An Investigation into Image Classification with Neural Networks and Convolutional Neural Networks

Muhammad Ali Bin Md Nazeri — Don Philip Bullecer

Department of Physics, University of Oslo, Norway

https://github.com/dondondooooon/DONFYS-STK3155

December 19, 2022

Contents

Abstract

This article explored the use of neural networks and convolutional neural networks for classification on the CIFAR-10 data set (Krizhevsky, Hinton, et al., 2009), consisting of 60,000 32x32 colour images in 10 classes, with 6,000 images per class. Using the Keras library, we developed a feed-forward neural network (FFNN) as a benchmark for modelling the data set and further tested our FFNN code with 3 different activation functions: Sigmoid, Rectified Linear Unit (RELU), and SoftMax. We used the Keras Tuner library (O'Malley et al., 2019) to acquire accuracy scores and kept close track of certain parameters from our FFNN, such as model architecture, learning rate, and choice of regularizer, before comparing it to a convolutional neural network (CNN) also built with the help of Keras's preset methods. For the comparison between our FFNN and the CNN, we discovered that the CNN had a better overall performance with an accuracy of (76.0% vs 16.8%) and a cross-entropy loss score of (0.776 vs 2.07). Additionally, we discovered that Sigmoid was the activation function of choice for the FFNN, and ReLU for the CNN (with the exception of the output layer, which we set to SoftMax due to the multi-class nature of the dataset).

1 Introduction

In the field of deep learning, the performance of neural networks has been extensively studied for various tasks such as image recognition, language translation, anomaly detection, and image classification. Neural networks are computational models that are inspired by the structure and function of the brain, where we have an input and an output layer, and several hidden layers, all with various numbers of neurons or nodes that connect to other nodes in other layers with their respective weights and biases.

In this investigation, we will focus on the problem of image classification and take a closer look at the performance of the Feed-Forward Neural Network(FFNN) and Convolutional Neural Network (CNN) on the CIFAR-10 dataset. CNNs are a type of neural network that utilizes the idea of convolution in mathematics to filter out and or reduce the dimensionality of a given problem. That is why CNNs specifically handles images with high resolutions (dense matrix) exceptionally well, and they have been shown to achieve outstanding performance on many image classification tasks. We will be using Keras to build both our FFNN and CNN. Our investigation focuses on the task of classifying various images in the CIFAR-10 data set into different categories. We will compare some performance metrics such as the cost and loss in the form of accuracy and cross-entropy values for both the FFNN and CNN, and investigate the impact of different hyperparameter values, network architectures and training strategies on the accuracy of the models. We also discuss the potential applications and limitations of these techniques for image classification and other related tasks, and our results will provide insights into the strengths and weaknesses of these algorithms, and can be used to guide the choice of algorithm for image classification tasks.

This report is separated into 5 sections: Methods Theory, Implementation, Results & Discussion, Conclusion and Appendix. The first section gives a brief introduction to the methods and theories that were employed through the investigation and the mathematics and underlying ideas behind them. Implementation covers the code we used in the investigation, and how we implemented the methods and theory into programs with a reproducible output. Important snippets of the code will be shown as figures in this section, and extra, not-so-important figures can be found in the appendix. The results and discussion section covers the use of neural networks for the classification of the CIFAR-10 data set. Furthermore, a number of selected plots produced from our models will be placed in this section, where we analyze the figures and examine them carefully to see what we can learn from them. Finally, the conclusion section will be where we come to a decision regarding whether FFNN or CNN performs the best in classifying the CIFAR-10 data set.

The extra exercise concerning the Bias-Variance Trade-off Analysis will be covered directly after this report, with the following sections: Introduction, Methods & Theory, Results, and Conclusion. These sections will follow a structure similar to the outline above, with some differences.

2 Method & Theory

This section explores the background information and relevant equations for the methods and theories we employed over the course of the investigation.

2.1 Neural Network

2.1.1 Feed-Forward Neural Networks (FFNNs)

Feed-Forward Neural Networks are neural networks where information only moves in a single direction, from the input layer through a number of layers where the input is sorted before it finally arrives at the output layer.

