0.02	153.53	1.41	0.09	0.07	0.09	0.09	0.09	0.99
FCTH, JCH	384	0.17	119.51	3.07	0.09	0.07	0.09	0.09	0.09	0.99
JCH, LL	256	0.18	97.03	2.61	0.09	0.07	0.08	0.09	0.09	0.99
EH, Tamura	98	0.06	151.35	1.82	0.09	0.07	0.08	0.09	0.09	0.99
JCH, LBP	448	0.17	153.62	3.46	0.09	0.07	0.08	0.09	0.09	0.99
BPP, LL	820	0.14	126.93	5.28	0.08	0.07	0.08	0.10	0.08	0.99
LBP, SCH	320	0.02	132.40	3.30	0.09	0.07	0.08	0.09	0.09	0.99
JCH, SC	256	0.17	132.47	2.34	0.09	0.07	0.08	0.09	0.09	0.99
SCH, Tamura	82	0.06	58.98	1.64	0.09	0.07	0.08	0.09	0.09	0.99
ACC, RILBP	292	0.09	96.77	2.20	0.09	0.07	0.09	0.09	0.09	0.99
FOH, JCH	768	0.19	135.73	5.45	0.09	0.07	0.09	0.09	0.09	0.99
CL, FOH	609	0.04	132.89	3.94	0.09	0.07	0.08	0.09	0.09	0.99
JCD, Tamura	186	0.07	205.96	1.93	0.09	0.07	0.08	0.09	0.09	0.99
FOH, LBP	832	0.04	180.69	5.33	0.08	0.07	0.08	0.09	0.08	0.99
Gabor, JCH	252	0.17	121.06	2.58	0.09	0.07	0.08	0.09	0.09	0.99
CL, SC	97	0.02	128.05	1.42	0.08	0.07	0.08	0.09	0.08	0.99
FCTH, Gabor	252	0.03	114.89	1.96	0.09	0.07	0.09	0.08	0.09	0.99
JCD, SC	232	0.03	107.97	2.07	0.08	0.07	0.08	0.09	0.08	0.99
FCTH, Tamura	210	0.06	203.14	2.02	0.08	0.07	0.08	0.08	0.08	0.99
CEDD, Tamura	162	0.06	187.36	1.84	0.08	0.07	0.08	0.08	0.08	0.99
FCH, JCH	317	0.65	113.09	2.95	0.08	0.07	0.08	0.09	0.08	0.99
EH, FCH	205	0.50	113.08	2.13	0.08	0.07	0.08	0.09	0.08	0.99
CL, LL	97	0.03	138.08	1.52	0.08	0.07	0.08	0.09	0.08	0.99
FOH, JCD	744	0.05	133.97	4.52	0.08	0.07	0.08	0.08	0.08	0.99
BPP, FCH	881	0.60	218.32	4.67	0.08	0.07	0.08	0.09	0.08	0.99
LBP, RILBP	292	0.03	159.80	2.14	0.08	0.07	0.08	0.07	0.08	0.99
CEDD, SC	208	0.02	103.29	1.77	0.08	0.07	0.08	0.08	0.08	0.99
CL, OH	97	0.02	84.15	1.68	0.08	0.07	0.07	0.08	0.08	0.99
LBP, Tamura	274	0.06	231.04	2.29	0.08	0.06	0.08	0.08	0.08	0.99
Gabor, LBP	316	0.02	183.82	2.23	0.08	0.06	0.08	0.07	0.08	0.99
ACC, Gabor	316	0.08	103.09	2.73	0.08	0.06	0.07	0.09	0.08	0.99
RILBP, SCH	100	0.03	60.10	1.96	0.08	0.06	0.08	0.08	0.08	0.99
LBP, OH	320	0.02	133.17	2.52	0.08	0.06	0.07	0.08	0.08	0.99
RILBP, Tamura	54	0.07	133.20	1.31	0.08	0.06	0.08	0.07	0.08	0.99
FCH, Tamura	143	0.54	126.56	1.67	0.08	0.06	0.08	0.08	0.08	0.99
CL, SCH	97	0.02	81.81	1.69	0.08	0.06	0.07	0.08	0.08	0.99
SC, Tamura	82	0.06	172.04	1.48	0.08	0.06	0.08	0.07	0.08	0.99
CEDD, FOH	720	0.04	119.21	4.42	0.08	0.06	0.07	0.08	0.08	0.99
ACC, FOH	832	0.10	138.40	5.26	0.08	0.06	0.07	0.08	0.08	0.99
FCTH, FOH	768	0.04	86.68	4.84	0.08	0.06	0.07	0.07	0.08	0.99
JCD, LL	232	0.04	90.21	2.07	0.07	0.06	0.07	0.08	0.07	0.99
ACC, LL	320	0.09	98.13	2.67	0.07	0.06	0.07	0.08	0.07	0.99

Table A.6: Average performances over different feature combinations on the **Caltech-256** dataset (cont.)

