School of Information Technologies
Faculty of Engineering & IT

ASSIGNMENT/PROJECT COVERSHEET - GROUP ASSESSMENT

Unit of Study: COMP5328 Machine Learning and Data Mining

Assignment name: Assignment 2

Tutorial time: PRAC11 **Tutor name:** MD. Mashud Rana, James Collins

DECLARATION
We the undersigned declare that we have read and understood the *University of Sydney Student Plagiarism: Coursework Policy and Procedure*, and except where specifically acknowledged, the work contained in this assignment/project is our own work, and has not been copied from other sources or been previously submitted for award or assessment.

We understand that understand that failure to comply with the *Student Plagiarism: Coursework Policy and Procedure* can lead to severe penalties as outlined under Chapter 8 of the *University of Sydney By-Law 1999* (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

We realise that we may be asked to identify those portions of the work contributed by each of us and required to demonstrate our individual knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

Project team members

Student name	Student ID	Participated	Agree to share	Signature
1. Ananda Aziz Hardjanto	470469051	[✓ Yes] / No	[✓ Yes] / No	[illegible]
2. Anthony Guo	470394395	[✓ Yes] / No	[✓ Yes] / No	[illegible]
3.		Yes / No	Yes / No	
4.		Yes / No	Yes / No	
5.		Yes / No	Yes / No	
6.		Yes / No	Yes / No	
7.		Yes / No	Yes / No	
8.		Yes / No	Yes / No	
9.		Yes / No	Yes / No	
10.		Yes / No	Yes / No	

Assignment 2 - Group: 73

Tutors: Md. Mashud Rana, James Collins

Group members: Anthony Guo (aguo0375), Ananda Aziz Hardjanto (ahar3481)

Abstract

Neural Network is one of the most advance and most used method in the machine learning community to create a classifier model. However, most classifier models created currently are created for datasets that are very large and informative. This report would like to explore the use and creation of classifier models that would perform relatively well on a small dataset such as CIFAR-100. This report would argue on how a simple neural network would be as useful when in comparison to a deep neural network on classifying datasets that are not too complicated. The three neural network algorithms used in this experiment are a base CNN, an AlexNet-base CNN, and a Residual Network with 14 layers (ResNet-14). In the result it shows that a simple baseline CNN is able to gain similar accuracy results as a more complex algorithm such as ResNet-14, as the more complex model would suffer from overfitting due to small dataset. This shows that simple CNN are still essentially useful, especially for beginners to learn and apply on datasets that are relatively small.

1. Introduction

The Canadian Institute for Advanced Research 100 classes (CIFAR-100) is a subset of the tiny images dataset which consists of almost 80 million 32 by 32 images which has recently been retracted due to offensive images in the dataset. However, the CIFAR-100 dataset remains clean and is a very common dataset used in computer vision research. The data contains 50,000 training samples of 100 classes, each class containing exactly 500 training examples. The data is used in research extensively due to the small amount of training examples per class, which leads to the problem of generalization. In the machine learning community, the benchmark for this dataset is currently held by EffNet-L2 (SAM) with an accuracy of 96.08%.

A simple classifier for this dataset with high accuracy would be beneficial due to the complexity of the dataset with relatively few training examples for each class. The main problem with designing classifiers and preparing the dataset for this problem is to figure out how to obtain enough information from the 500 images for each class and maintain a high level of generalization. We explore three simple convolutional neural networks (CNNs), a simple CNN, a modified LeNet-5 and ResNet-34. The models are trained on an augmented dataset to improve generalization and reduce training time. We use the Keras package from Tensorflow to build the CNNs and to augment the training data, this allows us to further generalize the problem which is important when there are limited training examples. We also attempted to run SVM on the dataset using the sklearn package however, due to sklearn's implementation of SVM which uses the 'one vs one' method for multiclass classification problems, the training time does not run in an acceptable timeframe, we do not include this model in this report.

Three different CNNs were chosen of varying depth and complexity to compare their classification performance on the CIFAR 100 dataset using metrics of accuracy, recall and precision. We aim to test models that are different in their architecture and observe the effect on the classification metrics. Due to the limited training data, we expect that generalization will be extremely difficult, in accordance an abundance of previous works both published and open source, we plan to use data augmentation and to train in batches to improve the generalization of our models and batch training will help keep training times manageable for our needs.

2. Related Work or Previous work

2.1 Base CNN

There are plenty of previous studies using different types of CNNs on the CIFAR 100 dataset. A common approach used is using a pretrained model and applying transfer learning as the CIFAR 100 dataset consists of limited samples. The base CNN used in this paper is not directly based off any previous work but is a very simple architecture and feeds the CIFAR 100 dataset through a shallow network.

