FYS-STK4155 - Project 3

Helene Wold & Tiril A. Gjerstad

December 2021

Abstract

In this paper, we are analyzing two kinds of neural network, and how well they classify different kinds of fruits from images. At first, we are considering the ordinary Neural Network, before moving on to the Convolutional Neural Network. Both methods have been optimized using the Stochastic Gradient Descent. The Convolutional Neural Network was the best method implemented, with a total accuracy of 93.8%. The ordinary Neural Network had a total accuracy of 87.5%.

Contents

1 Introduction **2**

2 Theory **2**
2.1 Activation functions 2
2.1.1 Sigmoid 2
2.1.2 Rectified Linear Unit - ReLU 2
2.1.3 Tanh 2
2.1.4 Exponential Linear Unit - ELU 3
2.1.5 SoftMax 3
2.2 Accuracy of data 3
2.3 Confusion matrix 3
2.4 Optimization Algorithm 3
2.5 Neural Network 4
2.5.1 Feed Forward 4
2.6 Convolutional Neural Network 4
2.7 Overview of parameters 6

3 The data set **6**

4 Methods **6**
4.1 Briefly explained 6
4.2 Preparation of the data 6
4.3 Neural Network 7
4.4 Convolutional Neural Network 7
4.5 Performance of the Neural Networks 8

5 Results & Discussion **8**
5.1 Neural Network 8
5.2 Convolutional Neural Network 10

6 Conclusion **13**
6.1 Future improvements 13

References **14**

1 Introduction

Within machine learning, the CNN is often used as a method for image recognition. In our second project, we implemented a simple neural network using a binary classification problem. [6] For this report, we wish to reuse the optimization method Stochastic Gradient Descent applied on a Neural Network (NN) to do classification on a more complex data set. We also want to implement a Convolutional Neural Network (CNN).

We will consider how these different types of neural networks manages to classify fruits, from a data set ***"Fruit Recognition"***. The data set has been reduced from 15 to 7 categories. The networks are both implemented using TensorFlow's packages [12] with Keras' interface [3]. This will reduce the computational cost and time spent executing our algorithms, as opposed to our own implementation which may turn out inefficient compared to Tensorflow's implementation.

Both the NN and CNN depend on several parameters, such as the learning rate η and the regularization parameter λ. Other parameters are the batch size, activation function for hidden layers, amount of layers and nodes, and for the CNN, amount of filters and filter size. The networks has been analysed using different combinations and magnitudes of the parameters needed, with an ultimate aim of optimizing the performance of our networks. This is done with some trade-off between the performance and computational cost.

This report contains an introductional theory, which takes us through the necessary prior knowledge of the activation functions tested, optimization method, Neural Network and Convolutional Neural Network. Some knowledge from our second project has been reused, with some citations presented using text boxes. Next, the data set is more thoroughly introduced, as well as a description of the methods implemented in our algorithm. Lastly, a presentation and analysis of the network performances is done.

2 Theory

2.1 Activation functions

When implementing Neural Networks, the activation functions are a critical part of the design of the network, as they define how the weighted sum of the input transforms into an output. The chosen activation function used on the hidden layer(s) controls the performance of the training of the network, while the activation function used on the output layer defines the kind of prediction the network will make. The purpose of the activation functions is to introduce non-linearity.

Activation functions

2.1.1 Sigmoid

The Sigmoid Activation function is defined as [5]

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

The Sigmoidal Unit helps by saturating to a high value at very positive z-values, and a rather small value at negative z-values. Sigmoid is sensitive when z is close to 0. [9]
The Sigmoid activation function returns values between 0 and 1, which makes it useful when predicting probabilities in binary classification.

2.1.2 Rectified Linear Unit - ReLU

The ReLU activation function is a variation of the Sigmoid activation function, given by [5]

$$\sigma(z) = max(0, z) \tag{2}$$

The output of the ReLU is then in the range $0 \leq \sigma(z) \leq \infty$, avoiding saturation except with negative inner product between the weights and the input. This makes the gradient vanish, and stops the nodes from learning. The ReLU may in general be used when hidden layers are implemented, as it is rather quick to calculate.

For this project, networks created by TensorFlow [12] will be used. The TensorFlow package has several new activation functions implemented, which simplifies the testing of new activation functions such as *tanh* and *ELU*.

2.1.3 Tanh

The tanh activation function is, as expected, applying the tanh to the input.

$$\sigma(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3}$$

The activation function is hyperbolic, and is quite similar to the Sigmoid activation function. The tanh activation function does however return $\sigma(z)$ in the range $(-1, 1)$, as opposed to $(0, 1)$ for the Sigmoid activation function, and is zero-centered. A zero-centered activation function simplifies the modelling of strongly positive and/or negative inputs. This makes the tanh activation function a slightly better activation function than the Sigmoid activation function, even though this activation function also have a vanishing gradient problem. [5]

2.1.4 Exponential Linear Unit - ELU

The ELU activation function with an Identity form for positive inputs, and smooths out the negative input, slowly making the output equal to $-\alpha$. The ELU is defined as [7]

$$\sigma(z) = \begin{cases} z & z > 0, \\ \alpha(e^z - 1) & z \leq 0, \alpha > 0 \end{cases} \tag{4}$$

As opposed to the ReLU, ELU gives negative outputs, and slowly makes the output become $-\alpha$ when the input is negative. Therefore, the ELU may be a good alternative to the ReLU, as the ELU may return negative output.

2.1.5 SoftMax

The SoftMax activation function calculates the distribution of probabilities of a specific class, over all possible classes. The probabilities is then useful when classifying the specific class. The SoftMax is defined as [11]

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{J} e^{z_j}} \tag{5}$$

The output will always be positive, as e^z never becomes negative, and the output of the SoftMax will sum up to 1. The SoftMax is therefore the chosen output activation, as the output is in the form of probabilities.

2.2 Accuracy of data

Just as in project 2 [6] when categorizing the breast cancer data set, we are categorizing using the accuracy model. Let us revisit our 2^{nd} project again:

Accuracy

When calculating the performance of our neural network in the classification cases, we are using the accuracy function. This function calculates the accuracy after the classification has been done by the neural network. It is defined as [11]

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(\tilde{y}_i = y_i)}{n} \tag{6}$$

2.3 Confusion matrix

When finding the accuracy of a model considering more than two categories, it may be beneficial to see how well each category is classified by the network. This can be done by visualizing the class accuracies using a confusion matrix. The confusion matrix shows how the performance of a classification algorithm.