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
LBP, LL	320	0.03	164.34	2.71	0.07	0.06	0.07	0.07	0.07	0.99
FOH, Tamura	594	0.08	120.44	3.51	0.08	0.06	0.08	0.07	0.08	0.99
OH, Tamura	82	0.06	67.48	1.46	0.08	0.06	0.07	0.08	0.08	0.99
JCD, SCH	232	0.03	69.76	2.84	0.07	0.06	0.07	0.08	0.07	0.99
CEDD, LL	208	0.03	88.53	1.98	0.07	0.06	0.07	0.08	0.07	0.99
LBP, SC	320	0.02	165.47	2.35	0.07	0.06	0.07	0.07	0.07	0.99
ACC, SC	320	0.08	103.32	2.60	0.07	0.06	0.07	0.08	0.07	0.99
JCD, OH	232	0.03	73.83	2.11	0.07	0.06	0.07	0.07	0.07	0.99
FOH, RILBP	612	0.05	109.30	4.13	0.08	0.06	0.07	0.07	0.08	0.99
ACC, SCH	320	0.08	84.37	2.95	0.07	0.06	0.07	0.08	0.07	0.99
CEDD, SCH	208	0.02	63.33	2.84	0.07	0.06	0.07	0.08	0.07	0.99
ACC, OH	320	0.08	85.16	2.66	0.07	0.06	0.07	0.07	0.07	0.99
CEDD, OH	208	0.03	66.92	2.28	0.07	0.06	0.07	0.08	0.07	0.99
FCTH, LL	256	0.03	102.12	2.18	0.07	0.06	0.06	0.07	0.07	0.99
LL, Tamura	82	0.07	120.82	1.48	0.07	0.06	0.07	0.07	0.07	0.99
FCTH, SC	256	0.02	119.60	1.92	0.07	0.06	0.07	0.07	0.07	0.99
FCTH, OH	256	0.03	68.26	2.23	0.07	0.06	0.06	0.07	0.07	0.99
FCTH, SCH	256	0.02	64.16	2.82	0.07	0.05	0.06	0.07	0.07	0.99
Gabor, RILBP	96	0.03	242.71	0.95	0.07	0.05	0.07	0.06	0.07	0.99
FCH, LBP	381	0.50	210.87	3.05	0.07	0.05	0.07	0.07	0.07	0.99
OH, RILBP	100	0.04	65.85	2.31	0.07	0.05	0.06	0.06	0.07	0.99
CL, FCH	158	0.50	187.92	1.86	0.07	0.05	0.07	0.07	0.07	0.99
Gabor, Tamura	78	0.06	381.45	1.71	0.07	0.05	0.06	0.06	0.07	0.99
LL, SCH	128	0.03	56.94	1.96	0.06	0.05	0.06	0.07	0.06	0.99
FOH, SCH	640	0.04	82.97	4.15	0.06	0.05	0.06	0.06	0.06	0.99
RILBP, SC	100	0.03	135.87	1.44	0.07	0.05	0.06	0.06	0.07	0.99
ACC, FCH	381	0.56	134.08	2.98	0.06	0.05	0.06	0.07	0.06	0.99
FCH, JCD	293	0.50	168.64	2.60	0.07	0.05	0.07	0.06	0.07	0.99
LL, RILBP	100	0.04	110.96	1.72	0.06	0.05	0.05	0.06	0.06	0.99
FCH, FOH	701	0.52	84.54	4.33	0.07	0.05	0.06	0.06	0.07	0.99
CEDD, FCH	269	0.50	161.12	2.46	0.06	0.05	0.06	0.07	0.06	0.99
FCH, RILBP	161	0.51	155.88	1.87	0.06	0.05	0.06	0.06	0.06	0.99
SC, SCH	128	0.02	117.27	2.00	0.06	0.05	0.06	0.06	0.06	0.99
Gabor, SCH	124	0.02	107.54	1.95	0.06	0.05	0.06	0.06	0.06	0.99
FCH, SCH	189	0.50	99.42	2.34	0.06	0.05	0.06	0.06	0.06	0.99
