Neural Networks Project
CIFAR-10 Image Recognition

EE4305 Introduction to Fuzzy/Neural Systems

Mario Gini, A0175086X
Thomas Michael Hayden, A0175215J

A Cooperation of
ETH Zurich & University of Oxford
at NUS

Contents

1 Introduction

T. M. Hayden & M. Gini

Machine learning and especially artificial neural networks (ANN) have been the object of intense research over the last few decades. Tangible useful results have been obtained over many different fields, from achieving superhuman performance in the game Go (see Section 2.2) to allowing users of social networks to automatically tag their friends in photo albums (see Section 3.3). Whilst there is a lack of the theoretical foundations behind ANN, it is still regarded as the state-of-the-art in many application areas.

The object of this assignment is to give us some exposure into how machine learning - especially ANN - can be applied to practical problems. This report will focus on how a multi-layer perceptron (MLP) can be trained on the CIFAR-10 dataset to perform image classification tasks. The CIFAR-10 data set contains 60000 32×32 RGB images labeled into 10 classes. It is a popular benchmark for image classification algorithms. Due to its relatively small size, training takes a relatively short amount of time which makes it suited for our project.

In the following sections of the report, we give a brief overview of general applications of ANN as well as some some notable recent accomplishments. The state-of-the-art algorithms developed for the CIFAR-10 dataset are reviewed as well. Section 4 summarizes our work implementing a MLP and Section 5 presents the results of implementing a convolutional neural network (CNN) which is a more advanced network architecture.

2 Literature Review on Artificial Neural Networks

M. Gini

This section gives a literature review on the broad topic of ANN. A more specific review on ANN designed to classify the CIFAR-10 dataset is found in Section 3. The significance and applications of ANN will be reviewed in Section 2.1 while recent trends and accomplishments are discussed in Section 2.2.

2.1 Significance and Applications of Artificial Neural Networks

This subsection will illustrate the significance and applications of ANN. Increasing computer power shifted the focus of research towards deep ANN and similar architectures which are coined under the term "deep learning". These powerful deep ANN are nowadays used in a variety of applications[1][2].

ANN are significant because they can work as a black box model. The performance can be improved by data preprocessing, augmentation and by finding an appropriate network architecture and training process. No a-priori knowledge of the classification process itself is required. This makes deep ANN suited for applications where such knowledge is difficult to obtain. Character and speech recognition are such difficult problems, as well as image classification. In speech recognition, deep ANN have been shown to outperform other methods on a variety of speech recognition benchmarks, sometimes by a large margin[3]. In the field of image classification, the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge) marks an important turning point. A convolutional neural network (CNN) architecture won the competition for the first time - by a large margin[4]. In both fields, ANN are now widely accepted as the most powerful approach.

However, the fact that ANN do not incorporate much a-priori knowledge can also backfire. In consequence, a trained model gives little insight into its inner workings and optimal network architectures are basically found through a trial-and-error process. Most design guidelines for deep learning methods are therefore rather based on empirical knowledge than on theoretical foundations.

New methods are developed to better understand the computations deep ANN perform at each layer. The resulting visualizations reveal the process of extracting high level features out of raw input data[5] [6]. In general, each layer extracts higher level features of the input the previous layer provides such that the features are highly abstract after a few layers. The last layer then classifies the input into one of the output categories.

2.2 Recent Trends and Accomplishments

Recent trends and accomplishments of ANN are described in this section. Two recent accomplishments are looked at in detail: The AlphaGo computer program and adversial examples. AlphaGo is a great example to illustrate the great capabilities of ANN. Adversial examples can easily fool very different kinds of neural networks which is a good way to exemplify the limitations the present ANN still possess.

The game Go is a complex board game with the impressive number of around 10^{170} legal positions[7]. Due to its enormous search space and difficulty to evaluate board positions, it is viewed as the most challenging of the classical games for artificial intelligence. A victory of a computer program over a professional human player has been considered to be at least a decade away. However, the computer program AlphaGo beat the European Go champion 5-0 in 2015[8].

AlphaGo makes extensive use of ANN. It consists of a ”value” and a ”policy” network to separately evaluate the board position and select moves. It is trained in a combination of supervised learning from human expert games and reinforcement learning through self-play. The training of such big networks requires notable computation resources. In a recent trend, dedicated hardware to train deep ANN is developed. Besides other adaptions, it is designed to speed up matrix multiplications which are one of the main components of the training process. The most notable example is the Tensor Processing Unit which achieves a 15- to 30-fold performance compared to a contemporary GPU or CPU[9]. It is important to note that the development of deep learning is closely connected to the ever improving available computing power[10].

AlphaGo received considerable media coverage and is considered as one of the most impressive feats of deep learning. In a follow-up paper, a further improved version of AlphaGo is presented, AlphaZero[11]. It uses a single neural network and trains solely through reinforcement learning with self-play, starting with random play. It is only provided with the rules of Go. After only days of training, it defeated all previous versions of AlphaGo and achieved a never seen before playing strength. It is quite intriguing that even for such a complex task, the network can achieve superhuman performance without any provided knowledge besides the rules of the game.

As a second recent trend, adversarial examples recently surprised a lot of researches and became a hot topic of interest. To generate an adversarial example, a slight perturbation is applied to a correctly classified image. The classification process is then repeated and the perturbation is adapted such that the prediction error is *maximized*. A slight perturbation which is not recognizable by a human is already enough to let the neural network misclassify an image with a high confidence level[12]. It has been shown that adversarial examples trained on one model are likely to be misclassified by another model as well, i.e. they possess a transferability property[13].

It is very likely that a randomly selected input to a neural network built from linear parts is processed incorrectly and the models only behave reasonably on a very thin manifold encompassing the training data[14]. This result questions the generalization abilities of ANN. Furthermore, the transferability property allows potential attacks on systems using ANN[15][16]. For example, stop signs could be slightly modified with stickers such that they are misclassified by autonomous vehicles which then behave unexpectedly. Further research is required to develop defense strategies against such attacks. Only then, ANN can deployed in safety critical applications.

3 Literature Review on the CIFAR-10 Dataset

T. M. Hayden

The CIFAR-10 dataset[17] is a well established dataset in the machine learning community. It is challenging because it is a relatively small dataset. Even so, excellent results, even exceeding human performance, have been obtained using a variety of CNN architectures[1]. At the time of writing, the highest published result on the CIFAR-10 dataset was achieved in 2015 with accuracy of 96.53%. This is considerably better than human performance which has an accuracy of around 94%[18].

3.1 Data Augmentation

Like many other machine learning problems, image classification will almost always benefit from additional data[19]. However, even when restricted to a particular dataset such as CIFAR-10 it is possible to generate more data using a technique called data augmentation[20]. Data augmentation manipulates existing images to create 'new' data for use in training.

Common methods to augment images for use in machine learning include mirroring, rotation and image translation[4]. Using these techniques it is possible to train on a dataset that can be several times larger than the original dataset. The leading architectures all made heavy use of data augmentation[21][22][23].

3.2 Leading Architectures for Classifying the CIFAR-10 Dataset

In this section, the results of several different CNN architectures are presented. It should be noted that these architectures were not designed specifically to perform on the CIFAR-10 dataset. As such, they may not be fully optimised and it is likely that they could be improved slightly.

3.2.1 Fractional Max-Pooling

In a standard CNN, convolutional layers are often interspaced with 2x2 max-pooling layers. These max-pooling layers serve to downsample the data. This allows the CNN to be somewhat spatially invariant to the locations of the features and improve accuracy. However each max pooling layer also removes 75% of the data[21]. This in effect reduces the maximum depth of the CNN due to the disjoint nature of the max-pooling regions.