One example could be the fruits apples and pears with 10 images of each. This could return a confusion matrix with true labels horizontally and predicted labels vertically, with the classes apple at row and column 1 and pear at row and column 2.

$$\begin{bmatrix} 8 & 2 \\ 3 & 7 \end{bmatrix} \tag{7}$$

This shows how 8 apples has been classified correctly, while 3 pears has been misclassified as apples. When applying the SoftMax activation function, the matrix will contain the percentage of the classifications, i.e.

$$\begin{bmatrix} 0.8 & 0.2 \\ 0.3 & 0.7 \end{bmatrix} \tag{8}$$

2.4 Optimization Algorithm

To optimize the performance of the neural networks, an algorithm named Stochastic Gradient Descent will be applied. This method has already been introduced in Project 2. [6]

Gradient Descent

In general, the gradient of any function describes how much the function changes with the respective variable. When a function consists of several variables, the gradient is calculated as a partial derivative. The gradient descent is a way of optimizing our algorithm, mainly used to minimize our cost function.
(...)
While the gradient descent is simple to understand, and has an easy implementation, it does have some limitations. As mentioned, it might find and get stuck in a local minimum of our function, it is sensitive towards our parameters, and is expensive on larger data sets.
(...)
Stochastic Gradient Descent
The Stochastic Gradient Descent (SGD) is a method avoiding some of the imperfections of the normal Gradient Descent (GD). We want to minimize our cost function, which can be written as a sum over n data points:

$$C(\boldsymbol{\beta}) = \sum_{i=1}^{n} c_i(\mathbf{x}_i, \boldsymbol{\beta}). \tag{9}$$

where the gradients can be computed as a sum over i-gradients.

$$\nabla_\beta C(\boldsymbol{\beta}) = \sum_i^n \nabla_\beta c_i(\mathbf{x}_i, \boldsymbol{\beta}). \tag{10}$$

only taking the gradient of a subset of the data, called minibatches. We therefore randomly select minibatches from the data set in each iteration, before computing the gradient for this batch only. This is a less expensive operation than the GD, because the GD method calculates the gradient of the entire data set for every epoch.

More thorough information about minibatches and epochs can be found in project 2.

2.5 Neural Network

As introduced in our report from Project 2 [6], an artificial neural network is put together by a set of layers, built up by a set amount of neurons. One node takes at least one input value, calculates an activation of the node, and its weight and bias. The output of the Neural Network is the predicted values of the data. Neural Networks learns to perform tasks through the process of taking examples, without any specific rule in general. [10] [6] The methods used in the neural network is the Feed Forward and the Back Propagation algorithm.

2.5.1 Feed Forward

FFNN

Throughout the Feed Forward Neural Network (FFNN), the information goes as expected from the name: *forward* throughout the layers of the network.
(...)
The general idea of a FFNN is that the input of our layer is the output from the previous layer.

The output of an FFNN is a result of the chosen activation function $\boldsymbol{\sigma}$ introduced throughout Section 2.1,

$$\mathbf{y} = \boldsymbol{\sigma}\left(\sum_{i=1}^{n} \mathbf{w}_i \mathbf{x}_i + \mathbf{b}_i\right) = \boldsymbol{\sigma}(z) \tag{11}$$

For each node in the l^{th} layer, it is possible to calculate the output $\mathbf{y}_i^l$ through [10]

$$\mathbf{y}_i^l = \boldsymbol{\sigma}^l\left(\sum_{j=1}^{N_{l-1}} \mathbf{w}_{ij}^l \mathbf{y}_j^{l-1} + \mathbf{b}_i^l\right) = \boldsymbol{\sigma}^l(z_i^l) \tag{12}$$

2.6 Convolutional Neural Network

The Convolutional Neural Network, CNN for short, is a method much used to recognise and categorize images. The CNN is rather similar to the ordinary Neural Network, both are made up by nodes with weights and biases that are to be updated. The CNN may, just as NN, learn using different techniques, i.e. the gradient descent which will be applied in this project.

The CNN is specialized at grid-structured data sets, such as a digital image. Since the CNN is specialized, the neural network can have certain properties written into its architecture, reducing the amount of parameters. The CNN takes multi-dimensional arrays, and discretely convolutes them.

An ordinary neural network is an affine transformation, i.e. the output is the vector input multiplied with a matrix of weights. This can be applied to all types of inputs. However, when wanting to apply this to images, they have to be flattened first. This is to flatten the multi-dimensional data set to a vector representation. However, this is not necessary when using a convolutional neural network. The properties of multi-dimensional arrays, such as images, are all preserved when applying discrete convolution, a linear transformation.

As mentioned, the CNN has a more adapted architecture for images, and handles the dimensions of images. This is because of the layer architecture, where the neurons are typically arranged with the dimensions *width*×*height*×*depth*. An image has a general form of *pixel height*×*pixel width*, and depending on the color scheme in the image, a third dimension with typically an RGB value.

A CNN implements sparse interactions, meaning that the main idea of CNNs is to activate a kernel, also called filter, which contains weights. The kernel used in CNNs are smaller than the input, and by convolution, the most important elements amongst thousands of pixels can be stored using just a fraction of the original pixel amount. The kernel is used as a sliding window above the image, performing a convolution inside the window. The step sizes of the kernel when it slides is called stride for CNNs. Typical sizes of kernels are rather small, e.g. 3×3 or 5×5. One example of this is shown in Figure 1, gathered from *Deep Learning* [8], showing how a convolution when using a kernel with a width 3 connects the layers.

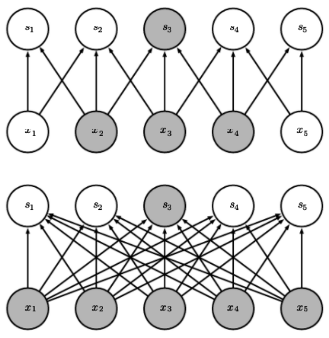


Figure 9.3: Sparse connectivity, viewed from above. We highlight one output unit, s_3, and highlight the input units in $\boldsymbol{x}$ that affect this unit. These units are known as the **receptive field** of s_3. *(Top)*When $\boldsymbol{s}$ is formed by convolution with a kernel of width 3, only three inputs affect s_3. *(Bottom)*When $\boldsymbol{s}$ is formed by matrix multiplication, connectivity is no longer sparse, so all the inputs affect s_3.