FCTH, FCH	317	0.50	167.72	2.70	0.06	0.05	0.06	0.06	0.06	0.99
FCH, LL	189	0.51	112.08	2.31	0.06	0.05	0.06	0.07	0.06	0.99
OH, SCH	128	0.03	43.11	1.88	0.06	0.05	0.05	0.06	0.06	0.99
FOH, SC	640	0.04	169.54	4.02	0.06	0.04	0.05	0.05	0.06	0.99
FOH, OH	640	0.04	88.26	4.23	0.06	0.04	0.05	0.05	0.06	0.99
LL, OH	128	0.04	70.51	2.11	0.05	0.04	0.05	0.06	0.05	0.99
OH, SC	128	0.03	121.21	1.65	0.06	0.04	0.05	0.05	0.06	0.99
Gabor, OH	124	0.03	115.97	1.63	0.06	0.04	0.05	0.05	0.06	0.99
FCH, Gabor	185	0.50	213.39	2.01	0.06	0.04	0.06	0.05	0.06	0.99
FOH, LL	640	0.05	145.35	4.22	0.05	0.04	0.05	0.05	0.05	0.99
FCH, OH	189	0.50	107.30	2.01	0.06	0.04	0.05	0.05	0.06	0.99
FOH, Gabor	636	0.04	187.20	3.95	0.06	0.04	0.05	0.05	0.06	0.99
FCH, SC	189	0.50	216.88	2.04	0.05	0.04	0.05	0.06	0.05	0.99
LL, SC	128	0.03	174.87	1.65	0.05	0.04	0.05	0.05	0.05	0.99
Gabor, SC	124	0.02	235.59	1.45	0.05	0.04	0.05	0.04	0.05	0.99
Gabor, LL	124	0.04	207.15	1.35	0.05	0.03	0.04	0.04	0.05	0.99

Table A.6: Average performances over different feature combinations on the **Caltech-256** dataset (cont.)

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC	256	0.36	1.31	0.28	0.65	0.63	0.60	0.66	0.65	0.95
CEDD	144	0.06	0.73	0.13	0.64	0.62	0.60	0.65	0.64	0.95
JCD	168	0.09	0.70	0.15	0.64	0.62	0.59	0.66	0.64	0.95
FCTH	192	0.06	0.61	0.16	0.63	0.61	0.59	0.64	0.63	0.95
CL	33	0.05	3.37	0.05	0.61	0.58	0.56	0.64	0.61	0.94
SCH	64	0.06	1846.94	0.07	0.56	0.53	0.51	0.62	0.56	0.94
FOH	576	0.16	2.43	0.43	0.54	0.51	0.49	0.57	0.54	0.93
FCH	125	2.47	392.43	0.10	0.54	0.51	0.49	0.61	0.54	0.93
SC	64	0.07	1.55	0.06	0.52	0.49	0.46	0.54	0.52	0.93
OH	64	0.08	50.83	0.07	0.50	0.47	0.44	0.54	0.50	0.93
JCH	192	0.87	2.89	0.17	0.48	0.46	0.42	0.53	0.48	0.93
EH	80	0.08	1.53	0.13	0.47	0.45	0.40	0.48	0.47	0.92
BPP	756	0.52	4.29	1.11	0.44	0.43	0.37	0.45	0.44	0.92
PHOG	630	0.16	5.38	0.98	0.40	0.38	0.32	0.44	0.40	0.91
LBP	256	0.07	1.55	0.24	0.37	0.33	0.29	0.38	0.37	0.91
RILBP	36	0.10	78.84	0.05	0.34	0.31	0.26	0.35	0.34	0.91
LL	64	0.08	2.29	0.09	0.32	0.29	0.23	0.33	0.32	0.90
Gabor	60	0.04	1.65	0.06	0.30	0.26	0.21	0.28	0.30	0.90
Tamura	18	0.11	3928.23	0.03	0.29	0.24	0.20	0.31	0.29	0.90