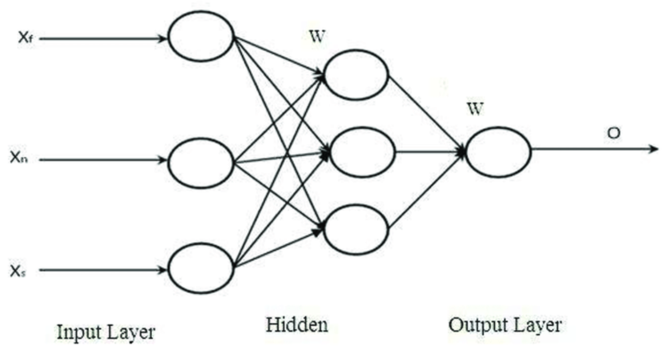

Figure 1: Diagram showing a general architecture of an arbitrary FFNN

As shown in Figure 1 above, as the input travels between nodes, a weight and variable, w, b, along with an activation function f is applied to them and at the output node, another activation function, f_{output}, which is specific to the problem, in this case, a multi-classification problem, is also applied. Note that all nodes in each layer are connected to the next layer, and so on, meaning that this neural network is a fully-connected FFNN.

2.1.2 Convolutional Neural Networks (CNNs)

CNN's are a type of deep learning architecture commonly used for image classification and other tasks that involve high-dimensional data. A CNN is composed of a sequence of layers that transform an input volume of activations through a differentiable function.

The starting input layer of a CNN typically consists of raw pixel values of an image, which are arranged in a three-dimensional array with dimensions representing width, height, and the number of colour channels (e.g. red, green, and blue). Then come the three main types of layers in a CNN: convolutional layers, pooling layers, and fully-connected layers.

1. Convolutional layers compute the output of neurons that are connected to local regions in the input, through a dot product between the weights of the neurons and a small region of the input volume. This operation results in a new volume of activations, which may have a different size and depth depending on the number of filters used.

2. Pooling layers perform a downsampling operation along the spatial dimensions (width and height) of the input volume, resulting in a smaller volume of activations.

3. Fully-connected layers compute the class scores of the input volume, resulting in a volume of size $[1 \times 1 \times N]$ where N is the number of classes.

Each neuron in this layer is connected to all the activations in the previous volume.

The activation function used for these layers is Rectified linear unit (ReLU) function, which applies an elementwise activation function, such as the *max(0,x)* thresholding at zero, to the output of the previous layer. This leaves the size of the volume unchanged. The parameters in the convolutional and fully-connected layers are trained with ADAM-optimized stochastic gradient descent to optimize the class scores computed by the CNN and make them consistent with the labels in the training set. Overall, CNNs transform the input image layer by layer from the original pixel values to the final class scores through a combination of fixed and parameterized transformations.

2.1.3 ADAM optimizer

We used the ADAM optimizer for our momentum-based gradient descent algorithm to train both the FFNN and CNN. ADAM keeps a running average on both the first and second moment of the gradient and uses this information to adaptively change the learning rate for different parameters. It also performs a bias correction to account for the estimation of the first two moments of the gradient using a running average.

2.1.4 Activation Functions

The choice of activation functions is a unique property of neural networks. These are parameters which must be tuned. We chose to have the possibility of having either the *Sigmoid* or *ReLu* activation functions for each hidden layer in our FFNN and CNN. The output layer, however, is bound to have the *Soft Max* function since we are doing multiclass classification.

2.1.5 Dropout

We implemented a dropout algorithm in both our FFNN and CNN such that at every training step, every neuron besides the output neurons has a probability $p = 0.5$ (dropout rate) of being temporarily dropped out and thus ignored during that specific training step. After training, the neurons are no longer dropped out. In our CNN however, our first 2 blocks (Conv2D and Max-Pooling2D layer) had a probability $p = 0.25$, and $p = 0.5$ just before the output layer. Dropout was implemented in our code to prevent overfitting and reduce complexity, but it should be noted that this might have reduced the model's performance when training the data.

2.1.6 Batch Normalization

To address both the problem of vanishing & exploding gradients and the general problem of the distribution of each layer's input changing during training as the parameters of the previous layers change, we employed the technique called

Batch Normalization. This technique adds an operation in the model just before the activation function of each layer such that the input is zero-centred and normalized. The result is then scaled and shifted per layer using a new shifting and shifting parameter.