Figure 1: Convolution when using a kernel with a width of 3, showing sparse interaction. [8]

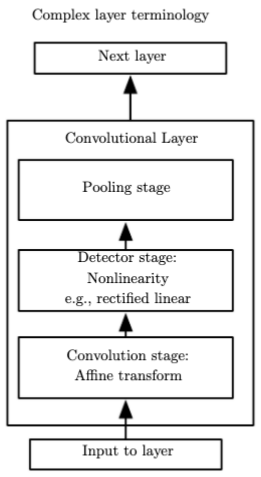


Figure 2: A general built up of hidden layers, gathered from *Deep Learning* [8]

The CNN also applies parameter sharing, i.e. applying the same parameters throughout all input. When computing a feature at one point in an image, this should be the same feature we want to compute at a different point. In general, to get an output, the weights applied throughout the network are the same. This reduces the storage necessary, as the CNN does not use a matrix element only once before throwing it away, therefore reducing the amount of weights necessary in general.[13] The parameter sharing also implies that if input is altered in a particular way, the output will also be altered this way. This is a property named *equivariant representation.*

The CNN is built using a classic neural network architecture. The hidden layer does however consist of convolution, an activation layer and pooling, as many times wanted. The classification in the network happens with a fully connected layer and the SoftMax activation function, and returns a class per image. A general hidden layer architecture is shown in Figure 2.

In the convolutional layer, the input is convoluted using a chosen amount of filters. The output from this layer is on the form $x_{pixels} \times y_{pixels} \times amount\ of\ filters$. This is where the sliding window-technique is applied, i.e. moving the kernel "window" along a pixel axis with a chosen stride. The stride is as mentioned earlier the amount of pixels the kernel is moved. A stride of 1 will move the kernel to the neighbouring pixel, and a stride of 5 will skip 4 pixels and move the kernel to the fifth pixel from its last position.

Next, we have the activation layer, which inflicts a transformation to the output at each neuron after the convolution. The transformation is a chosen activation function, usually ReLU. ReLU takes the output from a neuron, and maps it to it's highest possible value.

After this, the pooling stage is applied. Pooling reduces the dimensionality of the featured map, by dividing the output from the two earlier layers into several pieces of neurons, before picking out our chosen values. There exist many types of pooling, but the most popular pooling choice is the max pooling method, which chooses the maximum values within the piece of neurons. This does as mentioned reduce the dimensionality, and reduces the amount of calculations and weights.

These three operations is considered as one hidden layer, and can be repeated as many times as wanted. When the hidden layers have been run through, the data is flattened. This is to reduce the dimensionality of the pooling layer from a matrix form to a vector form, which prepares the data to be used as input to the fully connected layer. [4] The fully connected layer is, in short, just like a normal, artificial neural network, which is dependent on vector-input. Then, the output activation function is applied, before the CNN returns the classified input. [13]

2.7 Overview of parameters

Parameter	Description	Used in
η (eta)	Learning rate	NN&CNN
λ (lambda)	Regularization parameter	NN&CNN
bsize	Size of mini batches	NN&CNN
n_hidden	A list describing the amount of hidden layers, containing the number of nodes per hidden layer	NN
n_filters	Amount of filters	CNN
n_neurons	Number of neurons connected	CNN
Receptive fields	Width of the filters	CNN

Table 1: A table listing an overview of the parameters used for the different networks.

3 The data set

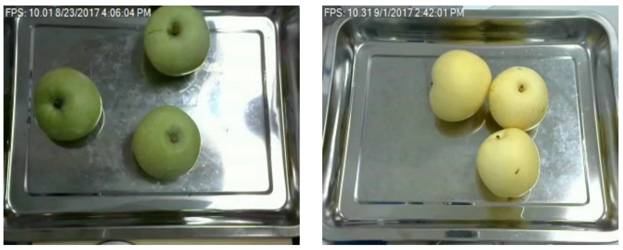
We are using the data set ***"Fruit Recognition"***, and do classification using NN and CNN. The data set is found at this link:
`www.kaggle.com/chrisfilo/fruit-recognition`

This data set is 8GB big, containing 44 406 images distributed between 15 categories of fruits divided into folders. Each image has 320 × 258 pixels. Some fruit folders has divided the fruit into several sub folders. One example is the apple, which has several colors. One sub folder then contains all the red apples, the next contains the green apples, and so on.

The images have been taken over a period of 6 months in a LAB environment. They have been taken while creating different scenarios, such as the light coming from different angles, the images being taken from different angles, and a different amount of the fruit in question. This is to make the data set more robust, as the images are more realistic.

Since the data set consists of so many images and categories, we have chosen to reduce the amount of data in our project, so that the data set is more manageable and fit for our project. The data set has then been reduced to the categories apple, banana, kiwi, mango, orange, pear and plum. We have chosen to not classify e.g. red versus green apples, and just look at apples in general. All of these fruits are known to us, and after some inspection of some images, we noticed visual similarity between some fruits. One example is shown below, a green apple and a pear, who share the same shape and color-scheme.

The piece of the data set used for this report can be found at ***the Google Drive of helenewo@uio.no***.


FPS: 10.01 8/23/2017 4:06:04 PM
FPS: 10.31 9/1/2017 2:42:01 PM

(a) Three green apples from our data set **(b)** Three pears from our data set

Figure 3: An example of how some fruits has similar features, such as shape and color-scheme, gathered from the data set.

If any trouble with accessing the images, please send an email to the listed email address.

Some more actions have been done to reduce the amount of data. We downscale the images from 320×258 pixels to 75×75 pixels, and choose a random set of images from each category for testing and training our neural networks. Each category has somewhere between 2000 and 10 000 images. Our tests has all been executed with 200 images from each category, making each category represented properly. These 200 images is randomly picked, to avoid the bias from choosing 200 images e.g. in a row, as they can in some cases all represent the same physical scenario.