Table A.7: Average performances over different feature combinations on the **Kvasir** dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
ACC, EH	336	0.44	2.06	0.42	0.67	0.66	0.63	0.68	0.67	0.95
ACC, FCTH	448	0.42	1.62	0.50	0.67	0.66	0.63	0.69	0.67	0.95
ACC, CL	289	0.41	1.73	0.32	0.67	0.66	0.63	0.69	0.67	0.95
ACC, CEDD	400	0.42	1.51	0.45	0.66	0.64	0.62	0.68	0.66	0.95
CL, EH	113	0.13	1.43	0.16	0.66	0.64	0.61	0.67	0.66	0.95
CL, JCD	201	0.14	0.83	0.18	0.66	0.64	0.61	0.69	0.66	0.95
CEDD, FCTH	336	0.11	0.99	0.32	0.66	0.64	0.62	0.66	0.66	0.95
ACC, LBP	512	0.43	2.02	0.54	0.65	0.64	0.61	0.66	0.65	0.95
EH, JCD	248	0.17	1.52	0.36	0.65	0.63	0.61	0.67	0.65	0.95
ACC, SC	320	0.43	1.82	0.31	0.65	0.63	0.61	0.65	0.65	0.95
CEDD, JCD	312	0.15	1.07	0.30	0.65	0.63	0.61	0.65	0.65	0.95
ACC, BPP	1012	0.88	4.62	1.40	0.64	0.63	0.60	0.65	0.64	0.95
FCTH, JCD	360	0.15	1.17	0.34	0.65	0.63	0.61	0.67	0.65	0.95
ACC, JCD	424	0.45	1.59	0.47	0.65	0.63	0.61	0.68	0.65	0.95
EH, FCTH	272	0.13	1.15	0.39	0.65	0.63	0.61	0.67	0.65	0.95
CEDD, CL	177	0.11	0.88	0.17	0.65	0.63	0.61	0.66	0.65	0.95
ACC, RILBP	292	0.46	2.19	0.29	0.64	0.63	0.60	0.66	0.64	0.95
ACC, Gabor	316	0.40	1.87	0.32	0.65	0.63	0.60	0.66	0.65	0.95
CL, FCTH	225	0.11	0.79	0.20	0.65	0.63	0.61	0.67	0.65	0.95
CL, Gabor	93	0.10	1.76	0.11	0.63	0.62	0.59	0.65	0.63	0.95
CEDD, EH	224	0.14	1.54	0.32	0.64	0.62	0.60	0.66	0.64	0.95
CL, SC	97	0.12	2.02	0.11	0.63	0.61	0.58	0.66	0.63	0.95
ACC, FOH	832	0.52	3.58	0.67	0.63	0.61	0.58	0.65	0.63	0.95
ACC, OH	320	0.44	2.29	0.31	0.64	0.61	0.60	0.65	0.64	0.95
JCD, SC	232	0.16	0.98	0.18	0.64	0.61	0.59	0.66	0.64	0.95
ACC, LL	320	0.44	1.91	0.38	0.63	0.61	0.59	0.65	0.63	0.95
FCTH, SC	256	0.12	0.92	0.20	0.63	0.61	0.58	0.65	0.63	0.95
JCD, LL	232	0.17	1.38	0.27	0.62	0.60	0.57	0.63	0.62	0.95
BPP, JCD	924	0.61	3.75	1.32	0.61	0.60	0.56	0.63	0.61	0.94
CEDD, OH	208	0.14	3.25	0.19	0.62	0.60	0.57	0.64	0.62	0.95
CEDD, SC	208	0.13	0.90	0.17	0.62	0.60	0.58	0.64	0.62	0.95
BPP, CEDD	900	0.58	4.07	1.28	0.61	0.60	0.56	0.62	0.61	0.94
ACC, SCH	320	0.42	3.31	0.30	0.61	0.60	0.57	0.66	0.61	0.94
CEDD, Gabor	204	0.10	1.06	0.17	0.62	0.59	0.58	0.62	0.62	0.95
