Project 2

Computational Physics and Machine Learning

Wenxue Cao, Fahimeh Najafi, and Tómas Zoëga

Abstract

Two different convolution neural network (CNN) architectures with two different optimization schemes and a support vector machine (SVM) with three different kernels were used for content-based image classification. We used k-fold cross-validation to validate our models and then applied the best set-ups to classify the images in the well known FashionMNIST and CIFAR10 datasets. We found that the SVM gives a maximum accuracy when a radial basis function (RBF) kernels is used, 88% for FashionMNIST and 54% for CIFAR10. The CNN algorithms performed best when using the so-called Adam optimizer. One of the architectures, termed CustomCNN1, achieved 90% and 60% accuracy for FashionMNIST and CIFAR10 respectively, while the other, termed CustomCNN2, achieved 86% and 52%. However, while CustomCNN1 and SVM take about equally long time to perform a task, CustomCNN2 needs only about a fifth of that time. When compared to the existing literature, our SVM code gave similar results as benchmark experiments. Our CNN models, on the other hand, performed considerably worse than other architectures in the literature, indicating that CNN's are indeed a great option for image classification and that there is room for improvement in our CNN algorithms.

Contents

1 Introduction

Content-based image classification refers to the practice of assigning an image to a specific class based on various characteristics of the image, such as color or texture [1]. Examples of this include identifying handwritten numbers and letters, distinguishing between pollen from different plant species, recognizing different land cover types form satellite images or even recognize human facial expressions. It is often simple for our human brains to perform this kind of classification and we do it all the time. The reader of this report is for example currently classifying letters and words and assigning meaning to them. Other classification tasks might be more difficult for us and better suited for computer algorithms. This could for example be analysis of satellite images which consist of multiple colour channels. Another issue we might face is a huge number of images to be classified. It can take a lot of effort and time for a human being to go through one million images and classify them and in that case a computer algorithm could be of great help. It could potentially both work faster than a human and give more accurate results.

There exist many types of algorithms computers can use to classify images. Examples include random forests, neural networks, logistic regression, maximum likelihood, and k-means clustering. In this report we will focus on two of them. Namely support vector machines and convolutional neural networks.

Support vector machines have long been used for image classification. This method is attractive since it can yield quite good results using relatively small training datasets. This is because the discriminator used in support vector classification is constructed based on only the training samples which lie closest to said discriminator [2]. However, in recent years neural networks have become increasingly popular to solve all kind of data science problems. In particular, convolutional neural networks (CNN's) have been proven to be very effective for image classification and are today one of the most popular methods for that task [3].

Our goal in this report is to compare different methods of content-based image classification. In section 2 we introduce the theory of the fundamental concepts used in this work, in section 3 we describe our datasets and methods, in section 4 we present our results and discuss them, and finally we summarize our findings in section 5.

2 Theory

2.1 Neural networks

It is very intuitive to get inspiration from the the main natural system that can learn and understand, i.e. biological brain, for building a system with such capabilities. This has been done by modelling the neurons in a the brain. This the main idea behind neural networks. Similar to a biological neuron, a node in a neural network receives a signal and based on its value, produces an output. In an artificial neural network the output of a node is a non linear function function of the sum of the inputs. A network can be consisted of different number of nodes and more than one layer of them. The term deep neural network is used when there are several layers in a network, where the output of each layer goes into the next layer, and it's the last layer of the network which directly provides the output of the system.

The input to a node can be a single or a set of numbers. It is usually denoted as a vector. The operation performed on a input vector $\boldsymbol{h}$ in a single layer of a neural network can be written as:

$$\hat{\boldsymbol{y}} = \sigma(\boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b}) \tag{1}$$

Where $\hat{\boldsymbol{y}}$ is the output of the layer, $\boldsymbol{W}$ is the weight matrix, $\boldsymbol{b}$ is the bias vector, and σ is the activation function.

The main task of the activation function is to introduce non-linearity to the system. In section 2.3 we will talk about the activation function used in this work.

The process of learning for a neural network is an iterative optimization problem when we try to find the best weight matrix that results in a system that can perform a regression or classification problem. This involves a loss function that is a measure of how good is the model doing at each stage and an optimizer which updates the weights so that the loss is decreased after each iteration.

2.2 Convolutional neural networks

Now that we have a general understanding of neural networks, we should look into how they can be used for more specific problems and how can we modify them to make them perform better based on the type of question in hand.

Here we have chosen the cases where input of our model is in the form of images. Images are made of pixel values arranged in a 2 dimensional array if they are black and white (single channel) or a three dimensional array for colour pictures (3 channels). The first thing that come to mind is that this type of data can be transformed into an 1 dimensional array, i.e. a vector, by putting all the pixel values in one row/column. We need to do this if we want to use this type of data as the input to our neural network as described in the last section.

But if we do this, we are loosing some very important information in our images. If you think about the way we see images, we do not analyse the value of each pixel (brightness and colour) individually. We see them as a whole. Close-by pixels might correspond to a single object for example. More technically we can say there is spatial correlation between pixels. This will be lost if we only put these values in a line and give them to a standard neural network.

Another issue with using this approach is that images normally have a large number of pixels. If you calculate the number of parameters that have to be computed for an average image size and a standard neural network, you will get a very large number.

If we can find a way to capture the spatial correlations and reduce the dimensionality of our model, we will definitely get better and faster result for problems with image data as input.

Convolutional neural networks [4] have these properties and have proved to be very successful in the field of image analysis. The term **convolution** here indicates that the mathematical operation called convolution is employed in this class of networks. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. [5]

Like a typical ordinary neural network, a convolutional neural network can be consisted of different number of layers. The main difference is that instead of learning the values for weights corresponding to each node in the fully connected layers, here the model learns a number of kernels in each layer. In the next section we will explain the convolution operation both mathematically and intuitively.

2.2.1 What is convolution?

Convolution can be thought of as an operation that calculates a weighted average of a function that gives more weight to recent measurements. It can be formulated as an integral:

$$s(t) = \int x(a)w(t-a)da \tag{2}$$

Where x is the function we are averaging and w is the weight function. Here, w needs to be a valid probability density function, or the output will not be a weighted average. Also, w needs to be 0 for all negative arguments, or it will look into the future. [5] This operation is usually denoted with an asterisk:

$$s(t) = (x * w)(t) \tag{3}$$

But this interpretation of weighted average is not the only functionality of convolution operation. In general, in the integral defined in equation 2, *x(a)* is called the **input** and *w(a)* is called the **kernel**. [5]

In real application the input of the convolution operation is generally a discrete set of sequential data. Therefor we need to use the discrete form of equation 2:

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{4}$$

Now lets go back to the case of having images as input data. If we call this 2 dimensional data I and the 2 dimensional kernel used in convolution K, equation 4 can also be written as:

$$S(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{5}$$

But what is the interpretation of the convolution operator when it is applied on image data? As you can tell from Eq. 5, the output of this operation, S is also a 2D array, hence an image. So we can think of this operation as applying a filter on an image. We can not talk about what this does to how the input image visually, as it depends on the kernel.

This property is what makes CNNs a very sensible choice of model for image data. In many image processing tasks, you can see that filters/transformations play a big role there. For example applying convolution with some specific kernel on an image can be used to detect the edges in that image, which is a very important feature of the images. When we say that in CNNs kernels are learned in the process of training the network, it means that we are automating the feature extraction stage, and that is what makes this method very practical for any machine learning task on images.

2.3 Activation functions

Every convolutional layer is followed by an detector stage where an activation function is applied. The purpose of the activation function is to introduce non-linearity to the model. Convolution is a linear operation, similar to the matrix multiplication in a fully connected layer. This step is responsible make a neuron active. In other words it effects how the output of the neuron will effect the next layer (or the final output if it's in the last layer).

An activation function must me continuous and differentiable. It is also important that it is not computationally expensive and it doesn't result in vanishing gradient problem. In general this function is taken to be a nonlinear function to do nonlinear regression and solve classification problems that are not linearly separable.

The most common type of activation function used after convolutional layers is the ReLU function. But for the last layer of a classifier usually a softmax function is used after the fully connected layer. For the output layer, the softmax activation function is generally a good choice for classification tasks when the classes are mutually exclusive [6].

2.3.1 ReLU

The ReLU function can be expressed as

$$f(x) = max(0, x) \tag{6}$$

The advantages of the ReLU activation function are that it is faster than other activation functions and it does not saturate for large input values as opposed to the logistic function or the hyperbolic tangent function, which saturate at 1 [6]. The ReLU function is illustrated in Fig. 1.

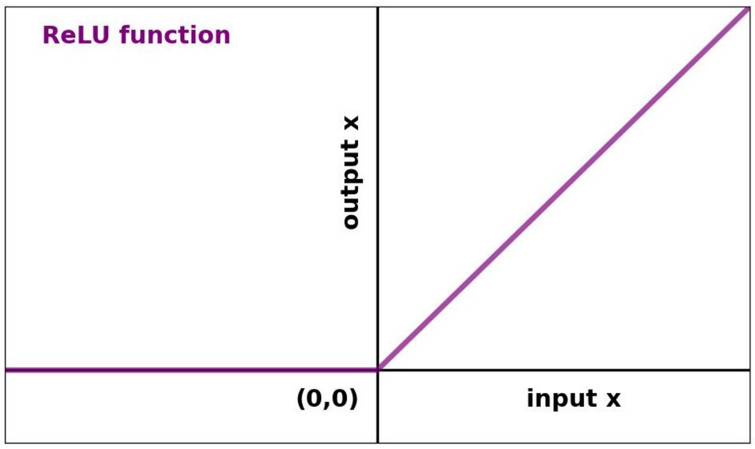


Figure 1: *An illustration of the ReLU activation function.*

2.3.2 Softmax

Another popular activation function is the softmax function:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{N} z_j} \tag{7}$$

where $\sigma(\mathbf{z})_i$ is the i^{th} element of the output of the softmax function and z_i is the i^{th} element of the input vector. N is the length of the input vector.

2.4 Pooling

In a CNN after the convolutional and activation (detector) layer, a pooling layer is added to decrease the dimensions of the output (feature maps) even more than what was achieved by using a convolutional layer. A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs [5]. This summary statistic is usually either the maximum (max pooling [7]) or the average value (average pooling) of the given window, see Fig. 2 for examples.

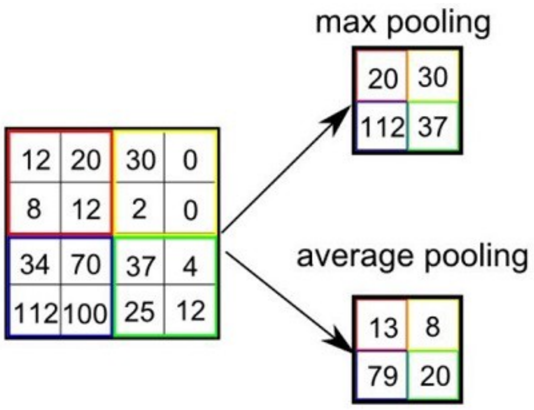


Figure 2: *Max pooling and average pooling illustrated.*

There is also one other advantage of applying a pooling function apart from reducing the dimensionality. By presenting each group of nearby values with one value, we reduce the sensitivity of the model to small changes in input. For example if one value changes in a 3-by-3 window, there is a good chance the the result of max pooling on that window does not change. This means that pooling reducing the chance of overfitting too.