4 Methods

The methods we use are all to be found in our ***GitHub repository***, found at this link:
`https://github.uio.no/helenewo/Project3`

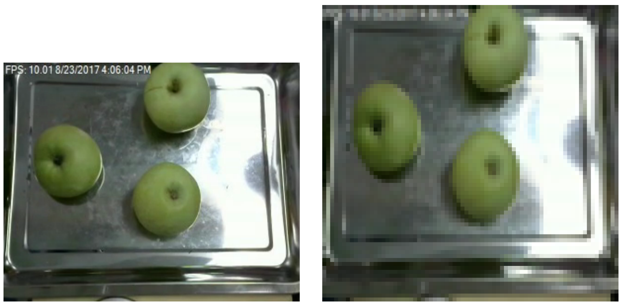
4.1 Briefly explained

We are applying two deep learning methods on our data set *Fruit Recognition.* The methods are Neural Network and Convolutional Neural Network, from the TensorFlow's packages [12] with Keras' interface [3]. The ultimate goal is to make our networks perform as well as as possible by adjusting and tweaking the different parameters for each network, so the networks manages to recognise and correctly categorise different fruits. We have downscaled the entire data set, so that it is more manageable for our computers. We have made several tests to find the best parameters for each neural network.

4.2 Preparation of the data

Reading the data: As earlier mentioned, our data set contains a great amount of data, and we reduce the data set to lower the computational cost of using the entire set. We have also added the possibility to only use a set amount of images from each category when running the code. This happens in the file *data_prep.py* , which contains the class *DataPrep*. This class takes a list containing the paths and labels for the categories as the input, as well as an optional choice of the amount of images from each class. The initial property of the class is to read the image into a matrix containing the pixel from each image, as well as the RGB value of each pixel.

The class also contains a method to downsize the pixel size of the image to a desired amount, where the default amount we have used is 75×75 pixels. This downsizing is important considering the amount of time spent when testing our networks, but there is also a trade-off with the information loss. It is therefore important not to downsize too much, such as downsizing to 15×15 pixels. The downsizing to 75×75 pixels reduces an image as shown in Figure 4. It is still visually possible to make an assumption of which fruit we are looking at. It makes it however harder to i.e. separate the fruit shown in Figure 3. With enough images, it is easier to distinguish the features of each fruit.



(a) Three green apples from our data set **(b)** The downsized Apple image.

Figure 4: An example of how a downsized image looks.

The class also has a flatten-function, useful when applying the NN. This is so that the data is less complex and possible to use in the ordinary NN. The CNN does however expect images, and it is therefore no need to flatten this.

Scaling the input data: When the images has been read and possibly downsized, it is necessary to scale them. This is to reduce the great variance of the pixel values in the images, and scale the values to less than 1 in variance. Note that we do not scale the matrix containing the expected output. This is because we created this as a one-hot vector, which should sum to one at each row.

The function read_fruit located in the file *FruitReads.py* reads the data with our *PrepData* class. The read_fruit function reads the chosen categories and creates the paths and labels, applies the *PrepData*-class, splits the data into test and train, and scales the image-matrix.

We have applied two different ways of scaling the data, by using SciKit-Learn's StandardScaler [2], and the method of dividing all values in the images with 255. Both scaling methods was just as stable, however the most efficient method throughout our tests was the StandardScaler. We have therefore chosen to use the StandardScaler when testing.

Splitting the data into test and train: Just as in earlier projects, we want to train and test our models using different sets of data. This is to avoid problems such as overfitting our model to the data set, making it only useful for that particular data set. By applying SciKit-Learn's's method *test_train_split* [1], we split our data set into train and test, with a ratio of $80 : 20(train : test)$. Since our data set still has more data available, we can test our model much more by using other pieces of the data set.

4.3 Neural Network

We have implemented the Neural Network using TensorFlow's packages [12] with Keras' interface [3]. This is implemented in the file *NN_keras.py*, and is structured as a class named *NN_Keras*. All parameters we might find useful to adapt and tweak is taken as input parameters to an object of the class. There are several parameters to be tweaked through several tests, namely η, λ, the batch size, amount of hidden layers, and the activation functions for the hidden layers. These have all been presented in Table 1 presented in Section 2.7.

We have also applied and tested the implementation of our own Neural Network created for project 2. This can be found in the file *NeuralNetwork.py*, with the activation functions and cost functions implemented in the files *activation.py* and *costfuncs.py*.

The Stochastic Gradient Descent is used as the optimization algorithm with both the Keras implementation and our own. Our own Neural Network includes the Feed Forward method and the Back Propagation method. All tests applied on the Neural Network can be found in the file *test_NN.py*.

4.4 Convolutional Neural Network

The second method we are testing is the Convolution Neural Network. As mentioned in Section 2.6, this is a good method when using images.

The Network is implemented using TensorFlow's packages [12] with Keras' interface [3], just as with the Neural Network. The CNN is implemented as a class named *CNN_Keras*, and is to be found in the file *CNN_keras.py*. The class takes the CNN's parameters as input, for easy testing. Just as with the Neural Network, we adapt the parameters η, λ and activation function for the hidden layers. However, for the CNN we also take a look at the amount of filters, number of neurons connected, and the receptive fields. We began with a simple model, consisting of only one hidden layer, one flattening layer, and two dense layers (Table 1). The Stochastic Gradient Descent is still used as an optimization algorithm.

All tests made for the CNN is implemented in the file *test_CNN.py*.

4.5 Performance of the Neural Networks

As mentioned in Section 2.2, we are measuring the accuracy of our network to decide how well they perform, and how precise they classify the fruits. The loss function in the networks is Cross Entropy [6], and the SoftMax is used as the activation function on the output layer. To visualise our results, we have used different graphs and heatmaps to show the accuracy score with different parameter combinations. These will be introduced shortly. Since these results only show how well the networks performs in total, as opposed to the accuracy of each category, we are calculating a confusion matrix (Section 2.3) to visualise how well the network classifies the fruits.

The different tests have as earlier mentioned been gathered in the files *test_NN.py* and *test_CNN.py*. The approach in both cases is to begin with a simple model, i.e. one hidden layer, before gradually making it more complex if it improves the performances significantly. The goal is to find a simple model while maintaining a good accuracy for all categories. The most important parameter is shown to be the learning rate η, as this is the parameter the models are the most sensitive towards. We therefore want to make this our first parameter to be decided. Our first priority is however to decide the best magnitudes of the parameters, which is done by less complicated tests back and forth between our tests. Then we set the best value for the learning rate, before repeating the process with the regularization parameter λ. We also test how different batch sizes, activation functions and different combinations of hidden layers affect the models, as well as the amount of images used as input.