2.2 Cost/Loss functions

2.2.1 Cross Entropy (CE)

CE is the cost function mainly used when dealing with classification problems. This, much like MSE for linear regression problems is used to gauge the quality of the model, and the parameters to minimize this function are found through the use of GD. The cost function itself can be derived from the Maximum Likelihood Estimation (MLE) function. First, using the MLE principle to define the total likelihood for all possible outcomes from a data set $D = \{(y_i, x_i)\}$ where the data points are drawn independently with the binary labels $y_i \in \{0, 1\}$, we get:

$$P(\mathcal{D}|\boldsymbol{\beta}) = \prod_{i=1}^{n} \left[p(y_i = 1|x_i, \boldsymbol{\beta})\right]^{y_i} \left[1 - p(y_i = 1|x_i, \boldsymbol{\beta}))\right]^{1-y_i}$$

This equation above approximates the likelihood in terms of the product of the individual probabilities of a specific outcome y_i. Rearranging this equation, we can get the log-likelihood and the beginning of our CE function:

$$\mathcal{C}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left(y_i \log p(y_i = 1|x_i, \boldsymbol{\beta}) + (1 - y_i) \log\left[1 - p(y_i = 1|x_i, \boldsymbol{\beta}))\right]\right) \tag{1}$$

Rearranging the logarithms from the log-likelihood function, and adding a negative, since the CE function is the negative log-likelihood, we get:

$$\mathcal{C}(\boldsymbol{\beta}) = -\sum_{i=1}^{n} \left(y_i(\beta_0 + \beta_1 x_i) - \log\left(1 + \exp\left(\beta_0 + \beta_1 x_i\right)\right)\right) \tag{2}$$

Note that the CE function itself is a convex function of the weights β, meaning that any local minimum is the global minimum.

2.2.1.1 CE: Accuracy and loss

The total data set is a sum of the true positive (TP) and true negative (TN) targets or outputs, labelled by n. The accuracy score then becomes the sum of correctly predicted TP and TN cases divided by the sum of true positive and true negative events in our data set. However, it should be noted that accuracy is not always the best metric to use, as it does not consider other factors such as the cost of making a mistake or the balance of classes in the dataset. In some cases, other metrics such as precision, recall, or F1 score may be more appropriate (Hjorth-Jensen, 2021).

$$\text{Accuracy} = \frac{\sum_{i=0}^{n-1} I(y_i = \tilde{y}_i)}{n}. \tag{3}$$

Cross-entropy loss is a measure of the difference between the predicted probabilities and the true labels. This metric is often used when evaluating the performance of a classification model. A lower cross-entropy loss indicates that the model is making more accurate predictions, while a higher cross-entropy loss indicates that the model is making less accurate predictions.

3 Implementation

This section covers how different parts of the theory and method have been implemented into code form.

3.1 Keras Tuner

Over the course of this investigation, we used Keras Tuner to help automate the process of tuning the hyper-parameters for our model, specifically the Bayesian Tuner to adjust and optimize the learning rate and regularization function, the number of layers in a neural network, and the type of activation function. The Bayesian Tuner, a variant of the Hyper-band algorithm, works by utilizing Bayesian optimization when searching for the optimal set of hyper-parameters for a machine learning model. This library works by giving it a set of values for each hyperparameter, then setting a number of trials for it to run, and as it runs it stores the scores obtained from the trials before finally compiling everything at the end, and showing the user the top 10 configurations with the highest validation accuracy for the model. The table shown in Section 4.1 presents our findings for our FFNN and CNN.

It should be noted that while "optimal", there is potential to have performed a deeper investigation into this library by comparing it with our own methods of finding the optimal hyperparameters.

3.1.1 Network architecture

Network architecture refers to the structure of a neural network, including the number of layers and the number of neurons in each layer. The architecture of a neural network can have a significant impact on its performance, as different architectures may be better suited for different types of data and tasks. In this investigation, we used Keras Tuner to help determine the optimal network architecture for our model. When running the Keras Tuner, in regards to the hyperparameters, we allowed it to pick from a selection of:

Table 1: Choices for Keras Tuner

Parameters	FNN	CNN
Neurons	32-256	16-128
Layers/Blocks	1-2	
Activation Function	Sigmoid/ReLU/SoftMax	
Learning Rate	0.01/0.001/0.0001	
Lambda	0.01/0.001/0.0001	
Filters		32-64

3.1.2 Activation Functions

Activation functions are used in neural networks to determine the output of a neuron given a set of inputs. A crucial component of neural networks, they introduce non-linearity that allows the network to learn and model complex relationships in data. There are many different types of activation functions, but some of the most common ones include the sigmoid function, the tanh function, and the rectified linear unit (ReLU) function. In this investigation, we will be using the sigmoid function, ReLU, and SoftMax, an activation function commonly used with multi-class image classification tasks.