The CNN model is based off and inspired from LeNet-5[1] but unlike the original model proposed by LeCun, consecutive convolutions layers exist in this architecture. In most architectures it is characteristic to see several convolutions before each pooling layer, this is to extract important information first via the convolutions before enhancing the features and decreasing the image using pooling. The problem with using too many pooling layers is that the image will not contain enough useful information to be classified by the neural network.

2.2 AlexNet

AlexNet is a CNN architecture proposed by Alex Krizhevsky in 2017, the original model has over 60 million parameters to train and performs well in image classification due to the depth of the mode[2]l. The model consists of 8 layers, the first 5 being convolutional layers and the remainder are fully connected layers of a neural network. Response-normalisation as described by Krizhenvsky aids in generalization of the model and is applied after the first and second convolution layer. Max pooling layers follow each response-normalisation and is applied to the last convolution layer. Each convolution and fully connected layer is accompanied by a ReLU activation function. Krizhenvsky concludes that the depth of the neural network is essential in its high performance, removing a single layer of convolution results in a measurable and significant 2% decrease from the top-1 performance of the network.

[1] LeCun Y, Botton L, Bengio Y, Haffner P. Gradient Based Learning Applied to Document Recognition [Internet]. Proc. Of the IEEE, 1998.

[2] Krizhevsky A, Sutskever I, Hinton, G. ImageNet Classification with Deep Convolutional Neural Networks [Internet]. 2012

In this paper we will attempt to run a modified 'quick and dirty' AlexNet with batch normalization in place of response normalization. We follow the general architecture, but the entire model is much simpler than the original, consisting of just under 4 million trainable parameters.

2.3 Residual Network

The Residual Network architecture was introduced by Kaiming He at 2015, in research to improving a deeper convolutional neural network[3]. In his paper, he explored that there is an extent on how deep a conventional convolutional neural network can go, before reaching a plateau of accuracy and efficiency. The paper showed that in a complex convolutional neural network, there is a problem of overfitting, as the model gets more complex with increasing number of convolutional layers. This have been showed with the better performance in models that have fewer layers.

In this paper he introduced a new method of extending the normal CNN, by using a 'residual block'. The aim of this residual block is to essentially to create a connecting of the output from a previous layer, into the input of the layer ahead. Due to the change in dimensions in between layers, the residual block would act as a channel to match the residual difference of the dimensions in between the layers[4]. This residual block would then be used to create a skip connection to double the filter depth, when the size of the feature map has changed, usually by half. This skip connection acts a solution the problem of vanishing gradient on complex and deeper CNN models[5].

In this paper, we will introduce the Residual Network neural network, with 14 layers on a different environment of dataset. On the paper introduced by Kaiming He, the ResNet model was trained on two different datasets, ImageNet, and Cifar-10. ImageNet consists of 1.28 million images, with 1,000 classes, giving the model the ability to train from a very large dataset in order to achieve a high accuracy. As for the second dataset, Cifar-10 consists of only 50 thousand images, with only 10 classes. In this scenario, the dataset given is significantly smaller, however, so is the classes it must train against. Giving the model approximately 5 thousand images per class to train against. As for the research in the paper, the ResNet-14 model will be used on the Cifar-100 dataset. This dataset consists of 50 thousand images, similar to the Cifar-10, however it has 100 classes, giving it only 500 images per class to train on. This gives the model less information on training.

3. Methods

[3] He K, Zhang X, Ren S, sun J. Deep Residual Learning for Image Recognition [Internet]. 2015 [cited 31 October 2021]. Available from: https://arxiv.org/pdf/1512.03385.pdf

[4] Shorten C. Introduction to ResNets [Internet]. Towards Data Science. 2019 [cited 31 October 2021]. Available from: https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4

[5] Shinde Y. How to code your ResNet from scratch in Tensorflow? - Analytics Vidhya [Internet]. Analytics Vidhya. 2021 [cited 31 October 2021]. Available from: https://www.analyticsvidhya.com/blog/2021/08/how-to-code-your-resnet-from-scratch-in-tensorflow/

3.1.1 Pre-processing: One hot encoding and image dimensionality

The labels given are numerical labels belonging to a class, for example images with the label apple were given the numerical value of 0 and there are 100 of such labels. To feed this data type into a neural network we one hot encode the class labels of the dataset such that each image was associated with a vector of length 100 with a single 1 in the whole vector corresponding to the index of its label. The images given to be classified are 32 by 32 colour images meaning that they have 3 channels for each pixel value, these are originally presented as a flat array with dimension 3072. This is reshaped to 3 by 32 by 32 representing each red, green, and blue colour intensity for each of its 32 by 32 pixels. Then transposed to give each image dimensions of 32 by 32 by 3.