5 Results & Discussion

5.1 Neural Network

Starting off with only one hidden layer with 200 nodes, we initiating our test with a rather simple network. The chosen activation function for the hidden layer is set to be ReLU, and SoftMax as the output activation function. We begin testing using a batch size of 4, and $\lambda = 0$, and run our tests using downsized images of 75×75 pixels and 200 images. To keep the network simple, we are initially only using one hidden layer with 100 nodes.

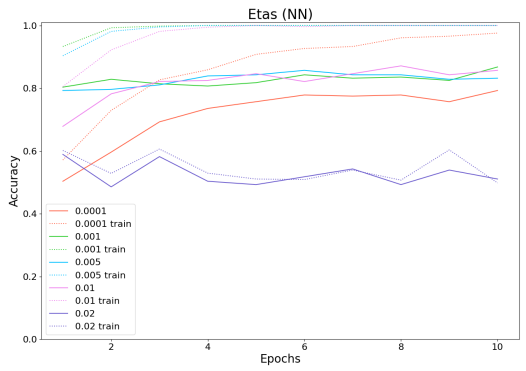


Figure 5: The development of the accuracy of our Neural Network with varying η. Set parameters: batch size 4, $\lambda = 0$.

Our first test is a test of the learning rate η, which is a sensitive parameter as shown in Figure 5. The accuracy is varying between about 50% and 90% with the testing data. Figure 5 shows that a learning rate $\eta = [0.001, 0.01]$ is the most ideal interval, as this is where the model performs the best within the amount of epochs we have considered. A larger learning rate of $\eta = 0.02$ gives a unstable performance and low accuracy of the network compared to the smaller learning rates.

A smaller learning rate of $\eta < 0.001$ makes the network learn slower, meaning a bigger amount of epochs is necessary for the performance to possibly be as good as $\eta = [0.001, 0.01]$. From Figure 5 it is difficult to properly differentiate between the three values tested for η in the interval $[0.001, 0.01]$. We therefore move on to the next test, i.e. taking a look at η versus λ.

From Figure 6 it is noticeable how a small learning rate benefits from a bigger value of the regularization parameter λ. As long as both η and λ isn't too large, the performance of our network is quite similar and good, and the intervals $\eta = [0.001, 0.01]$ and $\lambda = [0, 0.05]$ returns the in general best accuracies. We want to avoid peak accuracies where a slight change of a parameter can reduce the performance of

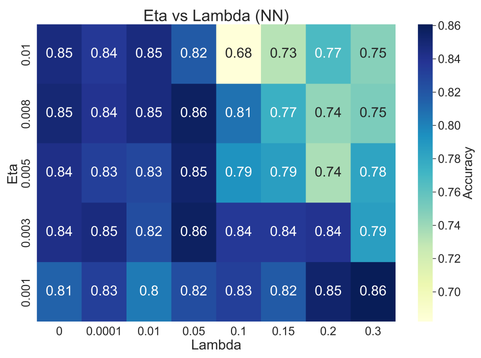


Figure 6: Accuracy of our Neural Network when considering a varying η versus λ. et parameters: batch size 4, 6 epochs.

the model, as a stable and good model is better than the best, however maybe unstable, model. We therefore continue our tests using $\eta = 0.01$ and $\lambda = 0.0001$.

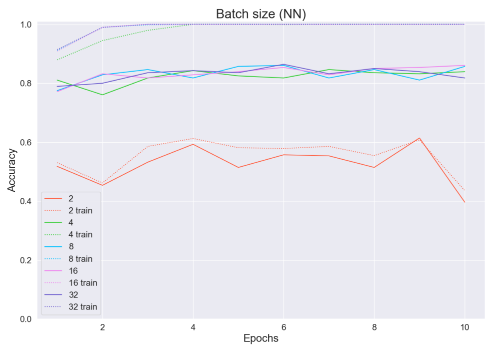


Figure 7: Accuracy of our Neural Network when considering a varying amount of batches versus epochs. Set parameters: $\eta = 0.01$, $\lambda = 0.0001$.

Now that the most sensitive parameters has been decided, we move on to a quick study of the batch size. Figure 7 shows us how a batch size of 2 is too small, reducing the performance of the model drastically compared to other values tested. Networks using batch sizes of 4, 8, 16 and 32 performs rather similar, with a testing accuracy of more than 80% after a couple of epochs. The network does however need a few more epochs to perform as well as the bigger batch sizes when looking at the training data. It would seem fit to then use a batch size of 8 or higher when testing further, but by repeating a few simple tests it was noticeable how a batch size of 4 actually performs better. We therefore do not deem the slightly slower training when using 4 batches significant, and will continue using a batch size of 4 through our further tests.

Our next move is to test some different activation functions introduced in Section 2.1. So far we have used the ReLU activation function at the hidden layer, which means our parameters has been chosen to fit this activation function the best. This does not necessarily mean that the ReLU is definitely the best, but it should give one of the best performing networks. By looking at the accuracy from this test, shown in Figure 8, the differences between the activation functions using the different parameters chosen so far is quite similar. The ReLU do need a few more epochs to be comparable to the other activation functions, but since the ReLU is a rather simple calculating operation while still performing well, we have chosen to keep using this.

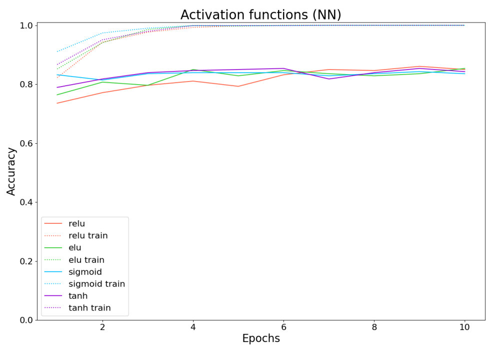


Figure 8: The development of the accuracy of our NN when considering different activation functions. Set parameters: batch size 4,$\eta = 0.01$, $\lambda = 0.0001$.

A more complex and deep Neural Network could possibly have a bigger difference between the networks, as i.e. the Sigmoid activation function performs worse when using deeper neural networks. Let us therefore look at the performance of the network when using ReLU as the activation function with different combinations of hidden layers amount of neurons at each hidden layer. Figure 9 indicates how a more complex neural network than used so far is beneficial. On the other hand, a large increase will result in a greater computational cost, which means that we want to avoid increasing too much without significantly better results. Figure 9 shows how two hidden layers using 300 nodes on each layer seems beneficial, as this is a good compromise.