2.5 Loss function

In classification problems, loss functions quantify the effects of misclassifications. One common way to measure the loss is to use cross-entropy. It is a measure of the difference between two probability distributions for a given random variable or set of events and can be expressed by the following equation:

$$H(p,q) = -\sum_x p(x)\log(q(x)) \tag{8}$$

where p and q are probability distributions for the variable x.

2.6 Support vector machines

Support vector machines (SVM's) are useful tools when classifying data. They are based on the idea that different classes can be separated using a decision boundary.

2.6.1 Hyperplanes as decision boundaries

A hyperplane is an m dimensional structure in an $m+1$ dimensional space which fulfills the equation

$$f(\mathbf{x}) = \mathbf{x}^T\boldsymbol{\beta} + \beta_0 = 0. \tag{9}$$

Here, $\mathbf{x}$ is the input vector (pixel values in the case of image classification), $\boldsymbol{\beta}$ a weight vector, and β_0 a bias [8]. As we can see from Eq. 9, hyperplanes are linear constructs and they are used as decision boundaries in the context of support vector machines.

Although SVM's can be used to separate data into multiple different classes, they are fundamentally designed to deal with a binary classification [8]. That is, determine on which side of a decision boundary an item lies. We will take that line in the following discussion and introduce the theory behind SVM's by assuming only two classes.

Let's assume that for $i = 1, 2, ..., n$ we have data points $\mathbf{x}_i$ with labels $y_i = \{-1, 1\}$. In order to separate the data into the two classes, a decision boundary in the form of a hyperplane is constructed, see Eq. 9. Data points giving $f(\mathbf{x}_i) > 0$ are classified as $\hat{y}_i = 1$, and for $f(\mathbf{x}_i) < 0$, $\hat{y}_i$ is set to -1. Here, $\hat{y}_i$ indicates a predicted class. If $\hat{y}_i$ and y_i have the same value, the data has been correctly classified. Otherwise misclassified.

If a datapoint is misclassified, the product of the true label, y_i, and the prediction, $f(\mathbf{x}_i)$, is negative. That is $y_i f(\mathbf{x}_i) < 0$. In order to quantify all the misclassifications in the dataset, the following cost function can be calculated:

$$E(\boldsymbol{\beta}, \beta_0) = -\sum_{i\in misc.} y_i f(\mathbf{x}_i) = -\sum_{i\in misc.} y_i(\mathbf{x}_i^T\boldsymbol{\beta} + \beta_0). \tag{10}$$

Here, the sum is over all misclassified datapoints. Note that since this cost function is always positive, minimizing it gives the optimal coefficients $\boldsymbol{\beta}$ and β_0 for the decision boundary [9].

2.6.2 The margin and the regularization parameter

The distance between the decision boundary and the nearest datapoint is called *margin*. For the optimal hyperplane, the margin is equal for each class. That is, the distance between the hyperplane and the nearest datapoint of one class is equal to the distance between the hyperplane and the nearest datapoint of the other class [8]. It is not always possible to construct a hyperplane which separates the classes completely. Hence we need to take data into account which is either misclasified or lies within the margin.

The optimization problem of minimizing E in Eq. 10 can be rewritten as

$$\min_{\boldsymbol{\beta},\beta_0} \left(\frac{1}{2} \|\boldsymbol{\beta}\|^2 \right), \text{ subject to } y_i f(\mathbf{x}_i) \geq 1, \tag{11}$$

see for example chapter 7 in Bishop [8]. Let's introduce a slack variable, ξ_i, for each datapoint. For a correctly classified point which lies outside the margin, $\xi_i = 0$. Otherwise, $\xi_i > 0$, increasing in value as the distance from the correct margin grows. Then the minimization problem in Eq. 11 can be extended to

$$\min_{\boldsymbol{\beta},\beta_0} \left(\frac{1}{2} \|\boldsymbol{\beta}\|^2 + C \sum_i \xi_i \right), \text{ subject to } y_i f(\mathbf{x}_i) \geq 1 - \xi_i. \tag{12}$$

Here, C is a constant larger than 0. It called the *regularization parameter* and controls the importance of the slack variable ξ_i [8].

2.6.3 Remapping data and kernel functions

Since support vector machines use linear decision boundaries, that is hyperplanes, it can be difficult to separate classes in a satisfying way. This can be improved by mapping the original data into a higher-dimensional space using so-called kernel functions. In general, a kernel can be expressed as a dot product:

$$K(\mathbf{x}, \mathbf{x}') = h(\mathbf{x})^T h(\mathbf{x}') = \langle h(\mathbf{x}), h(\mathbf{x}') \rangle \tag{13}$$

where h is some mapping function. As noted in Hastie et al. [9], we only need the knowledge of the kernel function and not the mapping function per se. For example, if the mapping function is linear, the kernel can be expressed as

$$K(\mathbf{x}, \mathbf{x}') = 1 + \langle \mathbf{x}, \mathbf{x}' \rangle. \tag{14}$$

A wide range of kernels is used in the application of support vector machines, for example based on polynomials or radial basis functions. We will not elaborate more on kernels here but further details can be found in the book by Hastie et al. [9] and the article by Hofmann et al. [10]. An example of how remapping can enable a linear decision boundary to separate data can be seen in Fig. 3. Originally the data is one dimensional. Then it remapped into two dimensions according to $y = x^4$.

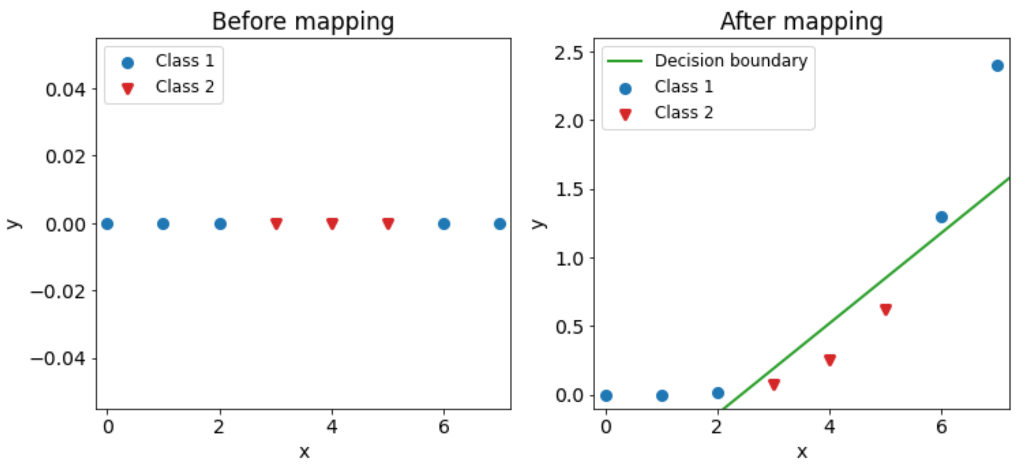


Figure 3: *An example of how data can be separated with a linear decision boundary after remapping. Here, the one dimensional datapoints x have been remapped into two dimensions using the 4th order polynomial $y = x^4$.*

2.6.4 Extending SVM's to multiple classes

Even though support vector machines are designed to deal with binary classification, their application can be extended to multiple classes. This is not straight forward but one way is to construct an n-class decision boundary. It can be written as

$$f_n(\mathbf{x}) = \mathbf{x}^T \boldsymbol{\beta}_n + \beta_{n0}. \tag{15}$$

The decision boundary between two classes i and j is then $f_i(\mathbf{x}) = f_j(\mathbf{x})$. One way of including more than two classes is to us the so-called *one-versus-one* method. In this method, one decision boundary is constructed for each pair of classes. Each point is then treated by all the different decision boundaries and is finally assigned to the class which most often comes up. For further details, see section 4.1.2 in Bishop [8].

2.7 Optimizers

Solving numerical problems with iterative processes usually involves minimizing some kind of a cost function. One way of doing that is to use *gradient descent* (GD). In their essence, GD methods start at a random point somewhere on the cost function and then they follow the steepest negative gradient until a minimum is reached. This gradient descent can be implemented in many different ways which vary in complexity, reliability, and flexibility [11]. In this project we will use two variants of GD, namely stochastic gradient descent with momentum, and the so-called Adam optimizer.

2.7.1 Stochastic gradient descent with momentum

On its most basic form, GD calculates the gradient of the cost function, E, at a given set of parameters, $\boldsymbol{\beta}$, according to the equation

$$(\Delta\boldsymbol{\beta})_{i+1} = -\gamma(\nabla_\beta E)_i. \tag{16}$$

Here, $\Delta\boldsymbol{\beta}$ is the change in the set of parameters at iteration step i, $\nabla_\beta E$ the gradient of the cost function with respect to $\boldsymbol{\beta}$, and γ the learning rate. When the GD algorithm has evaluated all the data available, the set of parameters $\boldsymbol{\beta}$ is updated. This process is called an *epoch*, and the number of epochs iterated over is usually predefined. Although using all available data simultaneously is accurate, it can be computationally expensive. One way of getting around this is to update the set of parameters for each training example instead of waiting until all of them have been evaluated. This is called *stochastic gradient descent* (SGD) and speeds the process up considerably. The term *stochastic* stems from the practice of randomly choosing training examples. In practice, the entire dataset is often divided into batches and the set of parameters is updated after evaluating each batch. This is called *mini-batch gradient descent* although it is often referred to as SGD.We will follow that line in this report.

As can be seen from Eq. 16, the learning rate γ is important when it comes to updating $\boldsymbol{\beta}$. A large learning rate helps reaching the minimum of the cost function quickly but it also invites the danger of overshooting. Then the GD method might repeatedly step over the minimum without decreasing the cost function. That is, not converge. Using a small learning rate solves this problem. However, if the learning rate is too small, the algorithm becomes very slow. It is even possible that the minimum of the cost function might not be reached at all before the iterative process is over. Furthermore, if a cost function has more than one minima, a small learning rate can lead the algorithm to be trapped in a local minimum instead of the global one. This trade-off between large and small learning rates can be compensated for by either allowing the learning rate to vary between iterative steps or introduce other factors which control the step size $\Delta\boldsymbol{\beta}$. A momentum term is often utilized to this effect [12]. Then Eq. 16 becomes:

$$(\Delta\boldsymbol{\beta})_{i+1} = -\gamma(\nabla_\beta E)_i + p(\Delta\boldsymbol{\beta})_i \tag{17}$$

where p is the momentum parameter. By including it, the algorithm is able to retain information about a previous iteration step and apply it to the next one. For example, if the GD algorithm is moving across a relatively even surface, it gains momentum and is not stopped by potential minima on the way. And when it reaches what might be a global minimum, instead of repeatitly overstepping it and not coverging, the momentum term will gradually decrease with as the gradient changes signs at each overstep, resulting in a precise estimate of the minimum.