By using a new approach known as fractional max-pooling, it is possible to max-pool using a non-integer mask size. In this manner, the size of the hidden layers is reduced by a lesser amount and it is possible to create deeper networks without having to add consecutive convolutional layers. This is important as generally deeper networks will lead to stronger classifiers[24]. However, deeper networks are also in general more expensive to train.

[1] http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

An architecture based on fractional max-pooling currently has the highest published classification accuracy on the CIFAR-10 dataset at 96.53%. This architecture also made heavy use of data augmentation. Additionally, the model was 'fine-tuned' after initial training by re-training on the original dataset for a few epochs using a low learning.

3.2.2 The All Convolutional Net(ALL-CNN)

In this architecture[23] a CNN consisting entirely of convolutional layers is proposed. Max-pooling layers are instead replaced with convolutional layers with increased stride. These increased stride layers act in a similar way to max-pool layers in that they downsample the data and allow the CNN somewhat invariant feature location. This architecture has an accuracy of 95.59% which is the 2nd highest published result. This architecture also makes heavy use of data augmentation.

3.2.3 Layer-Sequential Unit-Variance (LSUV) Initialization

LSUV initialisation provides a method to initialise deep CNN. This produces networks with better accuracy than uninitialised networks. In addition LSUV greatly accelerates the training of CNNs. An architecture based on the LSUV method machine managed to achieve an accuracy of 94.16%. Note that this only used a moderate amount of data augmentation.

It is important to stress the use of data augmentation when looking at these results. Table 1 shows the results of the three leading architectures along with the amount of data augmentation. Moderate data augmentation consists of mirroring in the horizontal axis and small translations in each axis. Extreme data augmentation involves upscaling the images to 126×126 pixel images and performing a variety of operations such as shearing, colour augmentation, rotation, translation and scaling. It may be the case that with additional data augmentation, LSUV outperforms the max-Pooling approach.

Data Augmentation	**Fractional Max-Pooling**	**ALL-CNN**	**LSUV**
None	-	90.92%	-
Moderate	-	92.75%	93.94%
Extreme	96.53%	95.59%	-

Table 1: Table showing the results of the leading CIFAR-10 architectures.

3.3 Application Areas of Image Recognition Algorithms

There are numerous applications of image recognition algorithms across many different fields. However in order for neural networks to be effective, large amounts of labeled data must first be collected. In practice labelling and uploading of images is mostly done by users of the application by 'tagging' images. For example in a stock image database such as Shutterstock[2], users would be prompted to tag uploaded images with the image contents. This provides Shutterstock with an enormous amount of data perfect for machine learning. This has allowed Shutterstock to develop powerful new tools to label images using machine learning. For example the newly released compositionally aware search which allows users to search for images with specific objects in different locations of the image[25].

[2] `https://www.shutterstock.com/`

Another application of image recognition is to automatically tag recognised faces when uploading photos onto social media websites. This is useful as it is often tedious to tag each image in large albums. Automatic tagging algorithms have been developed using machine learning to automatically tag faces with accuracy as high 99% [26]. This allows users to upload entire albums without having to tag each photo individually.

4 Multi-Layer Perceptron Classifier

M. Gini & T. M. Hayden

This section presents an MLP classifier designed to classify the CIFAR-10 dataset. The section is organized as follows: Section 4.1 introduces the software setup used to implement the MLP classifier. Section 4.2 discusses the data preprocessing and augmentation. Section 4.3 analyzes the effect of different network structures on performance. Section 4.4 analyzes the effect of different hyperparameters. Finally, Section 4.5 presents the optimized MLP classifier.

4.1 Software Setup

MATLAB's Neural Networks toolbox is employed to implement the MLP classifier. The toolbox provides convenient algorithms and applications to design the MLP. A network training function with a convenient graphical user interface (GUI) to observe the progress is included as well. Figure 1 shows the GUI.

Since this is a classification problem, parts of the network structure are given. The output of the MLP should be a prediction of to which class the input belongs to. This is accomplished with the help of the softmax function, also called normalized exponential function. Equation 4.1 shows the formula of such a function. A softmax layer is then used as the last layer. It gets a K-dimensional input vector $\boldsymbol{z}$ of arbitrary real values and "squashes" it into a K-dimensional output vector $\sigma(\boldsymbol{z})$ of real values in the range $[0, 1]$. In our case, $K = 10$ and the output values represent the probabilities that the input belongs to the respective class. The class with the highest probability then constitutes the predicition of the MLP.

$$\sigma(\boldsymbol{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for} \quad j = 1, \dots K. \tag{4.1}$$

Every ANN will have at the very least one input layer and one output layer. The size of the input layer simply depends on the size of the input data vector. In the case of the CIFAR-10 dataset, the input size is 1×3072, the number of pixels per image. Since the dataset is to be classified into 10 categories, the output layer is a softmax layer with 10 nodes.

The other hidden layers consist of standard MLP. The input to the MLP are the pixel values of the image. Preprocessing and augmentation as described in the next section is done before the pixel values are fed into the network. MATLAB offers lots of adjustable settings for the MLP. Unless mentioned otherwise, the following settings are employed as default settings:

- Training function: 'trainscg', the scaled conjugate gradient method.
- Loss function: 'crossentropy'
- Activation function: 'tansig'
- As a default network structure, two hidden layers are employed with 100 and 50 neurons in the first and second layers respectively.

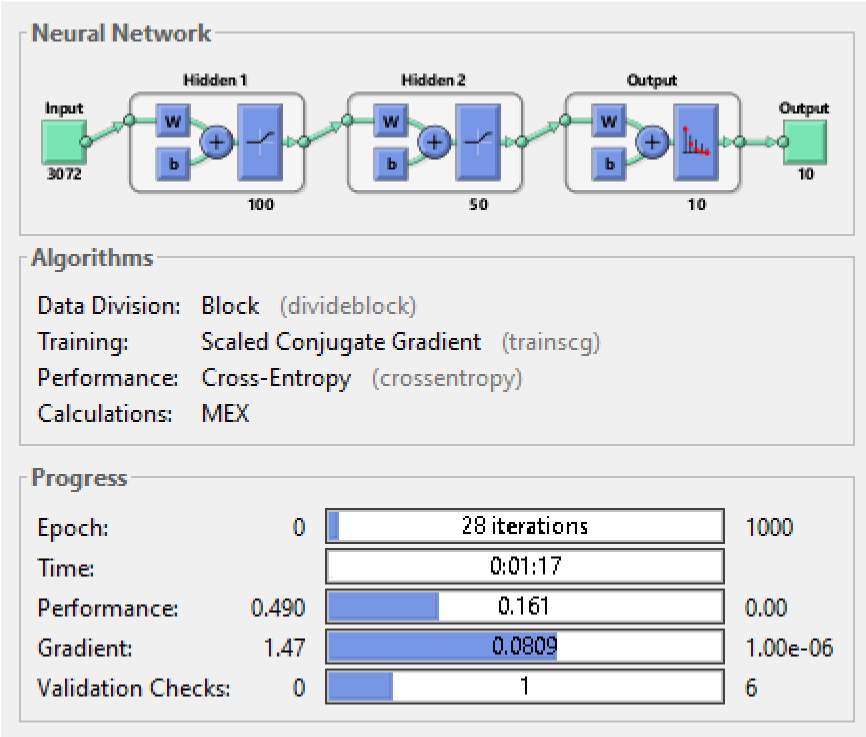

Figure 1: Screenshot of MATLAB's nntraintool.