One bottleneck we came upon initially with our tests is the amount of pictures used in our data set. Figure 10 illustrates how the amount of images affects the accuracy, by finding the accuracy when using different amounts of images between 100 and 1000 images from each category in the data set. The performance of our networks increases slightly when increasing the amount of images. We would however not consider this as a significant increase, as the accuracy with the least amount of images (100 from each category) has an accuracy of 78%, while the peak per-

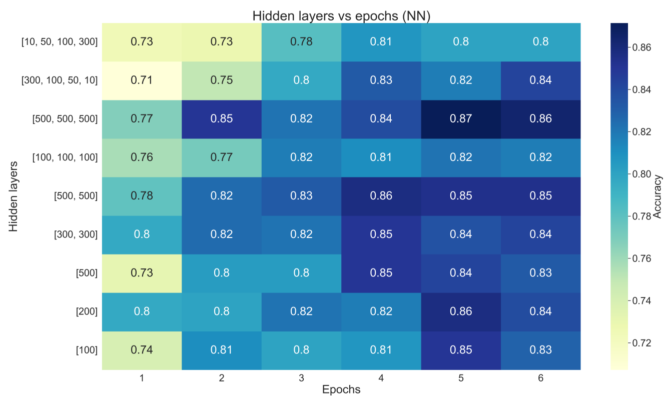


Figure 9: Accuracy of our Neural Network when considering a varying amount of hidden layers and amount of nodes, versus epochs. Set parameters: $\eta = 0.01$, $\lambda = 0.0001$.

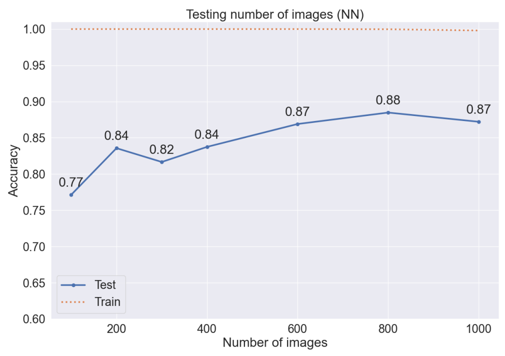


Figure 10: The development of the accuracy of our NN when increasing the amount of images chosen for the data set (ratio $80:20(train:test)$ either way). Set parameters: batch size 8,$\eta = 0.01$, $\lambda = 0.0001$, layer architecture=[300,300].

formance of 88% is when using 800 images from each category.

We do however run one last test using 800 images from each category using the parameters chosen earlier, and take a look at the confusion matrix of the network shown in Figure 11. The final and total accuracy of this network is 87.5%. The correctly classified fruit ranges from 81% to 100%, the plum being the easiest to classify. This is probably a result of the plum being the only properly dark fruit in the data set. The most misclassified fruit is the mango, which is sometimes mistaken as every other fruit in the data set, except as a plum. When taking a look through the images of the mango, we noticed how some mangoes actually has similar shapes and colors as e.g. bananas, apples or oranges.

The different fruits has, if misclassified, in general been misclassified less than 5% as other fruits. This

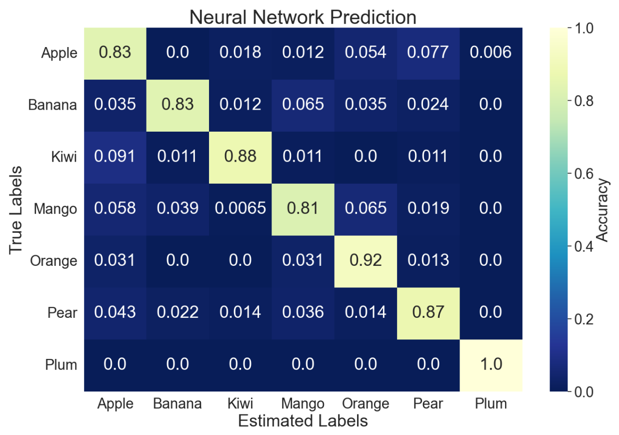


Figure 11: The confusion matrix between the estimated and true labeling of the images when looking at 800 images. Total accuracy of the model is 87.5%. Set parameters: with $\eta = 0.01$, $\lambda = 0.0001$, batch size 4, 6 epochs, layer architecture=[300,300].

observation is however with a few exceptions, such as 9% of true kiwis has been classified as apples, or 7% of true apples who has been classified as pears.

It is worth mentioning that the confusion matrix in Figure 11 is the result of one of our runs, as we randomly pick different images each run. When testing, we noticed that the performance of the network varies when considering the confusion matrix, while maintaining a rather high accuracy around 88% in total. In general, the plum is mostly classified 100% correct, while the other fruits are classified correctly with an accuracy somewhere between 80% and 95%.

Since our reruns of this test has mostly been done using few images considering the total amount of images, it is not unexpected that a randomly bad pick of images one run can struggle when classifying e.g. mangoes and bananas, and the next run has problems with the apples versus pears. This variance is because of the big data set containing many different versions of the same fruit, as mentioned in Section 3

5.2 Convolutional Neural Network

We repeat several of the tests we applied for the NN when moving on to the CNN. The first model that is to be tested is rather simple, with initial testing of the parameters η, λ, batch size, activation functions, receptive field, amount of filters, amount of neurons, with a final test with an increased amount of images.

With some quick testing back and forth, we noticed how the batch size does not greatly affect the total accuracy of our model, and we chose a batch size of 8. All tests has been done using the ReLU activation function on the hidden layers, and SoftMax on the output layer. Throughout all tests, we have used receptive field amount of 3, 20 filters and 100 neurons. This will be presented slightly more thorough later on. As mentioned, we do testing with 200 images downsized to 75×75 pixels.

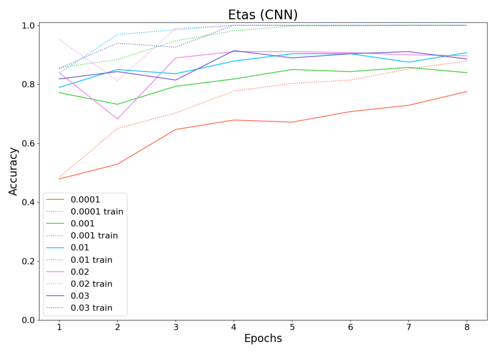
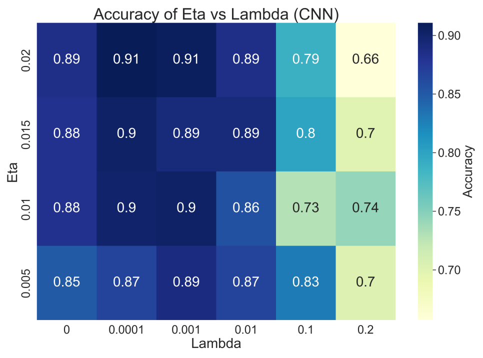


Figure 12: The development of the accuracy of our CNN with varying η. Set parameters: batch size 8, $\lambda = 0$.