2.7.2 Adam

The *Adam* algorithm (the name comes from *adaptive moment estimation*) was introduced by Kingma and Ba in 2014 [13] and has been used very successfully since. According Kingma and Ba, Adam is computationally inexpensive while also performing well. It is easy to implement and works well for large datasets and high numbers of parameters. Adam is a GD method and updates the parameters $\boldsymbol{\beta}$ according to

$$(\Delta\boldsymbol{\beta})_{i+1} = -\frac{\gamma\hat{m}_i}{\sqrt{\hat{s}_i + r}}, \tag{18}$$

where r is a small regularization constant to prevent divergence, $\hat{m}_i = m_i/(1-\eta_1^i)$ and $\hat{s}_i = s_i/(1-\eta_2^i)$, and η_1 and η_2 are decay rates for the estimates of the first and second order moments of the gradient. Here, η^i denotes η to the power of i, and i is the number of the current iteration step. As before γ is the learning rate. Furthermore,

$$m_i = \eta_1 m_{i-1} + (1 - \eta_1)(\nabla_{\boldsymbol{\beta}} C)_i, \tag{19}$$

$$s_i = \eta_2 s_{i-1} + (1 - \eta_2)\big[(\nabla_{\boldsymbol{\beta}} C)_i\big]^2. \tag{20}$$

3 Methods

3.1 Data

We apply our classification algorithms to the datasets described below.

3.1.1 FashionMNIST

The MNIST dataset was introduced in 1998 [14, 15] and consists of 70000 hadwritten numbers, ranging from 0 to 9. They are stored as single colour channel images, 28-by-28 pixels. Each class has equal amount of images and the dataset has been split by default into 60000 images for training and 10000 for testing. Due to the simplicity of these images and how easily accessable they are, MNIST has become very popular when trying out and benchmarking machine learning codes. However, it has been argued that MNIST is too simple and overused and in response to that Xiao et al [16] introduced the FashionMNIST dataset in 2017. FashionMNIST is very similar to MNIST, the images have the same number of pixels and colour channels and the datasets are of equal size. The only difference is that FashionMNIST depicts photos of clothes instead of numbers, making the images slightly more challenging to classify. For this reason we will use FashionMNIST in this report. The dataset is available for all to use on GitHub: `https://github.com/zalandoresearch/fashion-mnist`.

3.1.2 CIFAR10

The CIFAR10 dataset consists of 60000 photographs of which 50000 are used for training and 10000 for testing. They are 32-by-32 pixels large and have three colour channels. These photos are separated into 10 classes which depict various animals and vehicles [17]. CIFAR10 is quite similar to FashionMNIST but the images are more diverse and more complicated due to the three colour channels. The data is available at `https://www.cs.toronto.edu/ kriz/cifar.html`.

3.2 Neural networks

One of the methods we use for the image classification in this report are convolutional neural networks (CNN's). We construct our neural networks using functionalities from PyTorch. To build the classifiers we use *skorch* [18] which is a neural network library which enables functionalities from Scikit-Learn [19] to be used with PyTorch. We construct two CNN's and compare their results. They are called *CustomCNN1* and *CustomCNN2* and their architectures are outlined in Algorithms 1 and 2.

Algorithm 1 An overview of the architecture of CustomCNN1.

```
Input: Training images and labels
  Convolutional layer
    + Activation function: ReLU
  Convolutional layer
    + Activation function: ReLU
    + Pooling: Max pooling
    + Dropout with 0.25 probability
    + Flatten images
  Linear layer
    + Activation function: ReLU
    + Dropout with 0.5 probability
  Linear layer
    + Logarithmic softmax activation function
Output: Likelihood of image being in each class.
```

Algorithm 2 An overview of the architecture of CustomCNN2.

```
Input: Training images and labels
  Convolutional layer
    + Activation function: ReLU
  Convolutional layer
    + Activation function: ReLU
    + Pooling: Max pooling
    + Flatten images
  Linear layer
    + Activation function: ReLU
  Linear layer
    + Activation function: ReLU
  Linear layer
Output: Likelihood of image being in each class.
```

3.3 Support vector machines

The other method we use for the image classification is support vector machines. In essence, what we need for a classification is some number of features and a corresponding label. As described in Section 2.2 images need to be flattened into a one dimensional vector before they can be processed by a neural network algorithm. This is also the case for support vector machines.

In this report we use functionalities from the machine learning library Scikit-Learn [19] to perform the supper vector machine classification. We use the Support Vector Classifier (`SVC`) [20] with the following kernel functions:

$$\begin{aligned} \text{linear:} &\quad \langle \mathbf{x}, \mathbf{x}' \rangle, \\ \text{3rd degree polynomial:} &\quad \langle \mathbf{x}, \mathbf{x}' \rangle^3, \\ \text{radial basis function:} &\quad \exp\big(- ||\mathbf{x} - \mathbf{x}'||^2 \big). \end{aligned}$$

Scikit-Learn also includes the option to vary the regularization parameter C (see Sec. 2.6.2) and we take advantage of that. Finally it is worth mentioning that Scikit-Learn uses the one-versus-one method when dealing with more than two classes, see Sec. 2.6.4.

3.4 Accuracy

To evaluate the performance of our classification algorithms, we calculate the accuracy score A. The accuracy is defined as the ratio of correctly classified samples to the total number of samples. That is

$$A = \frac{1}{n_{samples}} \sum_i \mathbb{1}(y_i = \hat{y}_i) \tag{21}$$

where $n_{samples}$ is the total number of samples, y_i is the true label of the i-th sample and $\hat{y}_i$ the predicted label. $\mathbb{1}$ is the indicator function, taking a value of 1 when $y_i = \hat{y}_i$ but 0 otherwise [21].

3.5 k-fold cross-validation

k-fold cross-validation is a resampling method used to evaluate the performance of a prediction algorithm. The way k-fold cross-validation works is that is divides the whole dataset in question into k groups, also called folds. Then $k - 1$ of these groups are used to train the algorithm and 1 group used for testing. This process is repeated until each of the k groups has been used for testing once [22].

In this project, we use Scikit-Learn's `cross_validate` functionality to evaluate the performance of our chosen classification methods.

4 Results and discussion

4.1 Neural networks

To test our models we use two different datasets, namely FashionMNIST and CIFAR10. Both of these datasets have an official separation of training and testing data. In the case of FashionMNIST teh training data includes 60000 images but 50000 images for CIFAR10. In both cases the training dataset consists of 10000 images. Also, both datasets have ten classes and equal amount of images in each class. The we we approach our analysis is to use k-fold cross-validation on the official testing part of each dataset and from those results decide on the optimal set-ups of our models. Then we train these optimal models on the official training part of the datasets and test them on the official testing parts. The results of those final testings are displayed as confusion matrices.

4.1.1 FashionMNIST

Figs. 4 and 5 show accuracy as a function of learning rate for different model set-ups. Results from CustomCNN1 are shown in Fig. 4 and CustomCNN2 in Fig. 5. Here the focus is on the FashionMNIST dataset which has ten categories, each with equal amount of images. We use 5-fold cross-validation to assess the quality of the classification and set the number of epochs to five. In addition to varying the learnig rate, we look at different optimizers. That is, different methods of how the cost function is minimized, here the cross entropy loss. Those are stochastic gradient descent (SGD) with momentum (p) and the Adam optimizer.

As can be seen in Fig. 4, the learning rate has a strong effect on the overall accuracy. However, the same learning rate can give very different results depending on the optimizer used. If we look at the result from models which use a SGD optimizer, we see the best results when $p = 0.9$. However, in this case, the value of the momentum parameter does not have great effects on the overall accuracy if the learning rate is chosen appropriately. In all cases the maximum accuracy is about 0.8. This maximum accuracy is achieved for increasingly high learning rates as the momentum parameter p becomes smaller. This comes as a no surprise since the learning rate and the momentum parameter work together to control the step size when minimizing the cost function, see Eq. 17. For all the optimizers, the accuracy rapidly drops once the learning rate exceeds a certain threshold. This is likely the cause of an explosive gradient. By looking at Eq. 17 again, we see that that the step size also depends on the gradient of the cost function. That is, as the gradient becomes larger, the larger the step. If the learning rate becomes large enough, the optimization algorithm can start to repeatedly overstep the minimum it is trying to find and the absolute value of the gradient increases with every step. Hence we move away from the minimum of the cost function and the classification fails. This is most likely what we are seeing in Fig. 4. This sudden drop in accuracy is not observed for low learning rates in the case of CustomCNN1. Instead there is a gradual decrease in accuracy as the learning rate becomes smaller. A reason for this could be that when the learning rate is too small, a minimum of the cost function is not reached and hence the classification will not be as good as it could have been. Overall, the Adam optimizer gives the highest total accuracy and performs considerably better the the SGD methods for small learning rates.

Fig. 5 is based on the same data and the almost same models as Fig. 4. The only difference is that now a different neural network is used, CustomCNN2 instead of CustomCNN1. The overall accuracy is slightly lower for CustomCNN1 but we see similar behaviour when it comes to different optimizers and how they are effected by the learning rate. The most noticeable difference is that the total accuracy decreases much faster for small learning rates when using CustomCNN2.

Based on the results from Figs. 4 and 5, we decide to use the Adam optimizer with a learning rate of 10^{-3} for subsequent analysis.

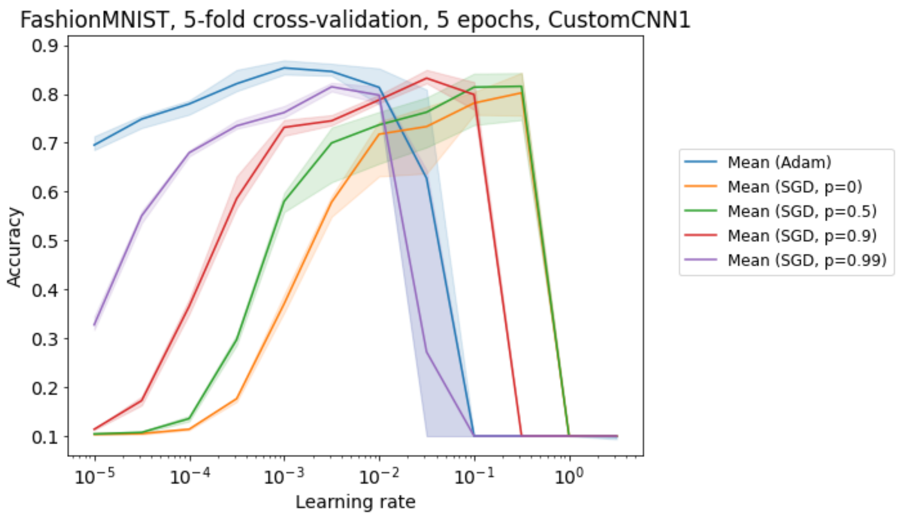


Figure 4: *The accuracy score for FashionMNIST (10000 images used) using CustomCNN1 and 5-fold cross-validation. The shaded areas indicate the ranges between cross-validation iterations. Different optimisation methods are used, both stochastic gradient descent (SGD) with momentum (p), and Adam. 5 epochs.*