- Weight initialization: The weights of the neurons are randomly initialized. Since this leads to slightly different performance for each run, the performance analysis for each configuration is run five times and then the test accuracy is averaged. The standard deviation of the different runs is then plotted as well.
- The training batch size is set to 20000 images. This constitutes a compromise between a good performance and a reasonable training time. 80% are used for training and 20% are used for validation.
- The testing batch consisting of 10000 images is employed for testing of the MLP.

4.2 Data Preprocessing and Augmentation

Data preprocessing and augmentation take place before the data is fed into the network. In the preprocessing step, the data is normalized and centered around the mean. In the augmentation step, the amount of data is augmented through operations like image flipping.

4.2.1 Data Preprocessing

Each image of the dataset is represented by a $32 \times 32 \times 3$ array, which results in 1024 pixel values per color channel. To be processed by the MLP, it is first transformed into a 1×3072 array. The pixel values are integers in the range [0,255]. For normalization, the data is divided by 255 to lie within the range [0,1]. Accordingly, the datatype changes from the integer type to double. In a second step, the mean per pixel over the whole training set is subtracted. This centers the data per channel.

Data preprocessing also includes the division of the complete dataset into appropriate training, validation and test data batches. There are 50000 images available for training. With the default splitting into training and validation dataset (80% for training and 20% for valida-

Figure 2: Comparison of network performance with and without data preprocessing. The errorbars represent one standard deviation. The default MLP classifier is used.

tion), Figure 2 shows the effect of varying the training batch size as well as employing data preprocessing.

As intuitively expected, the test accuracy increases for an increasing training batch size. However, the data preprocessing does not lead to a significant increase of performance. The normalization and mean subtraction do not have a big influence in the specific case of the CIFAR-10 dataset since the input values already lie in a well defined range and the mean per pixel is also a almost constant value over all pixels.

4.2.2 Data Augmentation

Figure 2 shows that a larger training batch size leads to an increased performance. A natural approach to increase classification accuracy is therefore to increase the training batch size. This can be done artificially by implementing data augmentation. We do this by implementing image flipping and image rotation. Figure 3 illustrates the performance gain by using horizontal mirroring. The test accuracy is significantly increased for all training batch sizes by around 3 %.

Figure 4 illustrates the data augmentation process on a typical image. The image is mirrored and rotated three times by 90 degrees. This results in a 8-fold increase of the training batch size.

4.3 Optimization of Network Structure

Choosing the correct architecture of a neural network remains an area of study which is still not fully understood[27]. For a given problem, there is a variety of valid MLP architectures. Approaches to choose an architecture are mostly based on heuristics and therefore not fool-proof[27].

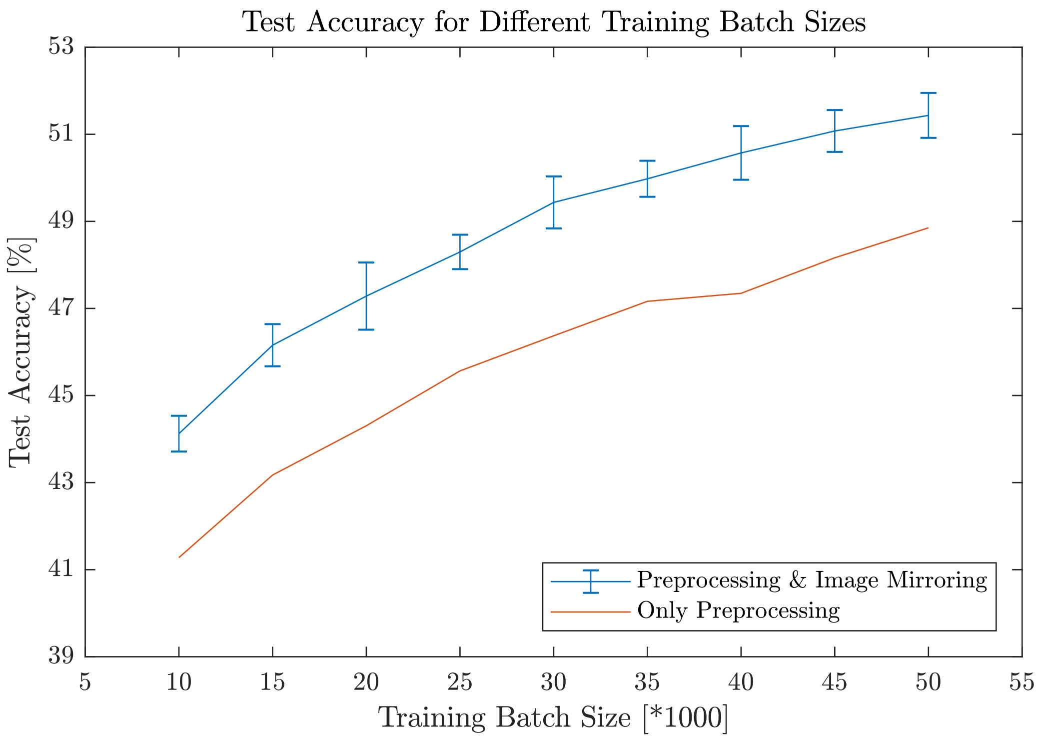

Figure 3: Comparison of network performance with and without data augmentation. The errorbars represent one standard deviation.

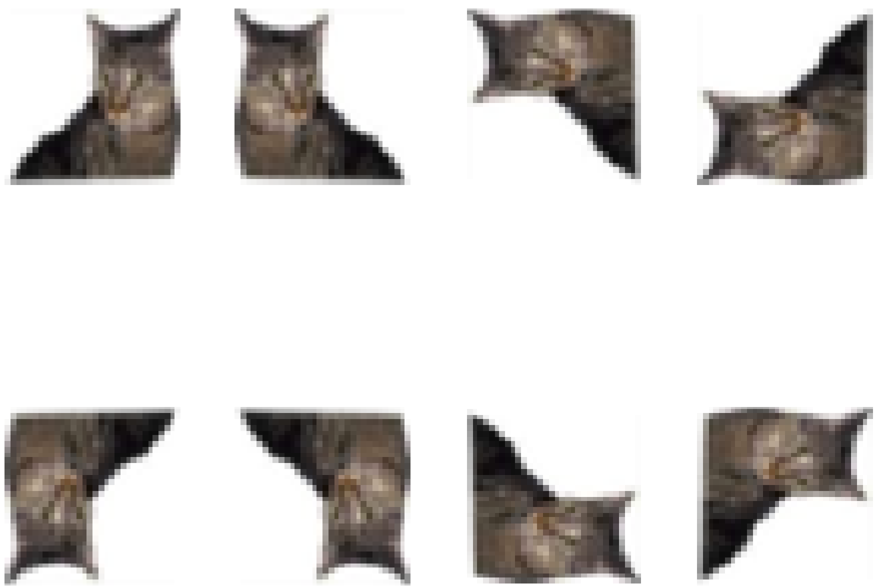

Figure 4: Illustration of the data augmentation process on a cat image. Image mirroring and rotation is employed.

4.3.1 Number of Hidden Layers

In general, a MLP can have an arbitrarily large number of hidden layers. However, MLP architectures with a large number of hidden layers are not common. This is because adding more layers increases the chance that the classifier finds a local minimum [28]. Additional hidden layers rarely result in improved performance. Figure 5 shows the effect of varying the number of hidden layers of our default MLP classifier. Note that as expected, the test accuracy remains almost constant for more than two hidden layers.