3.1.2.1 Sigmoid (σ)

A common activation function used in neural networks, the sigmoid function takes in a weighted sum of all the inputs to a neuron and produces an output in the range of 0 to 1. This makes it a useful activation function for binary classification problems, where the output of the network is interpreted as the probability that a given input belongs to one of the two classes.

In mathematical form, the Sigmoid function is shown as:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{4}$$

where x is the input to the neuron. It can also be seen plotted in the appendix, in Figure 6.

3.1.2.2 ReLU

The Rectified Linear Unit (ReLU) activation function is a non-linear activation function that takes a single numeric input and produces a single numeric output. The output of the ReLU function is defined as the maximum of its input and 0. In other words, if the input is positive, the output is equal to the input, and if the input is negative, the output is 0. Thus, the ReLU function only allows positive values to pass through and sets negative values to 0. This function is often used in the hidden layers of a neural network, as it allows the model to learn non-linear relationships between the inputs and the outputs.

3.1.2.3 SoftMax

The SoftMax activation function is a generalization of the sigmoid function that is often used as the output layer activation function for multi-class classification problems. It takes a vector of real-valued inputs and produces a vector of outputs in the range of 0 to 1, with the sum of all outputs equal to 1. This allows the output of the network to be interpreted as a probability distribution over the possible classes.

3.1.3 Learning Rate (η)

The learning rate is a hyperparameter of a number of machine learning algorithms, including gradient descent-based optimization algorithms such as stochastic gradient descent (SGD), which we worked on in the last investigation. This hyperparameter determines the step size of the algorithm when updating the model parameters in the direction of the gradient. A higher learning rate can lead to faster convergence, but it also increases the risk of overshooting the optimal solution and causing oscillations or divergence. A lower learning rate, on the other hand, may lead to slower convergence, but it also reduces the risk of unwanted divergence. In this investigation, we used Keras Tuner to help determine the optimal learning rate for our model and ended up with a value of $\eta = 0.01$ for our FFNN and $\eta = 0.0001$ for our CNN.

3.1.4 Running the tuner

When using Keras Tuner to tune the hyperparameters, we defined a model and a set of hyperparameters to tune, which was shown above in Table 1. The optimization algorithm (in this case, the Bayesian Tuner) and any additional search criteria, such as the number of trials to run or the time limit for the search need to be defined. Once these have been defined, the tuning process can be started by calling the search function. We ran 50 trials with 20 executions for 10 epochs to test the values in the table, with the search lasting roughly 4 hours for the FFNN and 8 hours for the CNN.

3.2 Additional Parameters

3.2.1 Epochs

An epoch refers to an iteration over the entire data set or a single run through the entire training set, during which the model's parameters are updated based on the gradient of the loss function. The number of epochs can have a significant impact on the performance of a model, as a higher number of epochs can lead to better convergence, but also increases the risk of overfitting. In this investigation, we used the number of epochs as a hyperparameter to tune, in order to determine the optimal number of epochs for our model.

3.2.2 Batch size

The batch size is a hyperparameter that determines the number of samples to process at a time. When training a neural network, batch size determines the number of training samples to use in each iteration of the training process. Larger batch sizes can lead to faster convergence, but also risks increasing the memory requirements of the model and making it more difficult to train on smaller data sets. In this investigation, we decided to use a batch size of 32, as it is a commonly used batch size that strikes a good balance between convergence speed and memory requirements.

3.3 Data set

We use the CIFAR-10 data set from Keras. It consists of 60,000 32x32 colour images in 10 classes, with 6,000 images per class. The classes include airplane, automobile, bird, bird, cat, deer, dog, frog, horse, ship, and truck. The data was already preprocessed with 50,000 images set for training and 10,000 for testing, but we further split it to include some images for validation, leaving us with a 4:1:1 split of 40,000 images for training, 10,000 for validation and 10,000 images for testing.

4 Results & Discussion

This section covers the various results from our investigation and analysis for each of them.