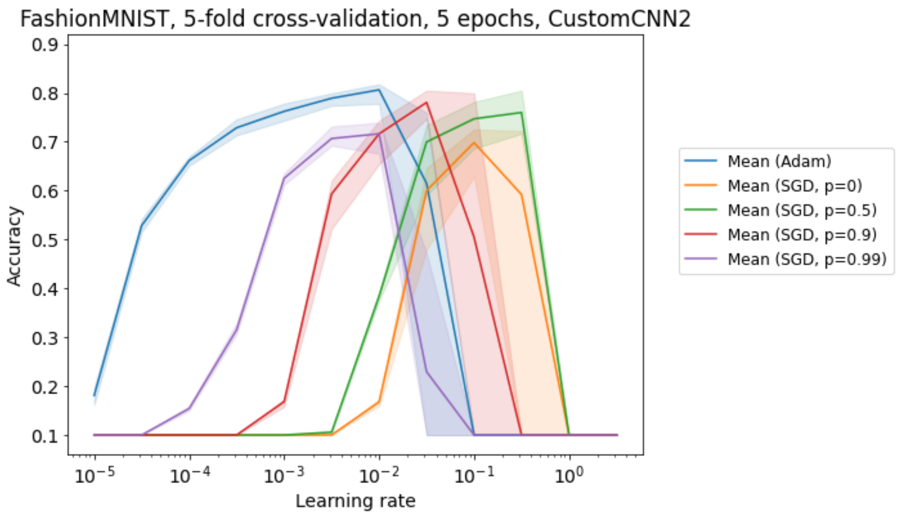


Figure 5: *The same as Fig. 4 unless here CustomCNN2 is used.*

In Fig. 6 we look at the effects of using different number of epochs. Unsurprisingly, as the number of epochs rises, so does the overall accuracy. The increase in accuracy between epochs is the greatest when the number of epochs is small but gradually levels out as the number increases. Iterating over many epochs increases accuracy but it is also computationally expensive. A balance between effectiveness and accuracy needs to be found in this case, four or five epochs seem like good choices.

In Fig. 7 the same set-up is used as in Fig. 6 but here the number of epochs is kept constant at five and the number of folds is allowed to vary. Generally speaking, increasing the number of folds increases the accuracy of the classification, both in the case of CustomCNN1 and CustomCNN2. We do, however, see that sometimes the minimum accuracy for a certain fold is suspiciously low. This is for example the case for 6-fold cross-validation for CustomCNN2. The reason for this is most likely overfitting. That is, the model fitted the five training folds too accurately and did therefore not manage to classify the remaining testing fold well enough. We could expect to see this kind of behaviour more often as the number of folds increases. This is because then the split of the original data into test and train becomes increasingly dominated by the training data. Based on the results in Fig. 7 we will use 5-fold cross-validation to validate the CNN classifications in this report.

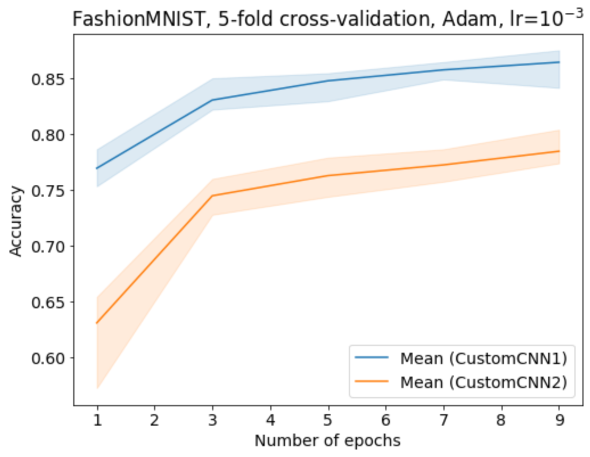


Figure 6: *The accuracy score for FashionMNIST (10000 images used) using 5-fold cross-validation and the Adam optimizer with learning rate of* 10^{-3}. *The shaded areas indicate the ranges of the cross-validation iterations. Different NN algorithms are used.*

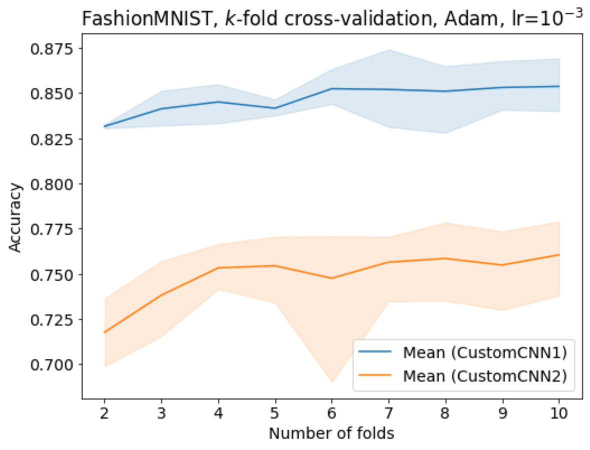
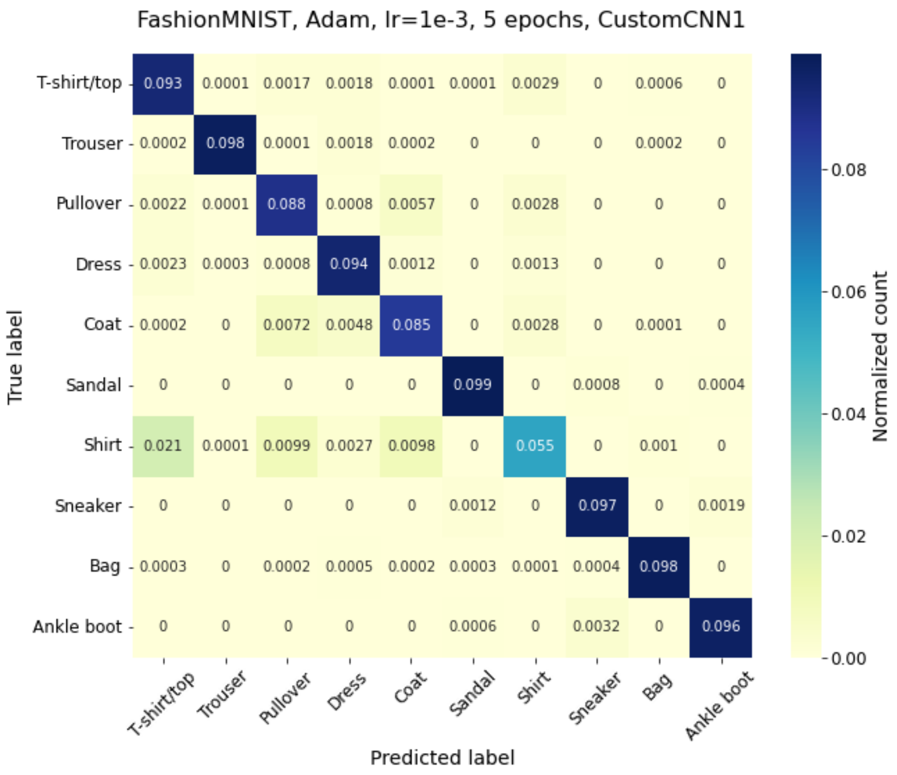


Figure 7: *Same set-up as Fig. 6 but now the number of epochs is kept at five and the number of folds allowed to vary.*

From the results described above, we figure that the best model for a classification of the FashionMNIST dataset uses the Adam optimizer with a learning rate of 10^{-3} and uses five epochs. CustomCNN1 gives higher total accuracy than CustomCNN2 but it is also considerably slower (see Section 4.3). Figs. 8 and 9 illustrate the confusion matrices resulting from that model set-up, using CustomCNN1 and CustomCNN2 respectively. 60000 thousand images were used for training the models and 10000 for testing. The overall accuracy for CustomCNN1 is 0.90 but 0.86 for CustomCNN2. CustomCNN1 also gave a higher accuracy for all but the shirt class. Shirts were also most commonly misclassified by CustomCNN1, often registered as either T-shirt/top, pullover, or coat. This was also the case for CustomCNN2 but to a lesser extent.



True label \ Predicted label	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	0.093	0.0001	0.0017	0.0018	0.0001	0.0001	0.0029	0	0.0006	0
Trouser	0.0002	0.098	0.0001	0.0018	0.0002	0	0	0	0.0002	0
Pullover	0.0022	0.0001	0.088	0.0008	0.0057	0	0.0028	0	0	0
Dress	0.0023	0.0003	0.0008	0.094	0.0012	0	0.0013	0	0	0
Coat	0.0002	0	0.0072	0.0048	0.085	0	0.0028	0	0.0001	0
Sandal	0	0	0	0	0	0.099	0	0.0008	0	0.0004
Shirt	0.021	0.0001	0.0099	0.0027	0.0098	0	0.055	0	0.001	0
Sneaker	0	0	0	0	0	0.0012	0	0.097	0	0.0019
Bag	0.0003	0	0.0002	0.0005	0.0002	0.0003	0.0001	0.0004	0.098	0
Ankle boot	0	0	0	0	0	0.0006	0	0.0032	0	0.096

Figure 8: *The confusion matrix for FashionMNIST using CustomCNN1, 5 epochs, and the Adam optimizer with learning rate* 10^{-3}. *The model was trained with 60000 images and tested with 10000. The results have been normalized to the total number of images, here 10000. The overall accuracy for the testing data is 0.90.*

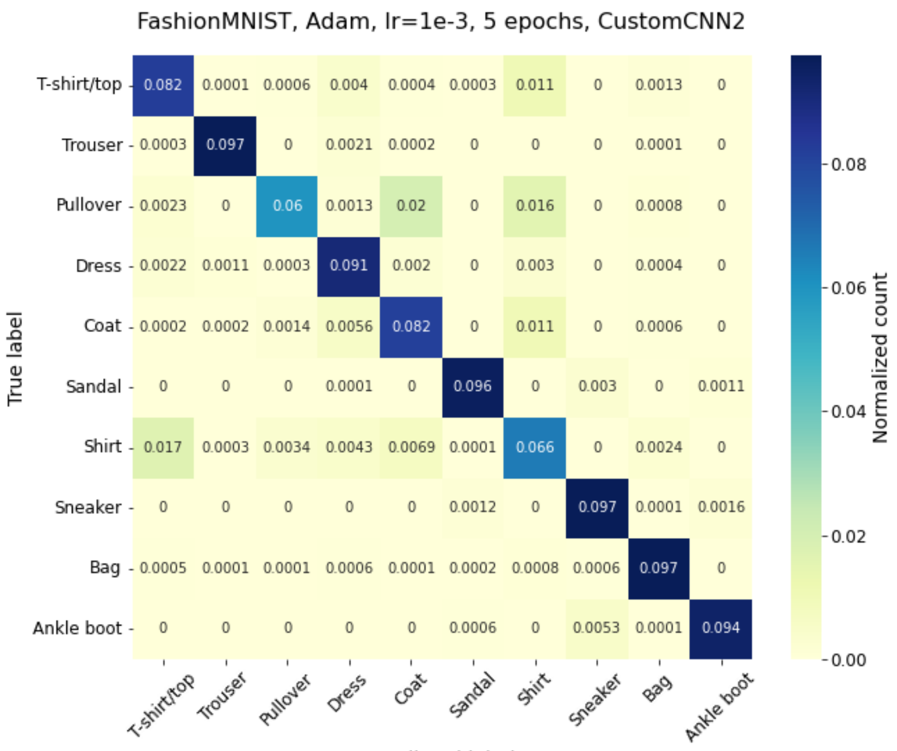


Figure 9: *The same as Fig. 8 but here CustomCNN2 is used. The overall accuracy for the testing data is 0.86.*

4.1.2 CIFAR10

The CIFAR10 images as divided into 10 equal classes but unlike FashionMNIST the images have three colour channels and slightly more pixels, 32-by32 instead of 28-by-28. The CIFAR10 images are therefore similar to the FashionMNIST images in many ways but more complicated. Figs. 10 and 11 are the same as Figs. 4 and 5 unless now the CIFAR10 dataset is used. As we can see, the overall accuracy for the classification is considerably lower for the CIFAR dataset than it is for FashionMNIST. The 5-fold cross-validation gives maximum total accuracy of 0.49 for CustomCNN1 and 0.41 for CustomCNN2. In both cases this maximum accuracy is achieved with the Adam optimizer and a learning rate close to 10^{-3}.