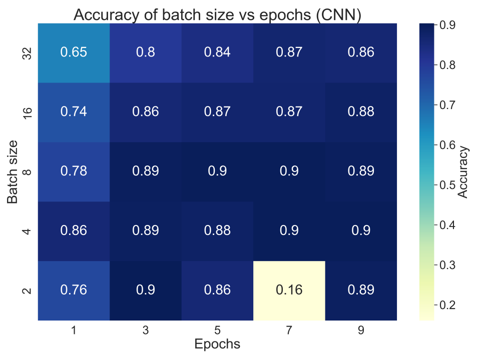
Eta \ Lambda	0	0.0001	0.001	0.01	0.1	0.2
0.02	0.89	0.91	0.91	0.89	0.79	0.66
0.015	0.88	0.9	0.89	0.89	0.8	0.7
0.01	0.88	0.9	0.9	0.86	0.73	0.74
0.005	0.85	0.87	0.89	0.87	0.83	0.7

Figure 13: Accuracy of our CNN when considering a varying η versus λ. Set parameters: batch size 8, 10 epochs.

Figure 12 and Figure 13 shows how the learning rate $\eta = 0.02$ is a good choice. A smaller η restricts how fast our network learns, meaning we have to increase the amount of epochs if we want to see how well the performance is of a model using the smallest learning rate shown in Figure 12. The learning rates $\eta = 0.1, 0.2, 0.3$ gives a good accuracy fast, however a bigger η may give a slightly unstable network as mentioned in project 2. [6] Since $\eta = 0.01$ gives slightly worse accuracy than $\eta = 0.02$, we do further tests with $\eta = 0.02$.

When looking at η versus λ in Figure 13, it is noticeable how a small enough regularisation parameter λ gives a better accuracy of our model. A good value of λ is then between 0.0001 and 0.001, which gives slightly better results than the networks with $\lambda = 0$. We then choose $\lambda = 0.0001$ for further testing.

By looking at Figure 14, it is noticeable how the batch size doesn't have a great affect on the accuracy of the CNN. A bigger batch size lowers the computational cost, as we train our network using more data for each mini batch, and run a smaller amount of batches in general. A bigger minibatch size does not improve the accuracy much either, just as with a smaller minibatch size. We therefore choose a batch size of 8, as this is gives a rather good accuracy.



Batch size \ Epochs	1	3	5	7	9
32	0.65	0.8	0.84	0.87	0.86
16	0.74	0.86	0.87	0.87	0.88
8	0.78	0.89	0.9	0.9	0.89
4	0.86	0.89	0.88	0.9	0.9
2	0.76	0.9	0.86	0.16	0.89

Figure 14: Accuracy of our CNN when considering a varying batch size versus epochs. Set parameters: $\eta = 0.02$, $\lambda = 0.0001$

The accuracy when using a batch size of 2 and 7 epochs can be dismissed, as this most probably is a result of numeric overflow.

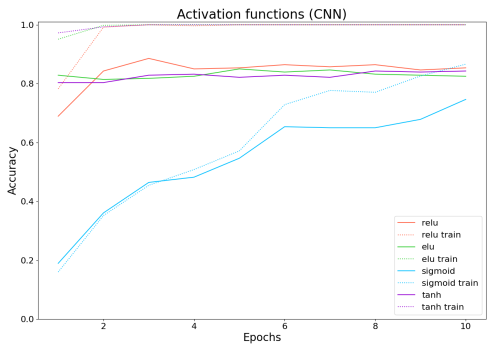


Figure 15: The development of the accuracy of our CNN when considering different activation functions. Set parameters: batch size 8,$\eta = 0.02$, $\lambda = 0.0001$.

Since the ReLU activation function is a stable choice (see Section 2.1.2), we have used this so far. We do find it necessary to evaluate how well the different activation functions performs using our chosen parameters so far. Since the parameters has been set using ReLU, we do expect this to be amongst the top performing activation functions. Either way, the SoftMax activation function has been used as the output activation functions, as mentioned initially in this discussion.

As seen in Figure 15, the ReLU is slightly better than the ELU and tanh activation functions. The simplicity of ReLU is a big reason for keeping the ReLU as our chosen activation functions for future tests. The Sigmoid activation function is, just as in project 2, a slower performer, meaning it would take more epochs for the Sigmoid to perform possibly as well as the other activation functions.

Additional tests has also been done looking at the parameters receptive fields, amount of filters and amount of neurons connected. These results are to be found in our ***GitHub repository***. The best results of these tests was 3 receptive fields, 20 filters and 100 neurons connected. The results gave no indication of drastically improved performance with bigger or smaller values for these parameters.

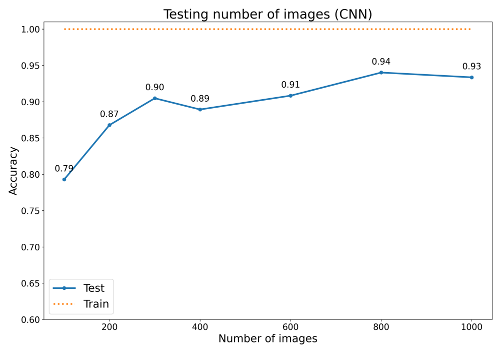


Figure 16: The development of the accuracy of our CNN when increasing the amount of images chosen for the data set (ratio 80 : 20($train$: $test$) either way). Set parameters: batch size 8,$\eta = 0.02$, $\lambda = 0.0001$.

Throughout our testing, we have seen how our CNN trains fast, with an accuracy of 100% after only about 3-5 epochs, and a testing accuracy around 90%. This implies that our network trains well, while might lacking some data to be able to improve the last 10%. We therefore test an increase of the amount of pictures from each category, as shown in Figure 16. This shows us how the increase of images from 200 to e.g. 800 increases the accuracy from 87% to 94%. Since there are so many images showing different scenarios using fruits, our model should have a peak performance when considering the entire data set. This is however a big computation, not fit for the computers used throughout this project.