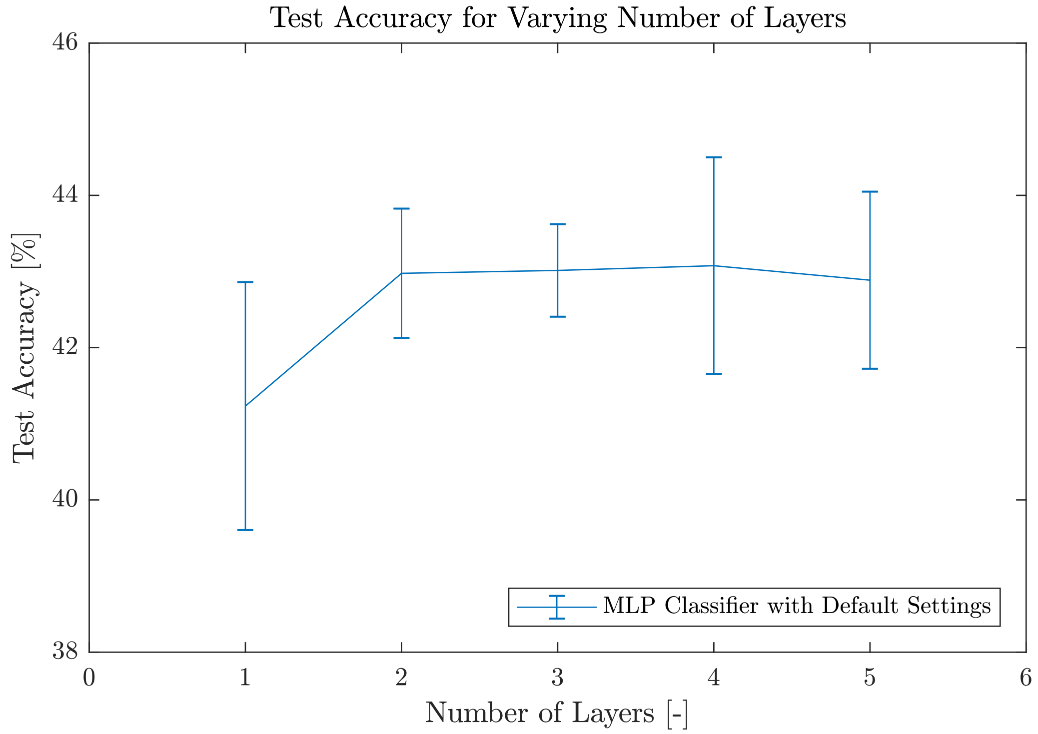

Figure 5: Effect of varying the number of hidden layers of the default MLP classifier. The errorbars represent one standard deviation of the averaging process.

4.3.2 Number of Neurons

Choosing the optimum number of neurons per layer is a challenging task when designing any MLP architecture. Even in modern archetectures, the number of neurons is generally first estimated using empirical rules and then optimised for the particular dataset [29]. This is especially the case when using noisy datasets such as CIFAR-10.

Figure 6 shows the result of varying the number of neurons using two hidden layers with our default MLP. The results show that the number of neurons in the first layer have very little effect on the test accuracy. In addition, the results also show that increasing the number of neurons beyond around 70 had very little to no effect. Increasing the number of neurons also increased the time to train. The number of neurons in the second layer also has a much stronger impact on the time taken to train than the number of neurons in the first layer.

4.4 Optimization of Network Hyperparameters

Hyperparameter selection for ANN such as MLP has become an active field of research with various algorithms being used to estimate optimal parameters [30]. However, since these algorithms rely on performing many trials and updating the parameters accordingly, they proved unsuitable for our purposes due limited computational resources. Instead, a few parameter are varied whilst using the default MLP structure. Optimal parameters are then estimated from the observed trends.

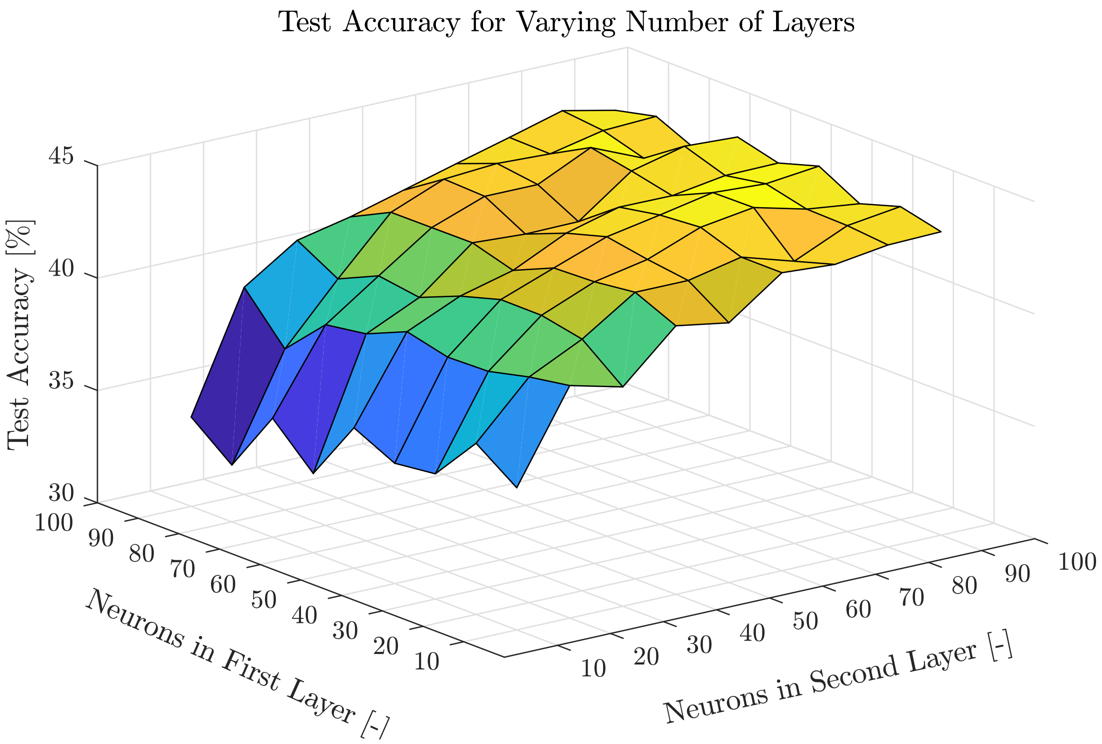

Figure 6: Surface plot showing the effect of varying the number of neurons in the hidden layers of the MLP

4.4.1 Performance Function

The performance function is used to measure the error that a specific input produces in the network. The error expression is then used in the training algorithm to update the weights of the model[31]. MATLAB offers the following different performance functions:

- SAE, the sum absolute error function
- SSE, the sum squared error function
- MAE, the mean absolute error function
- MSE, the mean squared error function
- Cross-entropy error function

Figure 7 shows that three performance functions result in a similar performance: MSE, SSE and Cross-entropy. SAE and MAE result in a slightly lower performance. An explanation might be that these two functions take the absolute error instead of the squared error. Using the squared error leads to a stronger penalization of larger errors. The best performance is achieved using the cross-entropy performance function. This performance function heavily penalizes large errors with very little penalty for small errors. It is also recommended by MATLAB for classification tasks.