ACC, PHOG	886	0.52	5.20	1.33	0.61	0.59	0.56	0.65	0.61	0.94
FCTH, LL	256	0.14	1.22	0.30	0.61	0.59	0.56	0.63	0.61	0.94
FCTH, LBP	448	0.13	1.30	0.39	0.62	0.59	0.57	0.64	0.62	0.95
JCD, OH	232	0.17	2.78	0.21	0.62	0.59	0.57	0.65	0.62	0.95
Gabor, JCD	228	0.13	1.05	0.18	0.62	0.59	0.57	0.65	0.62	0.95
CEDD, LL	208	0.14	1.15	0.25	0.61	0.59	0.56	0.63	0.61	0.94
FCTH, RILBP	228	0.16	1.56	0.19	0.61	0.59	0.57	0.64	0.61	0.94
ACC, FCH	381	2.83	8.11	0.33	0.60	0.59	0.56	0.66	0.60	0.94
JCD, LBP	424	0.16	1.30	0.37	0.62	0.59	0.57	0.62	0.62	0.95
FOH, JCD	744	0.25	2.54	0.54	0.61	0.59	0.56	0.64	0.61	0.94
CL, LBP	289	0.12	1.57	0.27	0.61	0.59	0.57	0.62	0.61	0.94
CEDD, LBP	400	0.13	1.27	0.35	0.62	0.59	0.57	0.63	0.62	0.95
JCD, RILBP	204	0.19	1.78	0.17	0.61	0.59	0.56	0.62	0.61	0.94
FCTH, FOH	768	0.22	2.34	0.56	0.61	0.58	0.56	0.64	0.61	0.94
CL, RILBP	69	0.15	8.25	0.09	0.60	0.58	0.56	0.64	0.60	0.94
CEDD, FOH	720	0.22	2.53	0.53	0.61	0.58	0.57	0.63	0.61	0.94
CL, FOH	609	0.21	2.73	0.46	0.61	0.58	0.56	0.63	0.61	0.94
BPP, CL	789	0.58	4.06	1.13	0.60	0.58	0.55	0.60	0.60	0.94
BPP, FCTH	948	0.58	4.66	1.36	0.60	0.58	0.55	0.60	0.60	0.94
BPP, FOH	1332	0.69	6.08	1.27	0.59	0.58	0.54	0.61	0.59	0.94
FCTH, Gabor	252	0.10	1.01	0.20	0.61	0.58	0.56	0.63	0.61	0.94
CEDD, RILBP	180	0.16	2.23	0.16	0.60	0.58	0.55	0.62	0.60	0.94
CL, OH	97	0.13	6.76	0.11	0.61	0.57	0.56	0.62	0.61	0.94
EH, SC	144	0.14	1.61	0.17	0.59	0.57	0.53	0.60	0.59	0.94
JCD, SCH	232	0.15	25.36	0.22	0.59	0.57	0.55	0.63	0.59	0.94
FCTH, OH	256	0.14	2.55	0.22	0.60	0.57	0.56	0.64	0.60	0.94

Table A.8: Average performances over different feature combinations on the **Kvasir** dataset

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
FCTH, SCH	256	0.12	33.35	0.24	0.59	0.57	0.54	0.62	0.59	0.94
CL, LL	97	0.14	1.72	0.14	0.58	0.56	0.53	0.61	0.58	0.94
FOH, SC	640	0.23	2.62	0.50	0.59	0.56	0.54	0.61	0.59	0.94
BPP, SCH	820	0.59	4.19	0.78	0.58	0.56	0.53	0.62	0.58	0.94
EH, SCH	144	0.14	4.59	0.15	0.58	0.56	0.54	0.63	0.58	0.94
EH, FOH	656	0.24	2.96	0.53	0.58	0.56	0.53	0.61	0.58	0.94
FOH, PHOG	1206	0.32	7.31	1.46	0.57	0.56	0.51	0.59	0.57	0.94
FCH, PHOG	755	2.63	5.71	0.71	0.57	0.56	0.52	0.63	0.57	0.94