Another noticeable difference between the classification of the two datasets is the range of learning rates which give high accuracy. For CIFAR10, this range of learning rates is considerable smaller, meaning that the learning rate has to be chosen even more carefully in the case of CIFAR10 than FashionMNIST. Finally, if we look at the results from the SGD optimizer, we see that the maximum accuracy is achieved with $p = 0.9$ and then it decreases rapidly as p approaches 1. This was also the case with FashionMNIST but is more prominent here. And as before, the reason for this is most likely that the step when the cost function is minimized has become too large for this high values of p.

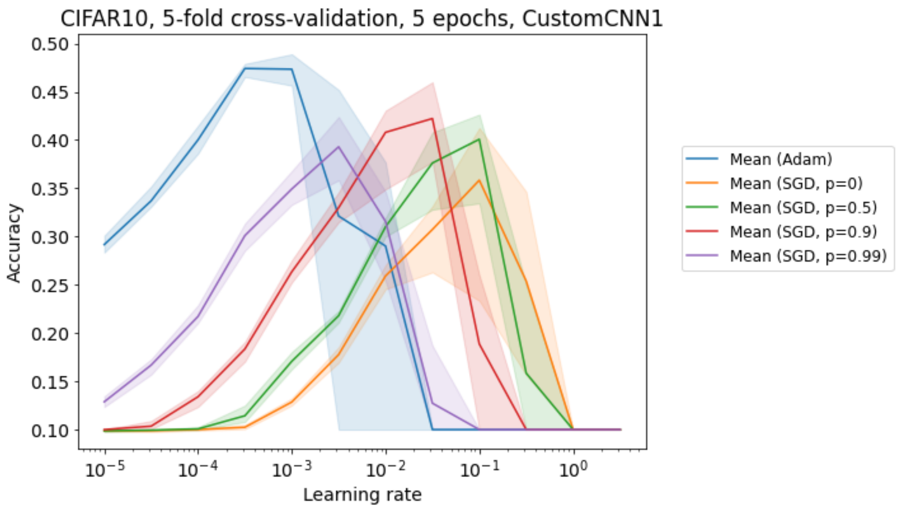


Figure 10: *The accuracy score for CIFAR10 (10000 images used) using CustomCNN1, 5-fold cross-validation, and 5 epochs. The shaded areas indicate the ranges between cross-validation iterations. Different optimisation methods are used, both stochastic gradient descent (SGD) with momentum (p), and Adam.*

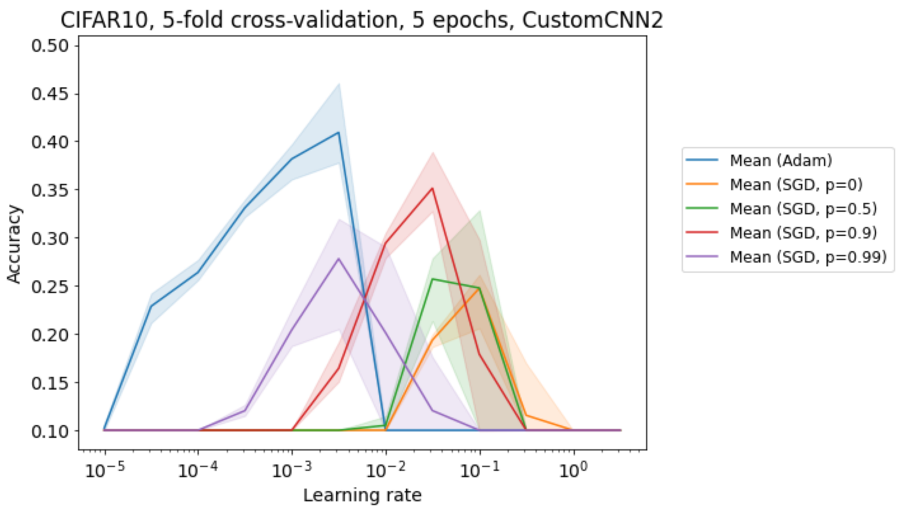


Figure 11: *The same as Fig. 11 but CustomCNN2 is used.*

Fig. 12 is the same as Fig. 6 but for CIFAR10. The results are very similar although in the case of CIFAR10, a higher number of epochs is needed to achieve the best results. Based on Fig. 12, the number of epochs should be at least seven and preferably higher. However, in our analysis we stick to five epochs due the increased computational cost when more are used.

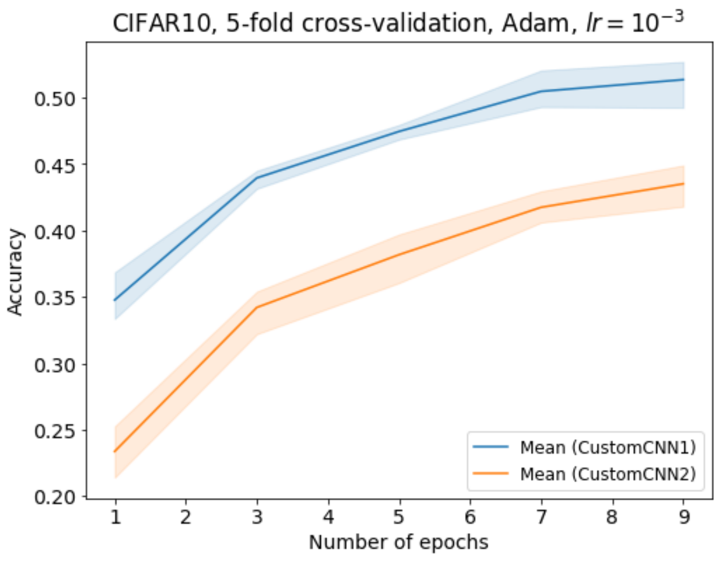


Figure 12: *The accuracy score for CIFAR10 (10000 images used) using 5-fold cross-validation and the Adam optimizer. The shaded areas indicate the ranges of the cross-validation iterations. Different NN algorithms are used.*

As with FashionMNIST we find that the best model set-up to classify the CIFAR10 images includes the Adam optimizer and a learning rate of 10^{-3}. We use that set-up and five epochs to calculate confusion matrices using CustomCNN1 and CustomCNN2, see Figs. 13 and 14 respectively. We trained the models with 50000 images and tested them with 10000. The overall accuracy turned out to be 0.60 for CustomCNN1 and 0.52 for CustomCNN2. CustomCNN1 performed better than CustomCNN2 for nine out of the ten classes. CustomCNN1 also showed smaller spread in the accuracy of the classes, meaning that the classification was overall more consistent than the one done by CustomCNN2.

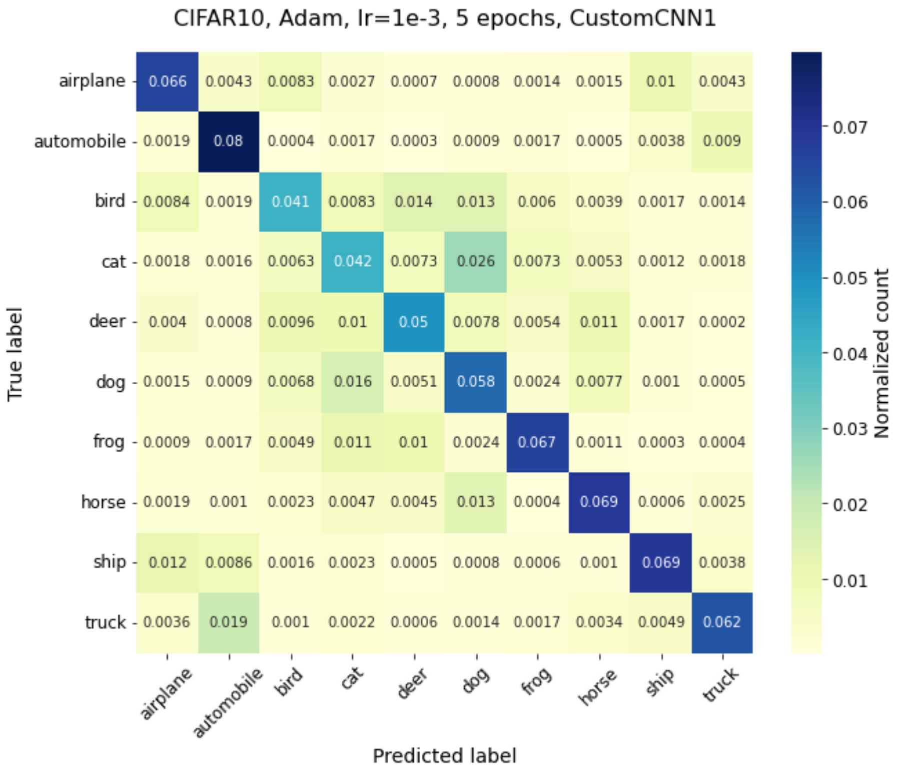


Figure 13: *Same as Fig. 8 but for the CIFAR10 dataset. The overall accuracy for the testing data is 0.60.*

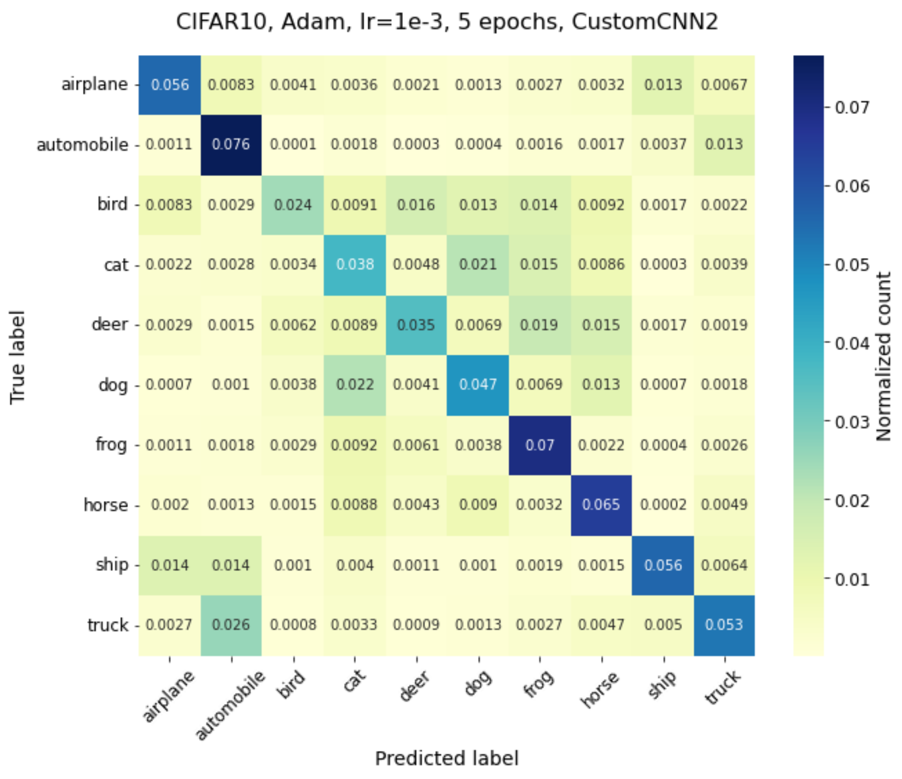


Figure 14: *Same as Fig. 13 but using CustomCNN2. The overall accuracy for the testing data is 0.52.*