4.4.2 Activation Function

The activation function controls the firing of a single neuron. After multiplying the input with the weight and bias vector, the result is fed into the activation function. The output of the activation function constitutes the output of the neuron. Again, MATLAB offers a variety of

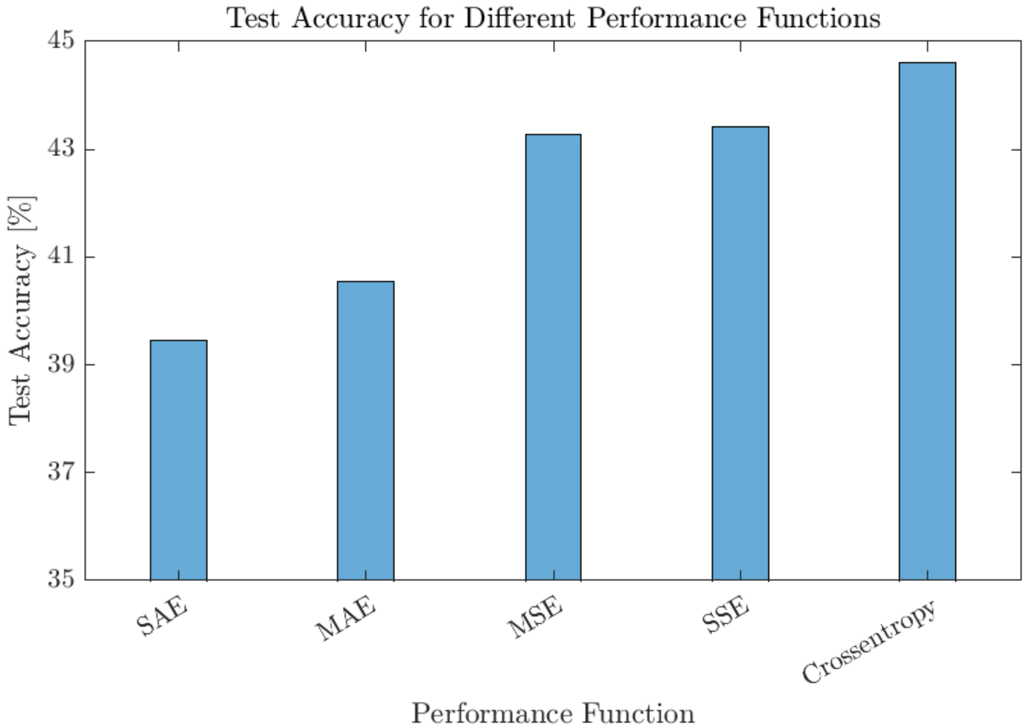

Figure 7: Comparison of the test accuracy of the default MLP using different performance functions.

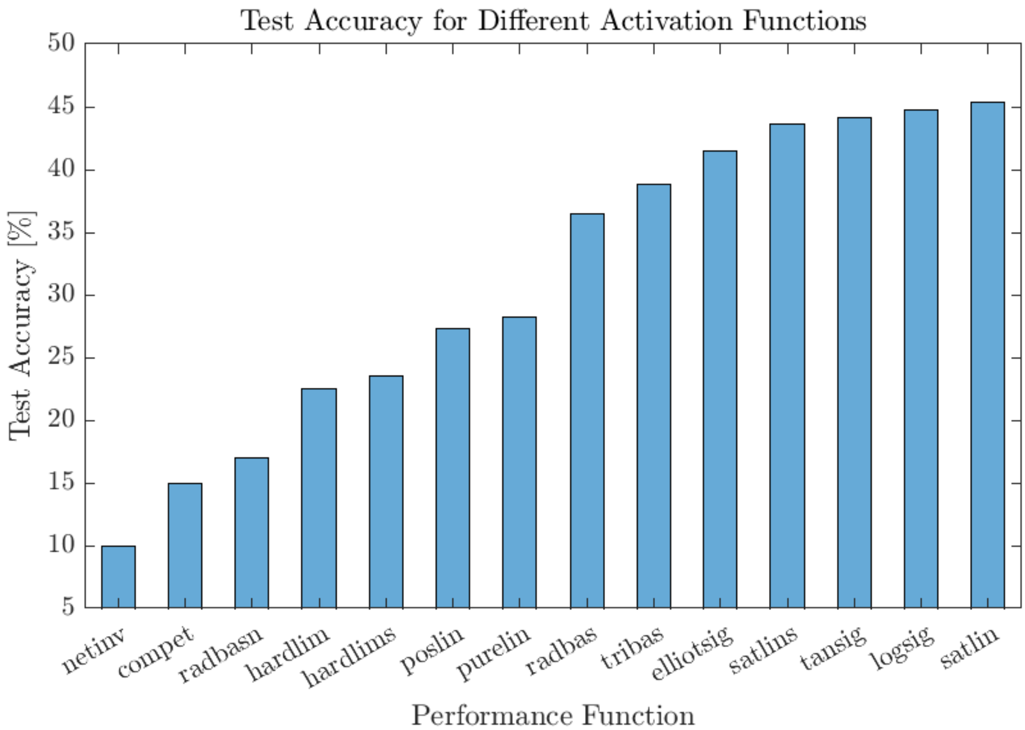

Figure 8: Comparison of the test accuracy of the default MLP using different activation functions.

different activation functions. For the sake of completeness, all activation functions are tested with the default MLP. The result is displayed in Figure 8.

It can be seen that several activation functions are not suited for a MLP classification problem, e.g the 'netinv' and 'compnet' type. Furthermore, there is a class of activation functions which are very similar and all perform well on that specific problem. Specifically, sigmoid shaped activation functions seem to be the appropriate choice for our problem setting. 'tansig' and 'logsig' are quite similar and 'satlin' and 'satlins' can be viewed as linear approximations of those nonlinear activation functions. It is interesting to note that using the 'satlin' type results in the best performance.

4.4.3 Training Function

The training function is the algorithm that dictates the training process of the network. Once again, MATLAB offers a wide selection of training algorithms. Most are variations of the backpropagation algorithm. Figure 9 shows the results of using various training algorithms on the default MLP structure.

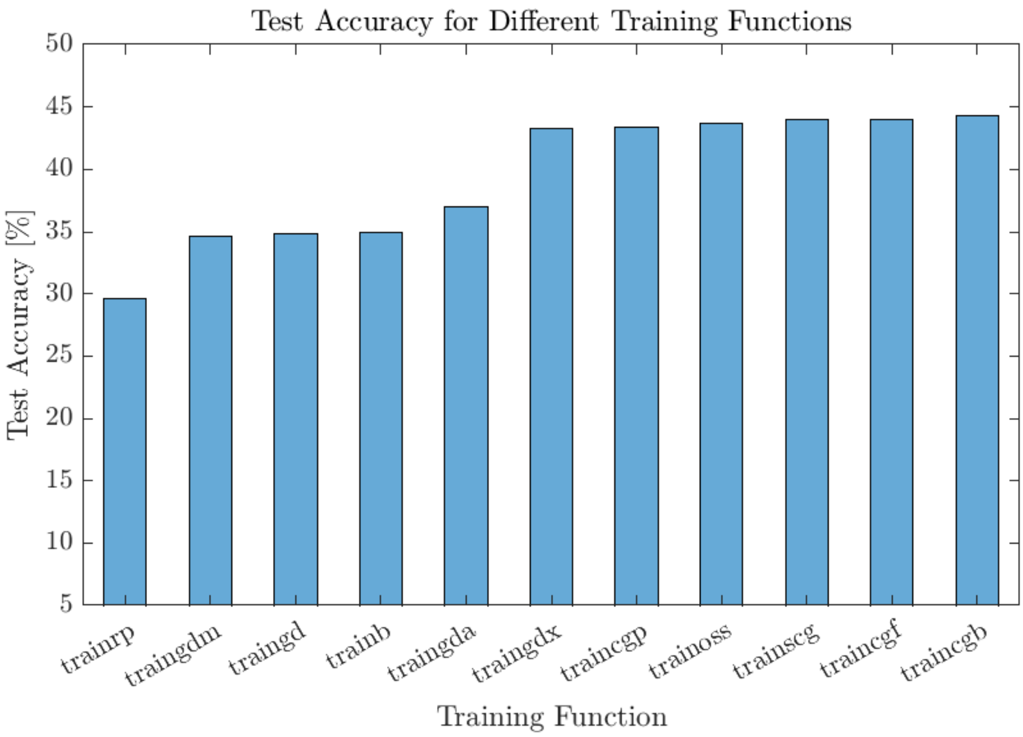

Figure 9: Comparison of the test accuracy of the default MLP using different training functions.