JCH, SCH	256	0.93	2.40	0.28	0.57	0.56	0.52	0.63	0.57	0.94
FOH, SCH	640	0.23	13.36	0.58	0.58	0.55	0.53	0.63	0.58	0.94
JCD, PHOG	798	0.25	5.36	1.23	0.57	0.55	0.52	0.60	0.57	0.94
FOH, OH	640	0.24	4.35	0.54	0.58	0.55	0.54	0.59	0.58	0.94
FCH, JCD	293	2.56	94.36	0.26	0.57	0.55	0.52	0.64	0.57	0.94
LBP, SCH	320	0.13	11.11	0.30	0.57	0.55	0.53	0.63	0.57	0.94
FCH, FOH	701	2.63	59.25	0.59	0.56	0.55	0.52	0.62	0.56	0.94
ACC, JCH	448	1.23	3.30	0.41	0.57	0.54	0.53	0.61	0.57	0.94
CL, PHOG	663	0.21	4.52	1.02	0.56	0.54	0.51	0.60	0.56	0.94
FOH, LL	640	0.25	3.42	0.55	0.57	0.54	0.51	0.58	0.57	0.94
BPP, OH	820	0.60	4.76	0.89	0.56	0.54	0.50	0.56	0.56	0.94
FCTH, FCH	317	2.53	183.81	0.28	0.57	0.54	0.52	0.63	0.57	0.94
BPP, FCH	881	2.99	6.55	0.76	0.56	0.54	0.51	0.61	0.56	0.94
CEDD, PHOG	774	0.22	4.91	1.19	0.56	0.54	0.51	0.59	0.56	0.94
FCTH, PHOG	822	0.22	5.38	1.26	0.55	0.54	0.50	0.59	0.55	0.94
CEDD, SCH	208	0.12	32.21	0.21	0.57	0.54	0.52	0.60	0.57	0.94
BPP, SC	820	0.59	4.07	1.05	0.55	0.54	0.49	0.56	0.55	0.94
PHOG, SCH	694	0.22	4.53	0.75	0.56	0.53	0.51	0.62	0.56	0.94
FCH, SC	189	2.54	89.63	0.18	0.56	0.53	0.51	0.62	0.56	0.94
CL, SCH	97	0.12	37.18	0.11	0.57	0.53	0.53	0.62	0.57	0.94
FCH, JCH	317	3.34	3.27	0.32	0.56	0.53	0.50	0.63	0.56	0.94
EH, FCH	205	2.55	22.64	0.19	0.55	0.53	0.51	0.62	0.55	0.94
SC, SCH	128	0.13	77.60	0.14	0.56	0.53	0.52	0.60	0.56	0.94
RILBP, SCH	100	0.17	214.58	0.12	0.56	0.53	0.51	0.63	0.56	0.94
CL, JCH	225	0.92	2.57	0.21	0.55	0.53	0.50	0.60	0.55	0.94
FCH, SCH	189	2.53	44.59	0.20	0.56	0.53	0.51	0.64	0.56	0.94
Gabor, SCH	124	0.11	350.00	0.14	0.55	0.53	0.51	0.61	0.55	0.94
JCD, JCH	360	0.96	3.32	0.34	0.55	0.53	0.51	0.61	0.55	0.94
FCH, Gabor	185	2.51	471.44	0.17	0.55	0.52	0.50	0.59	0.55	0.94
FOH, LBP	832	0.23	2.78	0.64	0.56	0.52	0.50	0.59	0.56	0.94
ACC, Tamura	274	0.47	5.49	0.28	0.55	0.52	0.50	0.61	0.55	0.94
EH, Gabor	140	0.12	1.89	0.20	0.54	0.52	0.48	0.55	0.54	0.93
FOH, RILBP	612	0.27	3.19	0.49	0.55	0.52	0.50	0.60	0.55	0.94
LL, SCH	128	0.15	33.29	0.14	0.55	0.52	0.50	0.62	0.55	0.94
FOH, Gabor	636	0.21	2.60	0.49	0.54	0.52	0.49	0.57	0.54	0.93
FCH, LBP	381	2.54	23.03	0.33	0.54	0.52	0.50	0.61	0.54	0.93