4.2 Support vector machines

4.2.1 FashionMNIST

Here we use Scikit-Learn's support vector machine algorithm to classify images from the FashionMNSIT dataset. Fig. 15 shows the accuracy score for three different kernels using k-fold cross-validation. The number of folds is varied and as can be seen, the accuracy of the classification increases with increasing number of folds. However, in the case of the FashionMNIST dataset, the number of folds does not seem to be a decisive parameter. The choice of the kernel function seems to be much more important. Of the three kernels we looked at, the RBF kernel gave the best results, having a mean accuracy of about 0.85 for four or more folds. For the same range of folds, the 3rd degree polynomial kernel gives accuracy of about 0.78 to 0.79, and the linear kernel accuracy of about 0.79. A 5-fold cross-validation is often a good choice since for each iteration validation, 80% of the data is used for training and 20% for testing. This means that we have relatively large training dataset but at the same time we are not in great danger of overfitting.

Unsurprisingly, the choice of the kernel function is important when it comes to the accuracy of the classification. What is surprising is that the linear kernel (blue line) gives higher accuracy than the more complex 3rd-degree-polynomial kernel (orange line). In general, we would have expected a more complicated kernel would give better results since it has the potential to separate the data better than a simple one. This does not seem to be the case here. The reason for this is the choice of the regularization parameter C as can be seen in Fig. 16. The best results are achieved with the RBF kernel, which is the most complicated kernel we looked at. Here we refer to kernels which require more computational operation to be more complicated. For that reason, a complicated kernel such as the RBF one results in a longer computational time than a simple one like the linear kernel. Hence the choice of kernel should depend on both the accuracy of the classification and the amount of time it takes to train the classifier.

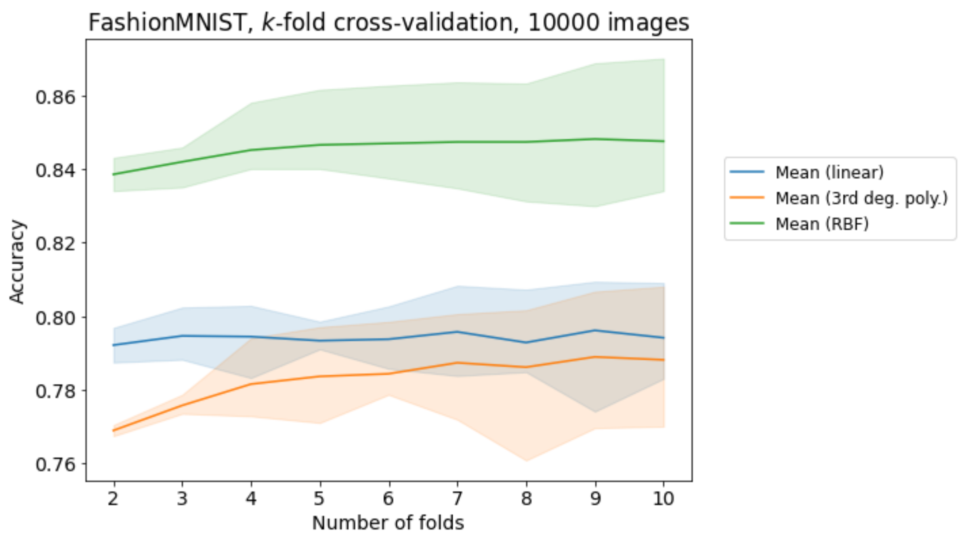


Figure 15: *The accuracy score for FashionMNIST (10000 images used) using k-fold cross-validation. The regularization parameter is $C = 1$. The shaded areas indicate the range of the cross-validation iterations.*

Based on the results from Fig. 15, we decided to stick to 5-fold cross-validation and look at the effects of varying the regularization parameter C. As we can see in Fig. 16, the regularization parameter has considerable effects on the accuracy. When using the 3rd degree polynomial and RBF kernels, the accuracy increases as the value of C increases. The opposite holds for the linear kernel. As before we see the best performance from the RBF kernel. In its case, the accuracy increases until about $C = 1$ where it plateaus at an accuracy of about 0.87. Since using a large C can lead to longer computational time we decide to stick to $C = 1$.

One thing to notice about Fig. 16 is that unlike the RBF and 3rd-degree-polynomial kernels, the accuracy when using the linear kernel decreases with increasing C. In this case, if C is low, the tolerance for misclassifed date is little and the decision boundary is constructed in such a way that most of the images lie on the correct side of the decision boundary and the margin line. This might indicate a small margin. However, once C is increased, the tolerance for images being misclassified or lie within the margin increases as well. And since the margin is maximized when the decision boundary is constructed, this can lead to lower accuracy when the model is applied to the testing data. This is because the potential of an image falling within the margin increases as the margin grows larger. However, in general we expect a model to perform better when C is increased. This is because when a regularization term is included, the occurrence of datapoints which are misclassified or lie within the margin is expected and taken into account then the decision boundary is constructed. The behaviour of the linear kernel here might indicate overfitting.

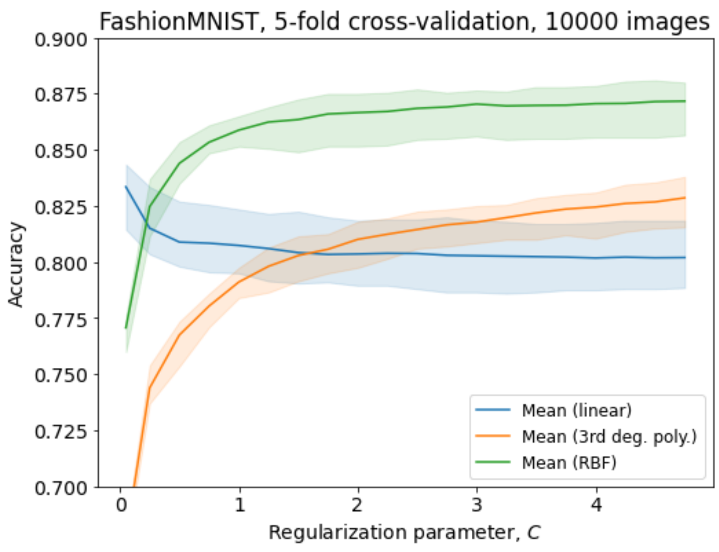
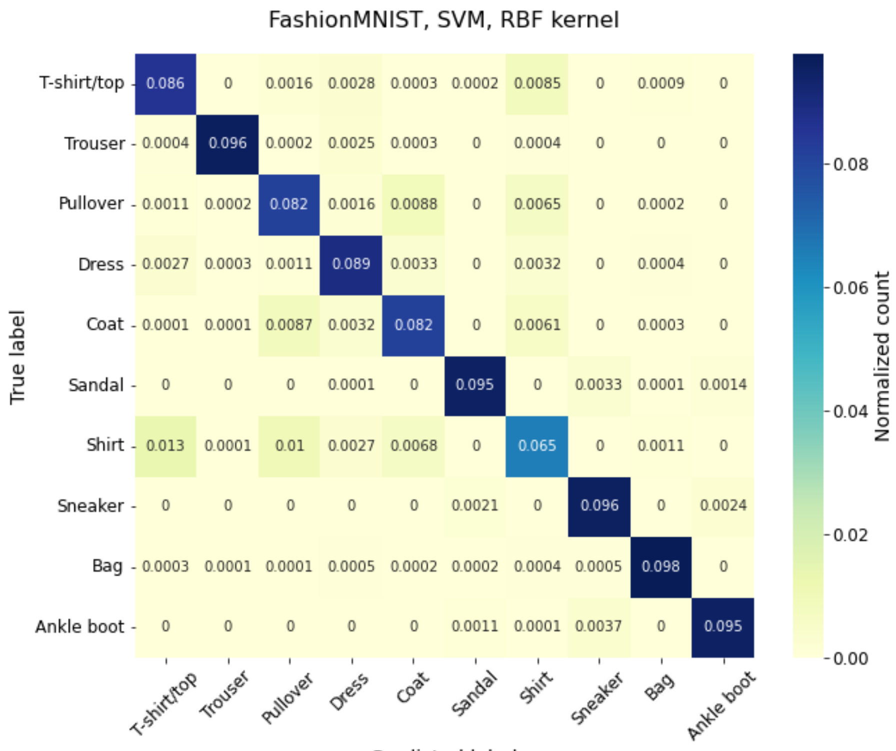


Figure 16: *The accuracy score for the FashionMNIST dataset (10000 images) using SVM and* 5*-fold cross-validation. The regularization parameter C is varied.*

Fig. 17 shows the confusion matrix for the classification of the images in the Fashion-MNIST dataset using Scikit-Learn's support vector classifier with the RBF kernel and $C = 1$. Here, we used 60000 images to train the model and then tested it on another set of 10000 images. The results have been normalized to the total number of images in the testing dataset. The total accuracy for this model is 91% for the training data and 88% for the testing data. As can be seen from the confusion matrix, the accuracy for each class ranges from 65% for shirts to 98% for bags, but mostly staying above 80%.



True \ Predicted	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	0.086	0	0.0016	0.0028	0.0003	0.0002	0.0085	0	0.0009	0
Trouser	0.0004	0.096	0.0002	0.0025	0.0003	0	0.0004	0	0	0
Pullover	0.0011	0.0002	0.082	0.0016	0.0088	0	0.0065	0	0.0002	0
Dress	0.0027	0.0003	0.0011	0.089	0.0033	0	0.0032	0	0.0004	0
Coat	0.0001	0.0001	0.0087	0.0032	0.082	0	0.0061	0	0.0003	0
Sandal	0	0	0	0.0001	0	0.095	0	0.0033	0.0001	0.0014
Shirt	0.013	0.0001	0.01	0.0027	0.0068	0	0.065	0	0.0011	0
Sneaker	0	0	0	0	0	0.0021	0	0.096	0	0.0024
Bag	0.0003	0.0001	0.0001	0.0005	0.0002	0.0002	0.0004	0.0005	0.098	0
Ankle boot	0	0	0	0	0	0.0011	0.0001	0.0037	0	0.095

Figure 17: *The confusion matrix for FashionMNIST using SVM with the RBF kernel and $C = 1$. The results have been normalized to the total number of images, here 10000. A perfect classification would results in a value of 0.1 along the diagonal an 0 elsewhere. The model was trained with 60000 images and tested with 10000. Overall accuracy for the testing data is 0.88.*

In their article, Hastie et al. [23] discuss the regularization parameter C. As they point out, if the data can be properly separated and C is large enough, then we will get the same solution for Eqs. 11 and 12 in Sec. 2.6.2 in this report.