4.1 Keras Tuner Results

4.2 FFNN Analysis & Evaluation

After determining the optimal hyperparameters and architectural parameters for our FFNN through a hyperparameter tuning process, we implemented the FFNN using these parameters and trained it for 100 epochs using a batch size of 32. Upon evaluating the performance of the FFNN on the test dataset, which consists of unseen data, we found that the network had a relatively low accuracy of approximately 16.8%. These results are summarized in Table 3. This poor performance on the test data suggests that the FFNN may not be generalizing well to new, unseen data and is likely overfitting to the training data. Further analysis and adjustments to the model may be necessary in order to improve its performance on the test data.

Table 3: Accuracy & Loss score for FFNN

	FFNN
Accuracy	0.1684
Loss	2.071

Parameters	**FFNN**	**Parameters**	**CNN**
η	0.01	η	0.0001
λ	0.0001	λ	0.0001
Input Layer Neurons	256	Block 1	Yes
Hidden Layer 1	Yes	Filter-size Layer 1	32
Neurons Layer 1	256	Activation Function	ReLU
Activation Function	Sigmoid	Block 2	Yes
		Filter-size Layer 2	64
		Activation Function	ReLU
		Hidden Layer 1	Yes
		Neurons 1	112
		Activation Function	ReLU

Table 2: Keras Tuner results

In order to analyze the model's performance as a function of epochs, we plotted the accuracy and loss of the FFNN on the training and validation datasets. The resulting plots, shown in Figures 2 and 3, reveal that the model exhibits poor performance and unstable behaviour. Additionally, the performance of the FFNN on the validation dataset appears to be better than on the training dataset, which is unexpected.

This unexpected behaviour could potentially be due to overfitting (Hastie, Tibshirani, & Friedman, 2009), where the model has learned patterns in the training data that are specific to that dataset and do not generalize well to other datasets. We deduced this as a sign of over-fitting, but this could also stem from a very noisy data set therefore making it harder for the model to learn from it.

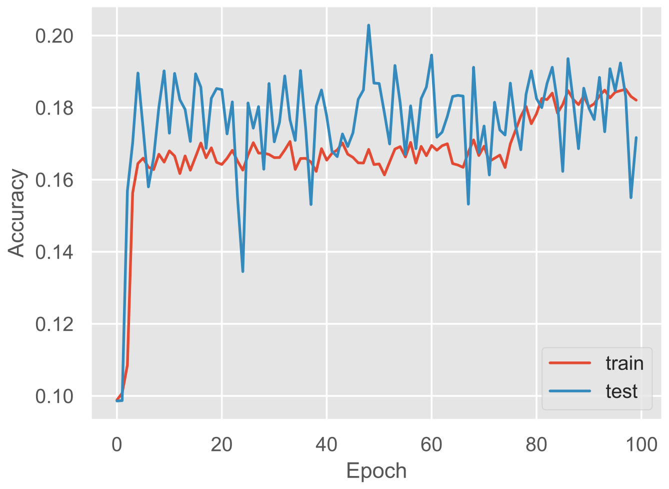

Figure 2: Plot showing accuracy score as a function of epochs for both the train data and validation data labelled as "test" in the plot using the optimal parameters described in Table 2

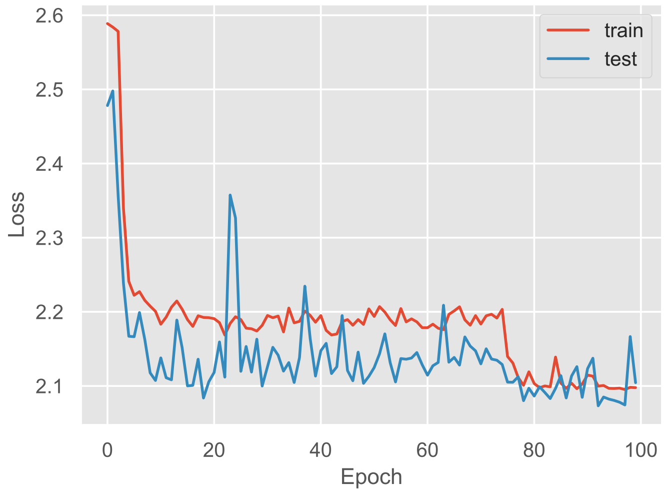

Figure 3: Plot showing loss score as a function of epochs for both the train data and validation data labelled as "test" in the plot using the optimal parameters described in Table 2

4.3 CNN Analysis & Evaluation

Following the process of determining the optimal hyperparameters and architectural parameters for our CNN, we implemented the CNN using these optimal values and trained it for 100 epochs using a batch size of 32. Upon evaluating the performance of the CNN on the test dataset, we found that the network had a relatively high accuracy of approximately 76.0%. These results are summarized in Table 4. This improved performance on the test data compared to the FFNN suggests that the CNN may be more effective at generalizing to new, unseen data and may be less prone to overfitting. These results may indicate that the CNN architecture is better suited for this particular task of multi-class image classification compared to the FFNN architecture.