All variations of the conjugate gradient backpropagation algorithm[32] achieve good performance. It is interesting that the performance is superior compared to variations of simple gradient backpropagation. 'traindx' which uses momentum is the only gradient backpropagation variation which achieves similar performance than conjugate gradient backpropagation.

An advantage of conjugate gradient backpropagation algorithms is that they do not require a learning rate parameter. Since our investigations show that they are superior over other backpropagation methods, no further analysis on finding optimal learning rates is conducted.

4.5 Optimized Classifier

We now attempt to use the information we have gathered to create our final MLP classifier. The parameters used are summarised below.

- Training batch size is set to all 50000 images. This will give the classifier the most data to work with and has consistently given the best results.
- Horizontal mirroring is used to augment the data.

- Data is mean normalised between -0.5 and 0.5. This has been shown to improve results slightly.
- The scaled conjugate gradient backpropagation training function is used. This algorithm proved to be quite fast and gives good results.
- The architecture will use two hidden layers with a softmax final layer. Both hidden layers have 100 neurons. Testing on the whole data set revealed that an increased number of neurons in the first layer improved results slightly.
- The cross-entropy performance function is used as this gives the highest performance.
- The saturating linear activation function is used as this gives the highest performance.

Using these results we managed to get a result of 52.61%. The confusion matrix is shown in figure 10. Table 2 shows how the labels used in the model correspond to real class categories. Note that the highest misclassification exhibited in the confusion matrix is between cats and dogs. This is not surprising as the general form of cats and dogs are quite similar and the poses of both vary considerably throughout the data set.

The MLP was most effective at classifying the ship class. This is because the ship class generally has many distinctive features such as a blue background. In contrast, the dog class performed the worst. The dogs can appear with a variety of colours, poses, backgrounds and scene prominence. This makes it a challenging category to classify.

The MLP took 35 minutes to train and is capable of classifying the entire test set in under 1 second.

Label	Class
1	Airplane
2	Automobile
3	Bird
4	Cat
5	Deer
6	Dog
7	Frog
8	Horse
9	Ship
10	Truck

Table 2: The different classes in the CIFAR-10 dataset.

4.5.1 Possible improvements

There are many improvements which could improve the accuracy of our MLP classifier. Perhaps the simplest and most effective of these is to use more data augmentation. For our final model we only used horizontal mirroring due to RAM constraints. Image rotations and translations are both commonly used to increase performance in image recognition neural networks. In addition other advanced data augmentation methods, such those used in the fractional max-pooling archetecture[21], could be used.

With more computational power and RAM, it would be possible to do a more thorough hyper parameter search. In particular the number of neurons used in the hidden layers is likely

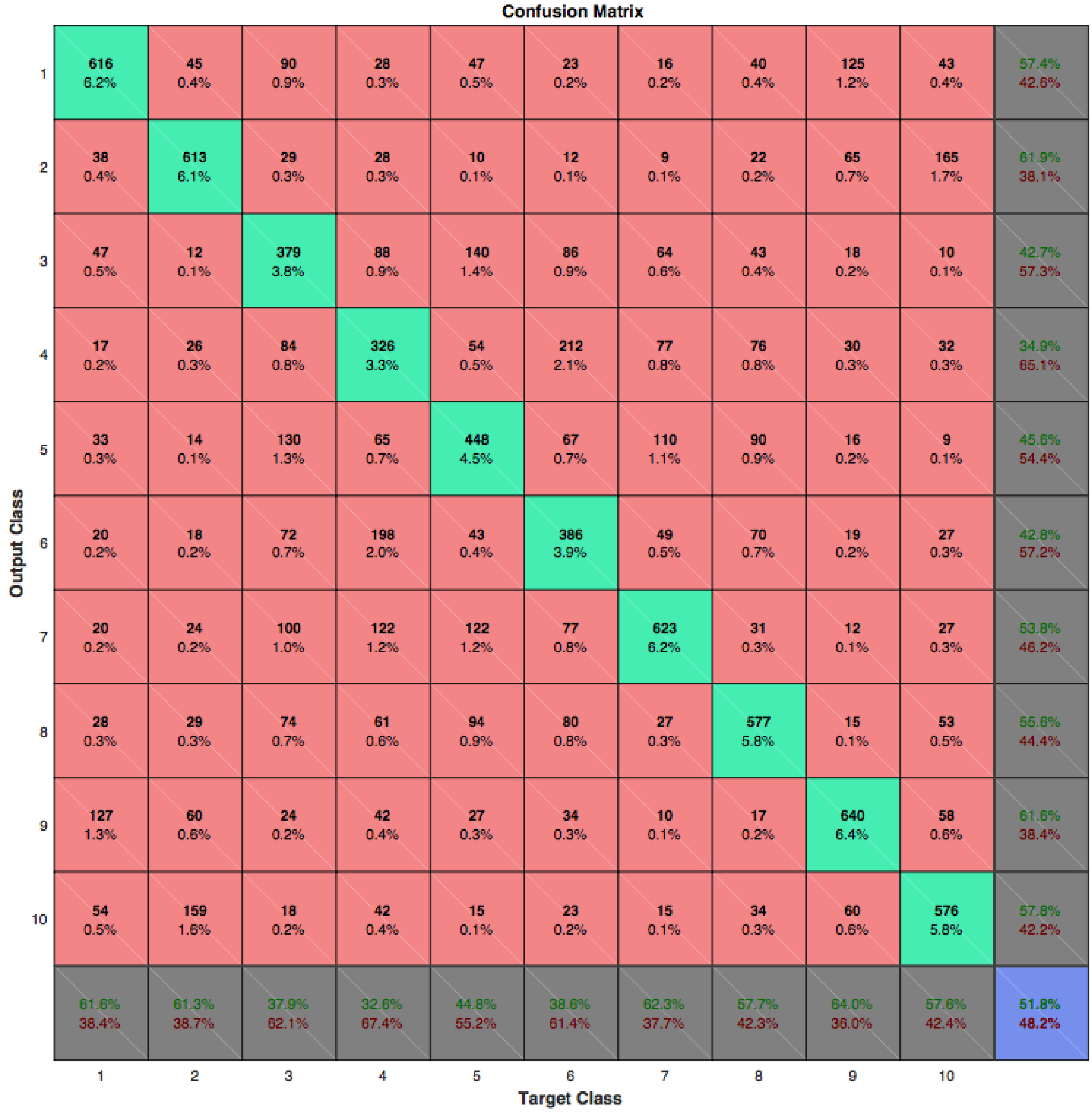

Figure 10: Confusion Matrix for the final MLP classifier

not optimal. A more thorough approach would be to test different numbers of neurons on the entire augmented dataset.