If the classes in a training dataset can be properly separated with a decision boundary, no points are misclassified nor do they lie within the margin. Hence the slack variable ξ is equal to zero for all datapoints, see Eq. 12. If an SVM model is trained on this data, it should therefore be independent of the regularization parameter C and give 100% accuracy for the classification of the training data. However, when this model is used to classify the testing data, the accuracy will probably not be that high since some datapoints might be misclassified. But since the trained model is independent on the regularization parameter C, varying it will also not affect the classification of the testing data.

We see this in effect in Fig. 18. For very small datasets, see the red line in Fig. 18 representing 500 images, the mean accuracy after a 5-fold cross-validation is independent on C. This indicates that the classes in the training dataset, here 400 images, are completely separable. However, when validating with the remaining 100 images, some of them are misclassified and the accuracy decreases. When the dataset is increased in size, the classes become less easily seperable (at least when using a linear kernel as in Fig. 18), and the choice of the regularization parameter starts to affect the accuracy.

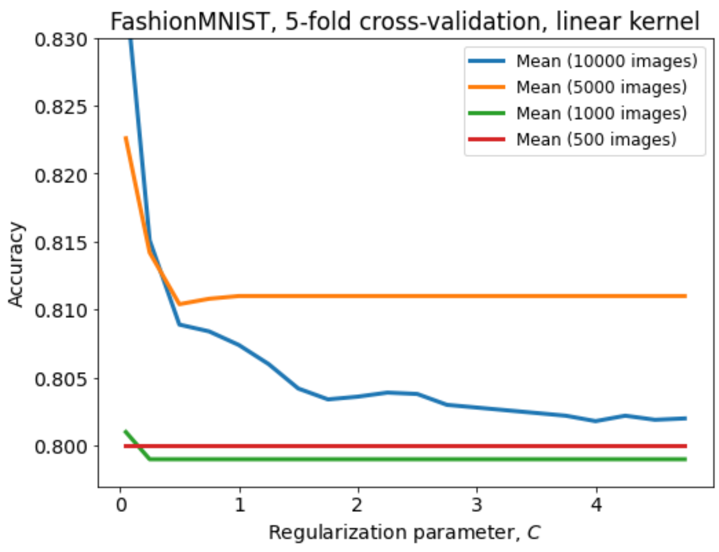


Figure 18: *The accuracy score for the FashionMNIST dataset as a function of the regularization parameter C, using 5-fold cross-validation and the linear kernel. The number of images used for the cross-validation process is varied.*

4.2.2 CIFAR10

Fig. 19 shows the same as Fig. 15 but this time for the CIFAR10 dataset. As before, the number of folds does not seem to be a decisive parameter. Also, the RBF kernel gives the best results, followed by the 3rd degree polynomial and the linear kernels in that order. However, even though we use the same number of images and the same number of classes as in Fig. 15, this time the total accuracy is much lower. For the RBF kernel, the total accuracy is about 0.46-0.47, increasing slightly with increasing number of epochs, compared to 0.85 in the case of FashionMNIST. The reason for this might be that while the two datasets have similar number of pixels, the FashionMNIST images only have a single colour channel while the CIFAR10 images have three, making the latter larger and more complex. In fact, the CIFAR10 images have about four times as many parameters to optimize ($n_{par,cifar10} = 32 \times 32 \times 3 = 3072$) compared to the FashionMNIST ones ($n_{par,fmnist} = 28 \times 28 \times 1 = 784$).

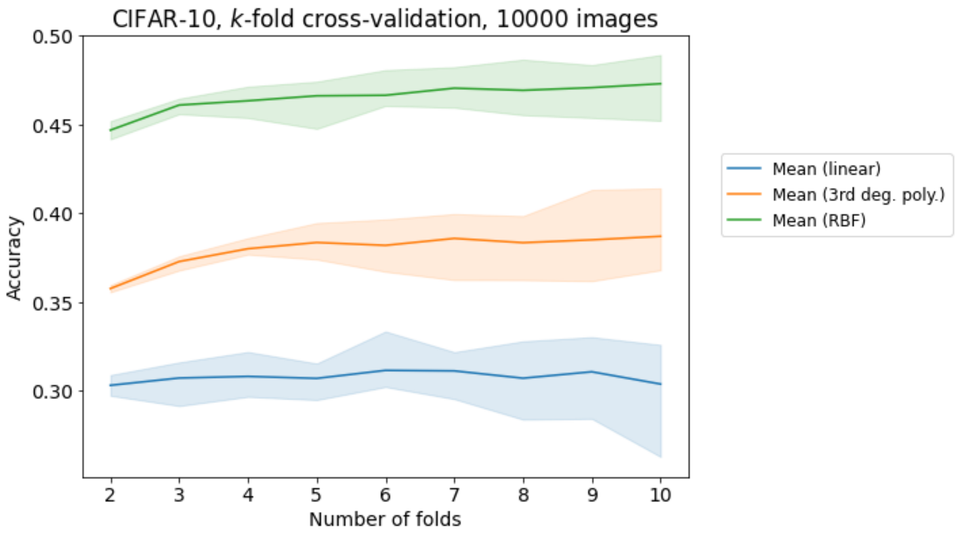


Figure 19: *The accuracy score for CIFAR-10 (10000 images used) SVM with RBF kernel and k-fold cross-validation. The regularization parameter is $C = 1$. The shaded areas indicate the range of the cross-validation iterations.*

Fig. 20 is the same as Fig. 16 unless here the CIFAR10 dataset is used. The results for FashionMNIST and CIFAR10 are very similar apart from the fact that the accuracy for the FashionMNIST classification is considerably higher and the accuracy varies less between the different kernels for that dataset. In the case of CIFAR10F, the mean accuracy for the RBF kernel is 0.466 for $C = 1$ and increases slowly for higher values of the regularization parameter until it plateaus at around 0.480. For the 3rd degree polynomial kernel the accuracy plateaus at around $C = 1$ at 0.390, and the linear kernel gives a relatively constant mean accuracy of about 0.307 for $C > 0.75$.

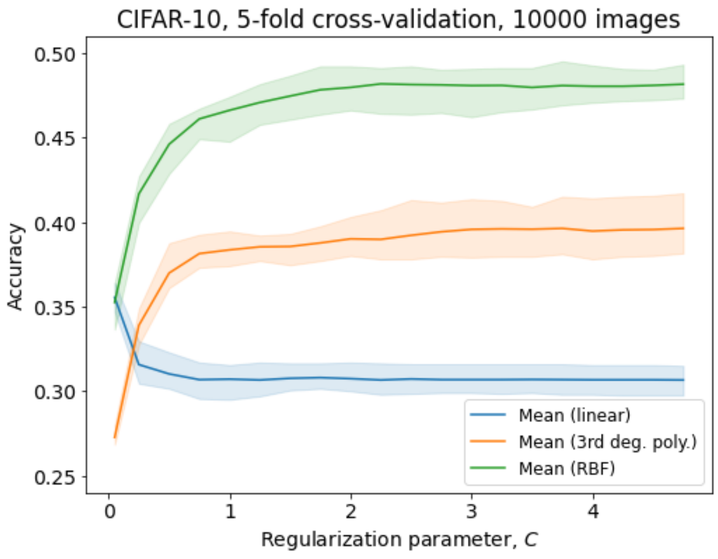


Figure 20: *The accuracy score for the CIFAR-10 dataset (10000 images used) using SVM and 5-fold cross-validation. The regularization parameter C is varied.*

Fig. 21 is the same as Fig. 17 but for the CIFAR-10 dataset. Here, the training data consists of 50000 images and the testing data of 10000. As before, there is an equal amount of images in each class and hence a perfect classification would yield 10% of the total testing data into each class. That would give a value of 0.1 along the diagonal in the confusion matrix. In the case of CIFAR-10, the accuracy for each class ranges between 39% for cats and 69% for ships. These values are much lower than for the FashionMNIST datasaet. The total accuracy of the classification of the testing dataset is 54.4%.

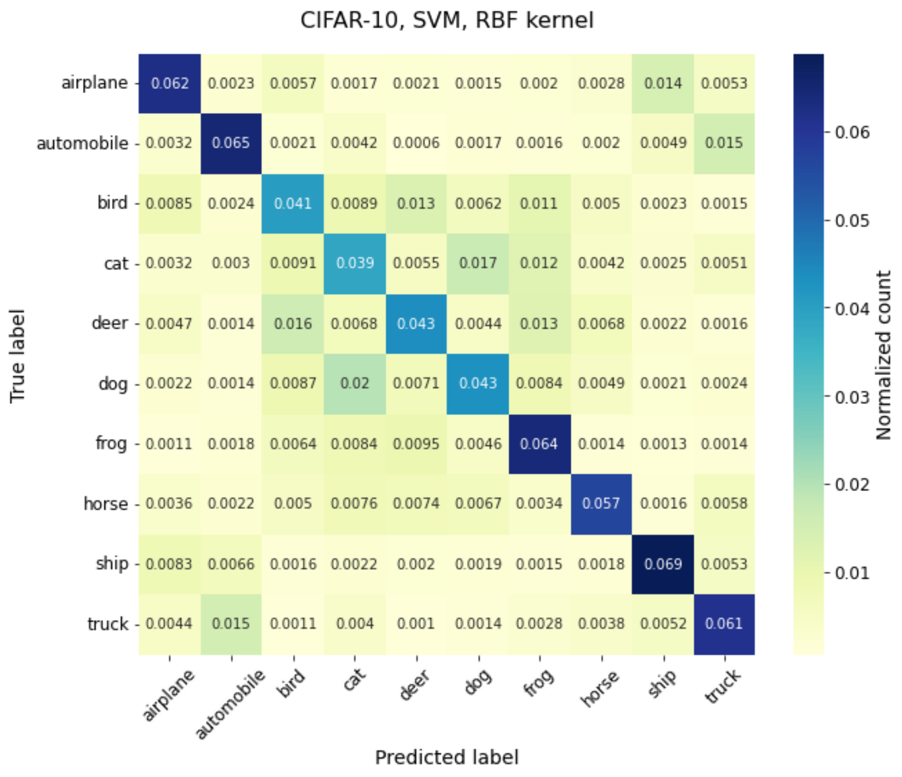


Figure 21: *The confusion matrix for CIFAR-10 using SVM with the RBF kernel and* $C = 1$*. The results have been normalized to the total number of images, here 10000. A perfect classification would results in a value of 0.1 along the diagonal an 0 elsewhere. The model was trained with 50000 images and tested with 10000. Overall accuracy for the testing data is 0.54.*

4.3 Speed comparison

We have seen how different classification methods vary in how accurately they classify items in a dataset. However the accuracy is not the only important metric when it comes to evaluate the model. It is also important to know how computationally expensive it is to run the model, that is how long it takes to run. In Table 1 we compare different models in terms of how long it takes them to fit training data and how accurate this fit is when applying to testing data. For training we used 50000 FashionMNIST images and 10000 images form the same dataset for training. We performed the fitting process five times for each model.