Table 4: Accuracy & Loss score for FFNN

	CNN
Accuracy	0.7599
Loss	0.7756

Plotting the model's performance as a function of epochs, we see in Figures 4 & 5 that the model seemed to perform better on validation set initially, but as it propagate to more epochs, we see the training data exponentially become better and better whilst the validation data stagnate underneath (for accuracy) or above (for loss) the training data as it seems to come to a convergence.

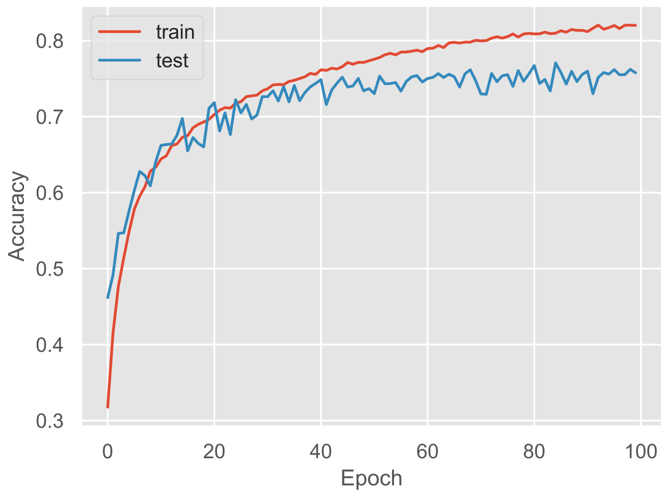

Figure 4: Plot showing accuracy score as a function of epochs for both the train data and validation data labelled as ”test” in the plot using the optimal parameters described in Table 2

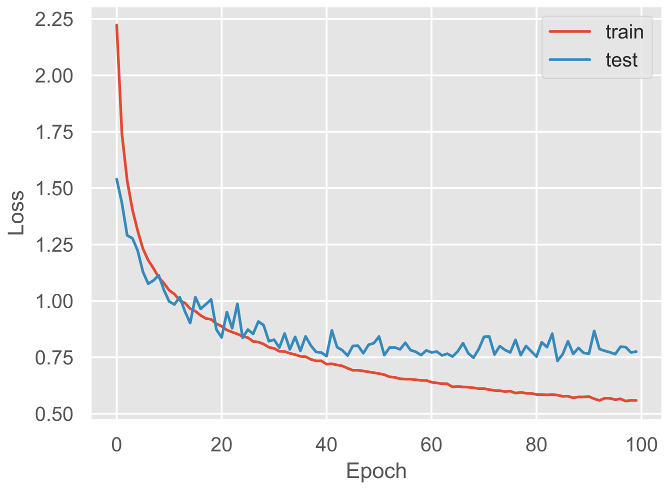

Figure 5: Plot showing loss score as a function of epochs for both the train data and validation data labelled as ”test” in the plot using the optimal parameters described in Table 2

4.4 FFNN & CNN Comparison

Our CNN performed better and was ≈ 60% more accurate than our FFNN. The poor performance of the FFNN on the image classification task was unexpected, as it was expected that the FFNN would at least achieve moderate performance on this type of dataset. There are several possible hypotheses as to why the FFNN performed poorly.

One possibility is the nature of the dataset itself. As previously stated, CNNs are known to perform well on high-dimensional datasets, such as images, due to their ability to reduce the dimensionality of these datasets through the use of convolutional layers. In contrast, FFNNs may not be as effective at handling such datasets due to their lack of convolutional layers. It is therefore possible that the FFNN struggled to learn from the high-dimensional images in the dataset and may have interpreted them as being particularly noisy, leading to poor performance.

Another hypothesis could be the complexity of the task itself. Image classification can be a complex task, especially if the images contain subtle features that are difficult for the model to learn. If the FFNN was not able to capture these features effectively, it may have resulted in poor performance on the task.