Weight initialisation should also be investigated. For our MLP, we simply used a random initialisation. However as discussed in section 3.2.3, a good initialisation was shown to improve preference of a CNN. It is likely that a similar method could also improve results for an MLP.

In addition it may be possible to improve the validation process. Our model uses a hold out validation strategy. This involves reserving a section of the data to be used as the validate. There are alternative approaches such as k-fold cross validation which have been shown to produce better results[33].

5 Convolutional Neural Network Classifier

M. Gini & T. M. Hayden

CNN is a more advanced architecture than a basic MLP. Similarly to a MLP, it consists of an input and an output layer, as well as several hidden layers. The hidden layers typically consist of convolutional units and fully connected layers. Due to that, CNN are usually much deeper than MLP networks.

CNN require significantly more computing power to train. Even though MATLAB supports training on the GPU, some quick maths reveal that our computational resources are insufficient for extensive parameter search on a CNN. A basic CNN is implemented to compare the performance with the MLP.

5.1 Network Structure

Since the number of hidden layers of a CNN can be quite large and many different types of layers are available, the number of hyperparameters is even larger than for a MLP classifier. This requires unreasonable amounts of training time. Therefore, this section only presents the final version of our CNN which is found through small trial-and-error tests. It consists of an input layer, convolutional units, fully connected layers and an output layer. The functionality of each layer type is briefly described below.

- **Input Layer**
  This layer inputs the raw image data to the network. It also applies data normalization by subtracting the per-pixel mean value over the training dataset. It therefore includes the data preprocessing step.

- **Convolutional Units**
  A convolutional unit consists of several layers. The first layer is always the *convolutional layer*. It has parameters for filter size and depth, which control the size and number of the feature maps the layer is analyzing. This layer is followed by a *batch normalization layer* which normalizes the activations and gradients propagating through a network, making network training an easier optimization problem.

  Then, a *rectified linear unit* (ReLU) follows. This is a nonlinear activation function very commonly used in CNN. Finally, a convolutional unit has a *max-pooling layer* that reduces the spatial size of the feature map. This removes redundant spatial information. As a consequence, the number of layers can be increased without increasing the amount of computation time per layer. A max-pooling layer simply returns the maximum values of rectangular regions of inputs.

- **Fully Connected Layers**
  Fully connected layers are usually employed between the convolutional layers and the output layer. As the name suggests, they are fully connected to all neurons in the preceding layer. It therefore combines all the features learned by the previous layers across the image to identify larger patterns.

- **Output Layers**
  The output layer consists of a softmax layer as described in Section 4.1 together with a classification layer which simply chooses the class that achieved the highest probability by the softmax layer.

Table 3 shows the implementation of our final CNN in MATLAB. Three convolutional units are employed with a filter size of 3 pixels and a varying number of filters. For the max-pooling layers, a filter size of 2×2 pixels is used. Two fully connected layers and the output layers complete our CNN architecture.

Layer Type	**MATLAB Implementation**
Input Layer	Imageinputlayer([32,32,2])
First Convolutional Unit	convolution2dLayer(3,16,'Padding',1) batchNormalizationLayer reluLayer maxPooling2dLayer(2,'Stride',2)
Second Convolutional Unit	convolution2dLayer(3,32,'Padding',1) batchNormalizationLayer reluLayer maxPooling2dLayer(2,'Stride',2)
Third Convolutional Unit	convolution2dLayer(3,64,'Padding',1) batchNormalizationLayer reluLayer maxPooling2dLayer(2,'Stride',2)
Fully Connected Layers	fullyConnectedLayer(10) fullyConnectedLayer(10)
Output Layers	softmaxLayer classificationLayer

Table 3: The CNN structure as implemented in MATLAB.

5.2 Network Training Results

MATLAB has introduced a convenient CNN training GUI in the R2017b release. Figure 11 shows a screenshot of the GUI that allows to observe the training progress. Together with the GUI, several applications have been updated as well. For example, any GPU is automatically detected and used for training. Our CNN training takes place on a single GPU of type NVIDIA GeForce GT 640M. The training of a single configuration takes around 20 min. The architecture described in Table 3 is found to obtain a test accuracy of 74.22%. This is considerably better than the MLP even though it is not yet a fully optimized CNN architecture.

Figure 12 shows the confusion matrix for the CNN. The most likely missclassifications are the same as for the MLP case, e.g. the misclassification between cats and dogs is significantly higher than inbetween other classes.

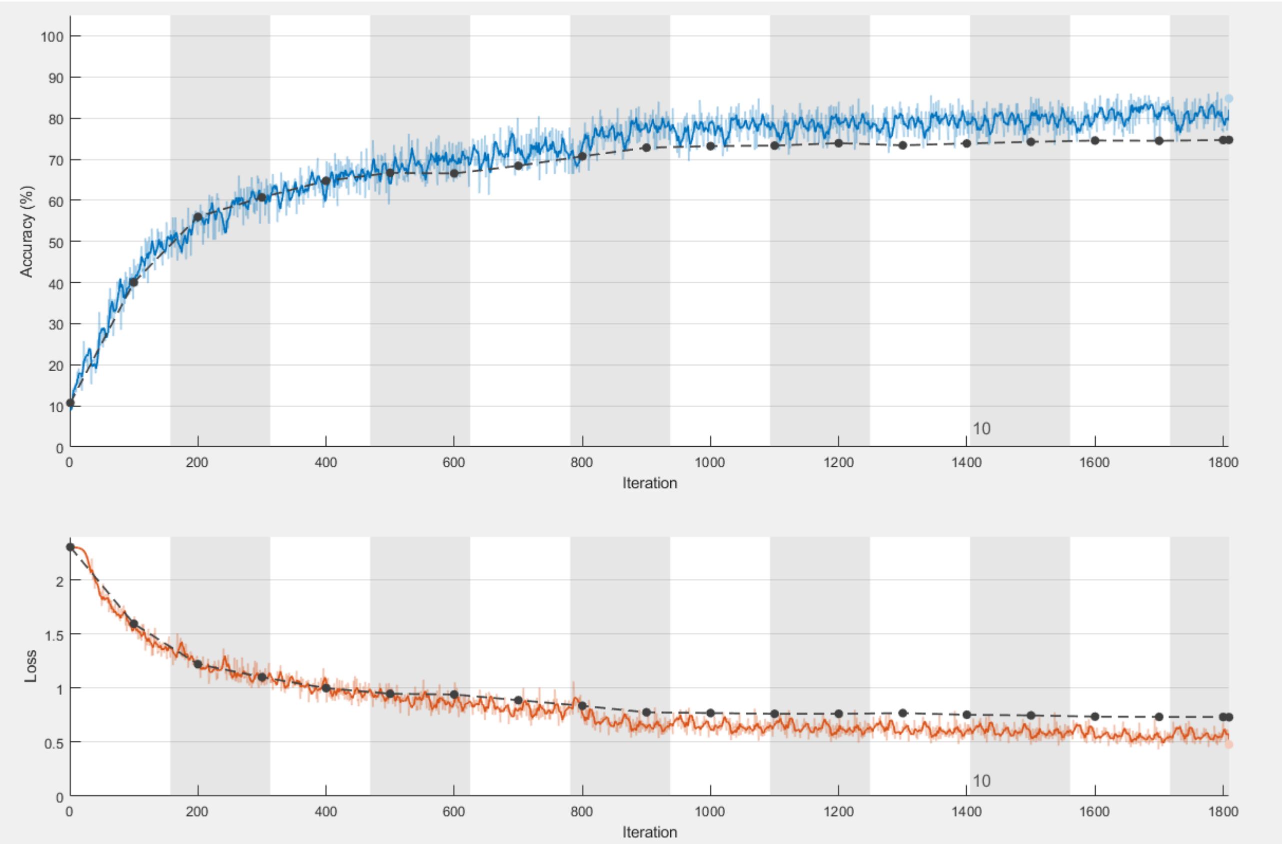

Figure 11: Screenshot of MATLAB GUI for training a CNN. The blue & red curves represent performance on the training dataset, while the black line represents the performance on the validation dataset.