As we saw earlier, CustomCNN1 gives the highest accuracy of the models we used, here with a mean accuracy of 0.914. The SVM classifier follows with a mean accuracy of 0.883 and in last place is CustomCNN2 with mean accuracy of 0.871. However, while CustomCNN1 and SVM both took more than five minutes to fit the FashionMNIST dataset, CustomCNN2 only needed about a minute. It is clear that CNN's can differ quite a bit depending on the model architecture and how tunable parameters are chosen. A well designed CNN could probably both be faster and more accurate than our two custom setups. When compared to CNN's, SVM's do not seem to be able to be as accurate nor as fast when it comes to multi-class classification.

Table 1: *This table shows the time it takes to fit a model to training data and the accuracy of the classification of the testing data. Five iterations over each model are made and the mean, maximum, and minimum values displayed. The models are CustomCNN1 (same setup as described in Fig. 8), CustomCNN2 (same setup as in Fig. 9), and SVM (same setup as in Fig. 17). Here, the models are trained on 60000 images from the FashionMNIST dataset and tested on 10000 images.*

	Time [mm:ss]			Accuracy		
	mean	max	min	mean	max	min
CustomCNN1	05:23	05:35	05:08	0.914	0.922	0.903
CustomCNN2	01:01	01:02	01:00	0.871	0.880	0.853
SVM	05:34	05:42	05:32	0.883	0.883	0.883

4.4 Comparison to existing results

4.4.1 FashionMNIST

Since we devised our own architectures for the CNN's we used in this report, a direct comparison to existing literature is impossible. However, CNN's have been applied extensively to FashionMNIST and seeing which results other model architectures have achieved can help us reflect on the quality of our models. In their paper Greeshma and Sreekumar [24] used several different CNN architectures to classify the images of the FashionMNIST dataset. They managed to get an overall accuracy of 93.00% compared to our 90% achieved with CustomCNN1. We do therefore have some room for improvement. However, it must be noted that while we only used five epochs Greeshma and Sreekumar [24] used 60. As we saw in Fig. 6 increasing the number of epochs really increases the accuracy of the classification. That being said, running a model for 60 epochs takes takes multiple times linger to run than only using five. In another paper, Khanday et al. [25] explored the effects of using differently sized kernels and got a maximum accuracy of 93.68% for the FashionMNIST dataset using a CNN with a 3-by-3 kernel. They find that having larger kernels, for example 5-by-5 or 7-by-7, decrease the accuracy of the classification. However, smaller kernels result in longer computing times. We can see this in our results. CustomCNN1 has a kernel size of 3-by-3 and is both considerably more accurate and considerably slower than CustomCNN2 which has a kernel size of 5-by-5.

The creators of the FashionMIST dataset [16] compiled a benchmark dashboard available at `http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/`. These benchmarks are created with functionalities from Scikit-Learn, in fact the same functionalities we used for our support vector classification. A benchmark using the `SVC` function with the RBF kernel and $C = 1$ - which happens to be the same setup as we used for Fig.

17 - gave an overall accuracy of 88%. This is the average of two independent runs. Unsurprisingly, this is the same accuracy as we got. When looking at other benchmarking experiments, we see that this set-up is among the ones which result in the highest accuracy. Only by increasing the value of the regularization parameter C we can expect to get higher accuracy. This rhymes with what we saw in Fig. 16 where the accuracy slowly increases with a higher value of C in the case of the RBF kernel. The reason we choose not to use a higher value is that the computational time increases for larger C's. However, as we can see from the benchmark list, the increase in computational time is minimal to nonexistent and we should probably have chosen a larger C for our optimal model set-up. Using a polynomial kernel gives similar results as when using the RBF kernel but the degree of the polynomial is not specified in the benchmark list. Hence we cannot compare those results to ours.

4.4.2 CIFAR10

As well as applying their CNN's on FashionMNIST, Khanday et al. [25] also classified the images from the CIFAR10 dataset. In short, they arrived at the same conclusions for those two datasets (see the previous our section about FashionMNIST). Their maximum accuracy for the CIFAR10 images was 73.04% which is considerably higher than our 60% achieved with CustomCNN1. This difference between our model and Khanday et al.'s is much larger for CIFAR10 than FashionMNIST, suggesting that our models are better suited to classify simpler data with fewer parameters.

As mentioned above in the discussion about the classification of the images in the FashionMNIST dataset, our support vector machine model set-up seems to be quite robust. For CIFAR10, our support vector classification with the RBF kernel and $C = 1$ resulted in an overall accuracy of 54.4%. Just for a quick comparison, Cayir et al. [26] got an overall accuracy of only 39.13% when using a support vector classifier. However, they used a linear kernel while we used an RBF one. And as discussed earlier, the RBF kernel generally yields higher accuracy than the linear kernel.

5 Conclusions

In this report we used convolutional neural networks (CNN) and support vector machines (SVM) to classify images. We looked at two datasets, namely FashionMNIST and CIFAR10. Those two are popular when testing classification codes due to their availability and ease of use. FashionMNIST imitates the MNIST dataset but instead of handwritten numbers, FashionMNIST includes images of clothing items. The images are 28-by-28 pixels and consist of a single colour channel, meaning it is computationally cheap to analyse. The CIFAR10 images are slightly larger, 32-by-32 pixels and consist of three colour channels. Both datasets consists of ten separable classes.

For our analysis, we used two different CNN architectures, termed CustomCNN1 and CustomCNN2, and a SVM algorithm. In both cases we varied several external parameters, such as the learning rate of the optimization and number of epochs for the CNN's, and the regularization parameter and the choice of kernel function for the SVM. In order to estimate the quality of our algorithms, we used k-fold cross-validations.

We used two different optimization algorithms with the CNN's. Both are based on gradient descent, one is called Adam [13] and the other stochastic gradient descend (SGD) with momentum. In addition we varied the so called momentum parameter in the SGD method. What both optimizers have in common is that they use information about previous iteration steps to perform the next iteration in the search of optimal parameters. For both CNN architectures we found that the Adam optimizer gave the best results, 90% classification accuracy for CustomCNN1 and 86% for CustomCNN2 in the case of FashionMNIST, and 60% classification accuracy for CustomCNN1 and 52% for CustomCNN2 in the case of CIFAR10. The reason for this difference between datasets is the different size of the images.

In the case of the SVM, the best result were achieved using the radial basis function (RBF) kernel. In that case, the maximum accuracy turned out to be 88% for FashionMNIST and 54% for CIFAR10.

However, the overall accuracy does not tell the whole story. Another important factor is computational efficiency. While CustomCNN1 and SVM take about equally long time to perform a task, CustomCNN2 needs only about a fifth of that time. With those information we can safely say that the CNN's are a more viable option for image classification, at least in our case. Another point which supports this statement is comparison to existing literature. While our SVM set-up seems to be on pair with most other SVM set-ups at this level, our CNN algorithms give lower accuracy than often reported in the literature. This indicates that whereas we do not have much room for improvement within our SVM set-up, the opposite holds for the CNN's.

References

[1] Ela Yildizer et al. “Efficient content-based image retrieval using multiple support vector machines ensemble”. In: *Expert Systems with Applications* 39.3 (2012), pp. 2385–2396.

[2] Giles M Foody and Ajay Mathur. “A relative evaluation of multiclass image classification by support vector machines”. In: *IEEE Transactions on geoscience and remote sensing* 42.6 (2004), pp. 1335–1343.

[3] Waseem Rawat and Zenghui Wang. “Deep convolutional neural networks for image classification: A comprehensive review”. In: *Neural computation* 29.9 (2017), pp. 2352–2449.

[4] Y. LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: `10.1162/neco.1989.1.4.541`.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* `http://www.deeplearningbook.org`. MIT Press, 2016.

[6] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* 2nd. O’Reilly Media, Inc., 2019. ISBN: 1492032646.

[7] Y. T. Zhou and Rama Chellappa. “Computation of optical flow using a neural network”. In: *IEEE 1988 International Conference on Neural Networks* (1988), 71–78 vol.2.

[8] Christopher M Bishop. *Pattern recognition and machine learning.* Springer, 2006. ISBN: 978-0387-31073-2.

[9] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2009. ISBN: 978-0-387-84857-0.

[10] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. “Kernel methods in machine learning”. In: *The annals of statistics* 36.3 (2008), pp. 1171–1220. DOI: `10.1214/009053607000000677`.

[11] Sebastian Ruder. *An overview of gradient descent optimization algorithms.* 2016. arXiv: `1609.04747 [cs.LG]`.

[12] Ning Qian. “On the momentum term in gradient descent learning algorithms”. In: *Neural networks* 12.1 (1999), pp. 145–151. DOI: `10.1016/S0893-6080(98)00116-6`.

[13] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2014. arXiv: `1412.6980 [cs.LG]`.

[14] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[15] Corinna Cortes Yann LeCun and Chris Burges. *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges.* URL: `http://yann.lecun.com/exdb/mnist/` (visited on Apr. 22, 2022).

[16] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.* Aug. 28, 2017. arXiv: `cs.LG/1708.07747 [cs.LG]`.

[17] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).

[18] Marian Tietz et al. *skorch: A scikit-learn compatible neural network library that wraps PyTorch.* July 2017. URL: `https://skorch.readthedocs.io/en/stable/`.

[19] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[20] *1.4. Support Vector Machines - scikit-learn 1.0.2 documentation.* URL: `https://scikit-learn.org/stable/modules/svm.html` (visited on Apr. 22, 2022).

[21] *3.3. Metrics and scoring: quantifying the quality of predictions - scikit-learn 1.0.2 documentation.* URL: `https://scikit-learn.org/stable/modules/model_evaluation.html` (visited on May 5, 2022).

[22] *3.1. Cross-validation: evaluating estimator performance - scikit-learn 1.0.2 documentation.* URL: `https://scikit-learn.org/stable/modules/cross_validation.html` (visited on May 5, 2022).

[23] Trevor Hastie et al. “The entire regularization path for the support vector machine”. In: *Journal of Machine Learning Research* 5.Oct (2004), pp. 1391–1415.

[24] KV Greeshma and K Sreekumar. “Hyperparameter optimization and regularization on fashion-MNIST classification”. In: *International Journal of Recent Technology and Engineering (IJRTE)* 8.2 (2019), pp. 3713–3719.

[25] Owais Mujtaba Khanday, Samad Dadvandipour, and Mohd Aaqib Lone. “Effect of filter sizes on image classification in CNN: a case study on CFIR10 and Fashion-MNIST datasets”. In: *IAES International Journal of Artificial Intelligence* 10.4 (2021), p. 872.

[26] Aykut Çayir, Işil Yenidoğan, and Hasan Dağ. “Feature extraction based on deep learning for some traditional machine learning methods”. In: *2018 3rd International Conference on Computer Science and Engineering (UBMK).* IEEE. 2018, pp. 494–497.