FCH, Tamura	143	2.58	11.64	0.16	0.54	0.52	0.48	0.63	0.54	0.93
CL, FCH	158	2.52	34.83	0.15	0.56	0.52	0.51	0.64	0.56	0.94
OH, SCH	128	0.14	204.69	0.14	0.56	0.52	0.51	0.60	0.56	0.94
CEDD, FCH	269	2.53	92.63	0.24	0.55	0.52	0.50	0.63	0.55	0.94
EH, OH	144	0.16	2.85	0.16	0.55	0.52	0.50	0.55	0.55	0.94
FCTH, JCH	384	0.92	3.32	0.36	0.55	0.52	0.50	0.59	0.55	0.94
FCH, RILBP	161	2.57	59.54	0.16	0.54	0.52	0.49	0.62	0.54	0.93
OH, SC	128	0.15	9.86	0.14	0.55	0.52	0.50	0.58	0.55	0.94
FCH, LL	189	2.55	51.11	0.18	0.54	0.51	0.49	0.62	0.54	0.93
SCH, Tamura	82	0.18	170.03	0.11	0.54	0.51	0.49	0.59	0.54	0.93
JCH, OH	256	0.95	2.66	0.25	0.53	0.51	0.48	0.59	0.53	0.93
FCH, OH	189	2.55	172.26	0.18	0.54	0.51	0.49	0.61	0.54	0.93
FOH, JCH	768	1.03	4.53	0.69	0.52	0.50	0.47	0.59	0.52	0.93
JCH, PHOG	822	1.03	6.02	0.80	0.51	0.49	0.45	0.56	0.51	0.93
BPP, Gabor	816	0.57	4.18	1.17	0.51	0.49	0.45	0.53	0.51	0.93
PHOG, SC	694	0.23	5.32	1.06	0.51	0.49	0.45	0.54	0.51	0.93

Table A.8: Average performances over different feature combinations on the **Kvasir** dataset (cont.)

Feature	Feature Space *length*	Prepare *s/IMG*	Train *s*	Predict *ms/IMG*	ACC	F1	MCC	PREC	REC	SPEC
OH, PHOG	694	0.24	4.81	0.91	0.52	0.49	0.46	0.56	0.52	0.93
LBP, OH	320	0.15	3.72	0.31	0.51	0.49	0.46	0.56	0.51	0.93
JCH, SC	256	0.93	3.07	0.24	0.51	0.49	0.45	0.57	0.51	0.93
Gabor, JCH	252	0.91	3.39	0.24	0.51	0.49	0.45	0.55	0.51	0.93
CEDD, JCH	336	0.93	3.20	0.31	0.52	0.49	0.47	0.58	0.52	0.93
EH, JCH	272	0.94	2.86	0.26	0.50	0.48	0.45	0.57	0.50	0.93
OH, RILBP	100	0.18	47.62	0.12	0.51	0.48	0.46	0.55	0.51	0.93
RILBP, SC	100	0.17	9.19	0.11	0.51	0.48	0.45	0.56	0.51	0.93
Gabor, SC	124	0.11	1.70	0.13	0.50	0.48	0.45	0.55	0.50	0.93
Gabor, OH	124	0.12	12.34	0.14	0.52	0.48	0.46	0.54	0.52	0.93
LL, SC	128	0.15	1.97	0.15	0.49	0.47	0.43	0.53	0.49	0.93
LBP, SC	320	0.14	1.83	0.29	0.51	0.47	0.45	0.54	0.51	0.93
BPP, JCH	948	1.39	5.66	0.88	0.49	0.47	0.44	0.54	0.49	0.93
BPP, EH	836	0.60	5.10	1.24	0.48	0.47	0.41	0.51	0.48	0.93
JCH, RILBP	228	0.97	2.90	0.22	0.49	0.47	0.43	0.54	0.49	0.93
LL, OH	128	0.16	10.97	0.15	0.50	0.47	0.44	0.53	0.50	0.93
JCH, Tamura	210	0.98	2.87	0.23	0.48	0.46	0.42	0.54	0.48	0.93