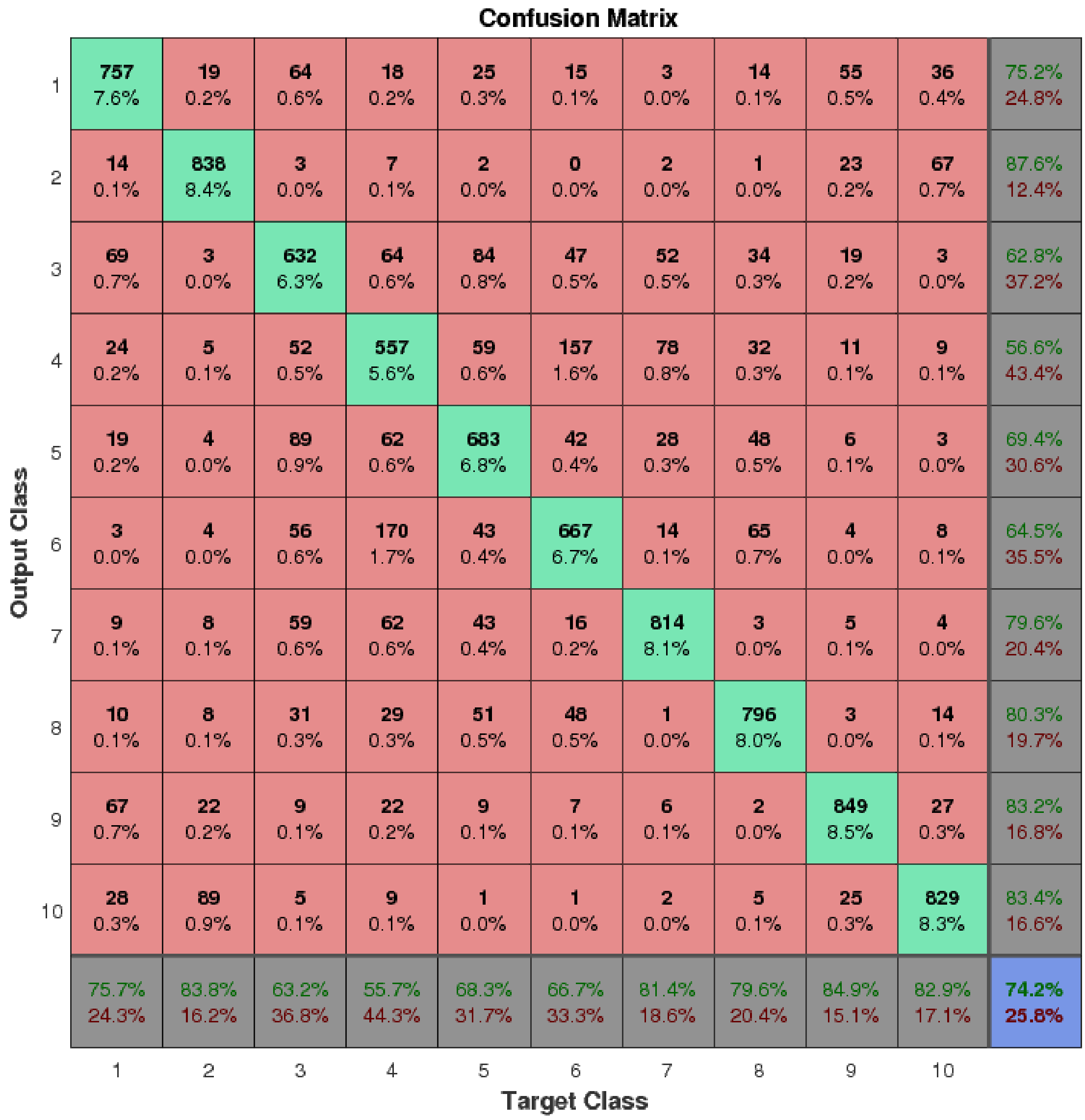

Figure 12: Confusion matrix for the final CNN.

6 Conclusion

M. Gini & T. Hayden

A literature review was carried out showing the strengths and limitations of using ANNs for a variety of tasks. The state-of-the-art architectures for the CIFAR-10 data set were examined and possible improvements were suggested. In addition, application areas of image recognition algorithms were investigated and ANNs proved key to progress.

We also trained our own MLP and CNN classifiers in MATLAB. The MLP network architecture and hyper parameters were heavily tested and optimised to give a satisfactory MLP classifier. Whilst the results of the MLP classifier do not compare favourably against the state-of-the-art, they prove that even using relatively simple methods it is possible to achieve good results. Testing of the CNN revealed that it heavily outperformed the MLP classifier. Possible improvements to increase the performance of both networks have been suggested.

This report has proved machine learning to be an excellent tool when applied to image classification tasks. It has been demonstrated that ANNs can be applied as a black box model with no a-priori knowledge of the classification process being required.

Bibliography

[1] Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

[3] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[5] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going Deeper into Neural Networks, June 2015. URL `http://googleresearch.blogspot.com/2015/06/inceptionism-going-deeper-into-neural.html`.

[6] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[7] John Tromp and Gunnar Farnebäck. Combinatorics of go. In *International Conference on Computers and Games*, pages 84–99. Springer, 2006.

[8] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[9] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. *arXiv preprint arXiv:1704.04760*, 2017.

[10] Jim X Chen. The evolution of computing: Alphago. *Computing in Science & Engineering*, 18(4):4–7, 2016.

[11] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[12] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[15] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[16] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2017.

[17] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[18] Andrej Karpathy. Lessons learned from manually classifying cifar-10. *Published online at http://karpathy. github. io/2011/04/27/manually-classifying-cifar10*, 2011.

[19] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.

[20] Xiaodong Cui, Vaibhava Goel, and Brian Kingsbury. Data augmentation for deep neural network acoustic modeling. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 23(9):1469–1477, 2015.

[21] Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[22] Dmytro Mishkin and Jiri Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

[23] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

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

[25] Mike Ranzinger, Nicholas Lineback, and Nathan Hurst. Composition aware search.

[26] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

[27] Tim Andersen and Tony Martinez. Cross validation and mlp architecture selection. In *Neural Networks, 1999. IJCNN'99. International Joint Conference on*, volume 3, pages 1614–1619. IEEE, 1999.

[28] Jacques De Villiers and Etienne Barnard. Backpropagation neural nets with one and two hidden layers. *IEEE Transactions on Neural Networks*, 4(1):136–141, 1993.

[29] Steve Lawrence, C Lee Giles, and Ah Chung Tsoi. What size neural network gives optimal generalization? convergence properties of backpropagation. Technical report, 1998.

[30] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[31] Robert Hecht-Nielsen et al. Theory of the backpropagation neural network. *Neural Networks*, 1(Supplement-1):445–448, 1988.

[32] Martin Fodslette Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.

[33] Avrim Blum, Adam Kalai, and John Langford. Beating the hold-out: Bounds for k-fold and progressive cross-validation. In *Proceedings of the twelfth annual conference on Computational learning theory*, pages 203–208. ACM, 1999.