Overall, further analysis and experimentation may be necessary in order to fully understand the reasons for the poor performance of the FFNN on this dataset.

4.5 Overall Critical Evaluation

There are several possible explanations for the poor performance of the FFNN on this task. One possibility is that the high-dimensional nature of the image dataset made it difficult for the FFNN to learn and create useful patterns, while the CNN was able to effectively reduce the dimensionality of the data through the use of convolutional layers. Another possibility is that the complexity of the image classification task itself may have overwhelmed the FFNN, leading to poor performance. Further analysis and experimentation may be necessary in order to fully understand the reasons for the differences in performance between the two models. As mentioned before, we feel that it should be noted that while Keras Tuner made it easier to find the "optimal" values for the hyper- and architectural parameters, there is a large potential to have performed a deeper investigation into this library by comparing it with our own methods that we learned throughout the course of finding the optimal hyperparameters.

5 Conclusion

Throughout this investigation, we experimented with and investigated multi-class image classification with the aim of determining which type of neural network would perform best on such tasks, a feed-forward neural network or a convolutional neural network.

Using Keras to build our neural networks, allowed us more flexibility and creativity in the design of our neural networks. We discovered and utilized a very neat library in Keras that allowed us to find the optimal hyper- and architectural parameters for each of the networks without the use of heat maps, granted they did take a considerable amount of time to run.

The results of the comparison between the FFNN and the CNN showed that our CNN had a significantly better overall performance, with an accuracy of 76.0% compared to 16.8% for the FFNN, and a cross-entropy loss score of 0.776 compared to 2.07 for the FFNN. We also found that the sigmoid activation function was the best choice for the FFNN, while the ReLU activation function was the best choice for the CNN (with the exception of the output layer, which was set to SoftMax due to the multi-class nature of the dataset).

Overall, the results of this study demonstrate the effectiveness of convolutional neural networks for image classification tasks, particularly when compared to feed-forward neural networks. The use of the Keras library and Keras Tuner made it easier to develop and fine-tune these models, and the fine-tuning of these hyper- and architectural parameters was found to be a key factor in optimizing the performance of the models.

6 Appendix

Formulas and equations derived from (Hjorth-Jensen, 2021).

6.0.0.1 Plot for the Sigmoid function:

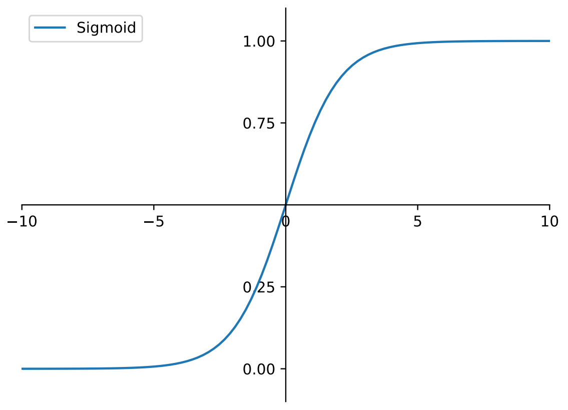

Figure 6: Sigmoid function

6.0.0.2 Plot for ReLU function:

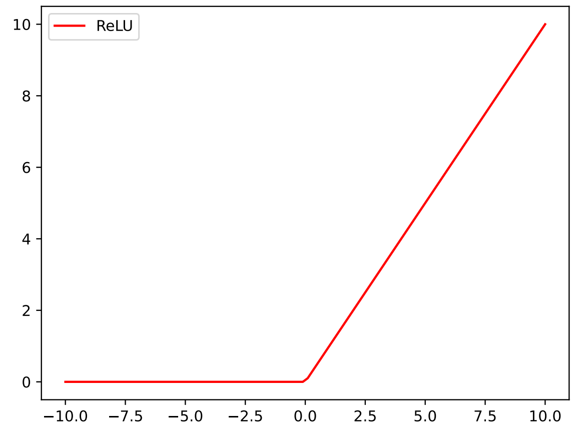

Figure 7: ReLU function

References

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning* (2nd ed.). Springer.

Hjorth-Jensen, M. (2021). Applied data analysis and machine learning. `https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html`.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L., et al. (2019). Kerastuner. `https://github.com/keras-team/keras-tuner`.