JCH, LL	256	0.95	2.81	0.25	0.48	0.46	0.42	0.53	0.48	0.93
FOH, Tamura	594	0.28	201.35	0.60	0.50	0.46	0.44	0.54	0.50	0.93
Gabor, PHOG	690	0.20	5.57	1.07	0.47	0.46	0.41	0.50	0.47	0.92
CL, Tamura	51	0.17	766.45	0.08	0.48	0.45	0.42	0.53	0.48	0.93
JCD, Tamura	186	0.20	662.01	0.19	0.49	0.45	0.43	0.54	0.49	0.93
EH, PHOG	710	0.24	5.34	1.11	0.46	0.44	0.39	0.49	0.46	0.92
BPP, RILBP	792	0.63	4.87	0.92	0.45	0.44	0.38	0.46	0.45	0.92
JCH, LBP	448	0.94	3.80	0.42	0.47	0.44	0.41	0.55	0.47	0.92
BPP, LBP	1012	0.59	5.47	1.37	0.45	0.43	0.38	0.47	0.45	0.92
EH, RILBP	116	0.18	4.66	0.14	0.46	0.43	0.39	0.50	0.46	0.92
FCTH, Tamura	210	0.17	972.59	0.21	0.48	0.43	0.42	0.51	0.48	0.93
EH, LL	144	0.16	2.13	0.22	0.44	0.43	0.37	0.47	0.44	0.92
Gabor, RILBP	96	0.15	10.85	0.11	0.45	0.43	0.37	0.45	0.45	0.92
BPP, PHOG	1386	0.68	9.46	2.14	0.44	0.43	0.37	0.46	0.44	0.92
BPP, LL	820	0.61	5.54	1.22	0.44	0.42	0.37	0.46	0.44	0.92
CEDD, Tamura	162	0.17	364.91	0.17	0.46	0.42	0.41	0.54	0.46	0.92
EH, LBP	336	0.15	2.09	0.38	0.44	0.42	0.38	0.49	0.44	0.92
Gabor, LBP	316	0.11	1.93	0.31	0.45	0.42	0.38	0.47	0.45	0.92
SC, Tamura	82	0.18	1142.28	0.11	0.45	0.41	0.38	0.49	0.45	0.92
OH, Tamura	82	0.19	1847.96	0.11	0.45	0.40	0.39	0.53	0.45	0.92
PHOG, RILBP	666	0.26	5.81	0.94	0.42	0.40	0.35	0.46	0.42	0.92
LBP, PHOG	886	0.23	6.86	1.35	0.41	0.39	0.33	0.44	0.41	0.92
LL, PHOG	694	0.25	5.60	1.09	0.40	0.38	0.33	0.44	0.40	0.91
Gabor, LL	124	0.13	2.96	0.17	0.40	0.37	0.33	0.41	0.40	0.91
EH, Tamura	98	0.19	77.41	0.13	0.40	0.37	0.32	0.44	0.40	0.91
LBP, LL	320	0.15	2.39	0.39	0.39	0.36	0.31	0.41	0.39	0.91
PHOG, Tamura	648	0.27	6.23	0.75	0.38	0.36	0.31	0.42	0.38	0.91
BPP, Tamura	774	0.64	7.11	0.83	0.38	0.35	0.31	0.43	0.38	0.91
LBP, RILBP	292	0.17	3.45	0.27	0.38	0.34	0.30	0.40	0.38	0.91
LL, RILBP	100	0.19	14.35	0.13	0.34	0.31	0.26	0.38	0.34	0.91
Gabor, Tamura	78	0.16	1965.14	0.11	0.34	0.31	0.27	0.40	0.34	0.91
RILBP, Tamura	54	0.22	2903.01	0.09	0.33	0.29	0.25	0.37	0.33	0.90
LBP, Tamura	274	0.18	828.08	0.30	0.32	0.28	0.24	0.36	0.32	0.90
LL, Tamura	82	0.20	1949.00	0.16	0.29	0.26	0.20	0.36	0.29	0.90

Table A.8: Average performances over different feature combinations on the **Kvasir** dataset (cont.)