It is also worth mentioning that the parameters used for Figure 16 is the best parameters chosen when considering 200 images, all downsized to 75×75 pixels. These parameters may not be the best for other situations, so the performance may not reach 100% when taking the entire data set into consideration. Also, the entire data set does not have an equal distribution of images throughout the categories, meaning it may have some trouble classifying the smaller fruit-categories.

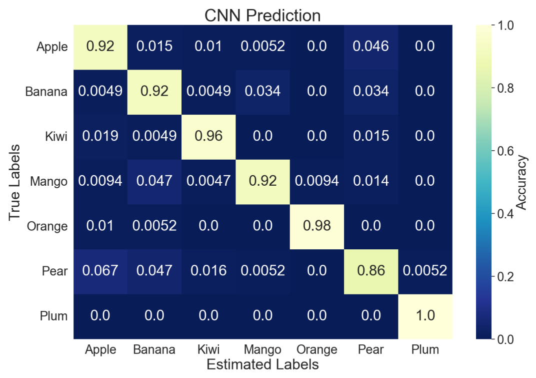


Figure 17: The confusion matrix between the estimated and true labeling of the images when looking at 800 images. Total accuracy of the model is 93.8%. Set parameters: with $\eta = 0.02$, $\lambda = 0.0001$, batch size 8, 6 epochs

At last, we have created a confusion matrix to visualise how correctly our network categorises the fruits using 800 images, shown in Figure 17. The most distinguishable fruits are the pears and the plums, where all plums are correctly classified, while the pears are the most misclassified ones. Most of the wrongly classified pears have been classified as apples by our model. As shown in Figure 3 in Section 3, the pears compared to the green apples are rather similar, making mistakes like this rather simple. It might also be a large amount of green apples amongst the chosen images in the data set, which makes it easier to confuse the two fruits.

6 Conclusion

Throughout our project, we have analyzed how well the Neural Network (NN) and Convolutional Neural Network (CNN) performs when considering a data set consisting of images of different fruits. How does the two networks compare to each other? Let us take a look at the results from our analysis.

NN (Keras)	**CNN (Keras)**
0.875	0.938

We also tested our own Neural Network from our second project. After some testing and execution of the code, we dropped the idea of maybe being able to use our own neural network for this project. Our own neural network performed poorly, and the model wound up with the best performance around 20%.

As the CNN has an architecture specialized on multidimensional arrays, it is expected a better performance with the CNN as for the NN. Throughout testing, we have unintentionally decided on quite similar parameters when comparing the two networks. Both networks performed best when using somewhere between 5 and 10 epochs, the same regularization parameter $\lambda = 0.0001$, and some slight difference in the learning rate $\eta_{NN} = 0.01$ versus $\eta_{CNN} = 0.02$ and batch sizes $bsize_{NN} = 4$ and $bsize_{CNN} = 8$.

When looking at the two confusion matrices from Figure 11 and Figure 17, it is possible to see how the networks have many of the same misclassifications, such as apples versus pears, or mangoes versus bananas. Both networks classified their plums correctly. The misclassifications may originate from how the images used for the data set has been picked – randomly at each run. If we keep the apples in mind, the majority may be green the first run, while mostly red the next run, and so on.

The biggest and most significant difference between the two models is probably how the two networks manages image-structures as input, where the CNN is better prepared for images as introduced in Section 2.6. This is shown through the results, where the CNN performed 6.3% better when comparing the two runs we have included in this report. This has also proven to be the general standard throughout our tests and executions. When classifying images, we would therefore use a Convolutional Neural Network instead of an ordinary Neural Network.

6.1 Future improvements

The last 6% remaining for our model to perform as well as physically possible could be achieved by using the entire data set with all images, and tweak the parameters using the new amount of data. This could lead to some misclassifications, as there are ”only” 3000 images of bananas, as opposed to about 10 000 images of the apples. These are not necessarily the most similar fruits when looking at them with our own eyes, but it decreases how familiar the network is with bananas versus with apples.

One could also simplify the input could be to convert the images to gray scale. This reduces the complexity of images from an RGB scale to a simple gray scale of a set amount of gray levels. This was chosen not to be done in this project.

For further improvement, we could also spend more time perfecting the amount of hidden layers and nodes for each layer with the CNN, in combination with the amount of filters at each hidden layer. An implementation of different optimization algorithms may also be beneficial, as we only implemented the Stochastic Gradient Descent.

References

[1] SciKit-learn developers (BSD License). *sklearn.model_selection.train_test_split*. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. Accessed: 17-12-2021. 2021.

[2] SciKit-learn developers (BSD License). *sklearn.preprocessing.StandardScaler*. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.htmll. Accessed: 17-12-2021. 2021.

[3] François Chollet et al. *Keras*. https://keras.io. 2015.

[4] *Convolutional Neural Networks (CNN): Step 3 - Flattening*. https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening. Accessed: 12-12-2021. 2018.

[5] Snehal Gharat. *What, Why and Which?? Activation Functions*. https://medium.com/@snaily16/what-why-and-which-activation-functions-b2bf748c0441. Accessed: 17-11-2021. 2019.

[6] Tiril A. Gjerstad and Helene Wold. *Project 2*. 2021.

[7] ML Glossary. *Activation functions*. https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html. Accessed: 06-12-2021. 2017.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. https://www.deeplearningbook.org/contents/convnets.html. MIT Press, 2016.

[9] Tushar Gupta. *Deep Learning: Feedforward Neural Network*. https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7. Accessed: 12-11-2021. 2017.

[10] Morten Hjorth-Jensen. *Week 40: From Stochastic Gradient Descent to Neural networks*. https://compphysics.github.io/MachineLearning/doc/pub/week40/html/week40.html. Accessed: 18-10-2021. 2021.

[11] Morten Hjorth-Jensen. *Week 41: Constructing a Neural Network code, Tensor flow and start Convolutional Neural Networks*. https://compphysics.github.io/MachineLearning/doc/pub/week41/html/week41.html. Accessed: 07-11-2021. 2021.

[12] Martìn Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[13] Mayank Mishra. *Convolutional Neural Networks, Explained*. https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939. Accessed: 08-12-2021. 2020.