3.1.2 Pre-processing: Train split and Data Augmentation

The data given contains 10000 testing examples already split from the training set. We further split the training set into train and validation in a ratio of 4:1 giving us a total of 40000 training and 10000 validation samples.

As mentioned previously, one of the challenges of this dataset is to generalize the training process to perform well on the validation and testing dataset. The main issue is that we do not have enough training examples to completely cover an entire class label. By using data augmentation we are able to flip images, shift vertically and horizontally to artificially create more training examples for each and every label using just a simple transformation of the existing image. We train the images using a generator, real time data augmentation is done at each epoch in batches to increase the generality of the model as well as reduce the size of the training to decrease training times.

3.2 Base CNN

3.2.1 Layers

The proposed method for the base model consists of four convolution layers with batch normalization between each convolution, this is followed by max pooling reducing the image by half and two fully connected layers. ReLU follows every convolution and every fully connected layer, a dropout of 0.25 follows the second batch normalization.

The kernel size is 3 by 3 throughout the convolutions and the number of filters are 128 in the first two layers, increasing by a factor of 2 in the third and fourth layer up to 512 filters. The final fully connected layer is followed by a dropout of 0.5 and then outputs to a 100-way SoftMax which gives us a probability distribution over the 100 class labels.

3.2.2 Advantages

The simplicity of this model makes it easy to interpret and also may be advantageous over more complex models as the simpler model is less prone to overfit to the training data. For a dataset with limited training samples, a shallow neural network or even traditional machine learning methods such as support vector machines or random forests performs better than a deep neural network. This is due to factors explored in previous works such as reaching poor local minimums (this has been addressed and solved in recency with models such as ResNet and AlexNet which will be explored later in this report).

3.3 Modified AlexNet

As described in 2.2, we will run a modified version of AlexNet, following the principles of the base architecture but replacing some normalization layers and in general simplifying the model. The dataset that AlexNet was originally trained on (ImageNet) had inputs of 224 x 224 x 3 whereas our image dimensions are only 32 x 32.

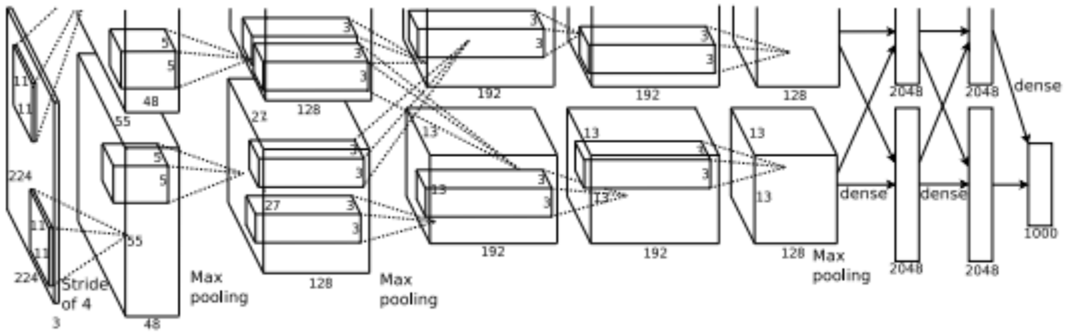

Figure 1. Original architecture of AlexNet in 2012, showing the separation of the two GPUs during learning and when they interact at which layer (Krizhevsky 2012).

We do not change the size of the kernels in this experiment. Another difference is that Krizhevsky utilizes two GPUs, separating the tasks between the computing units and only allowing them to communicate at certain layers, as we run our code off one GPU and do not separate out tasks our model may not perform like the original.

3.3.1 Layers

The input is fed into a convolution layer with 96 kernels of 11 by 11 and a stride of 4 by 4, this is followed by the ReLU activation function and batch normalization and max pooling with a size of 3 by 3. This layer is repeated with 256 kernels of size of 5 by 5 and a stride of 1 by 1. Then AlexNet has 3 consecutive convolution layers the first two identical being of 384 kernels of 3 by 3 and a stride of 1 by 1 and the last convolution having 256 kernels of 1 by 1 and a stride of 1 by 1. There is a final max pooling identical to the other pools. There are then two fully connected layers which include dropouts of 0.5 to increase generalization and finally the model outputs to SoftMax. We follow the method of the paper and use stochastic gradient descent as the optimizer with a learning rate of 0.2 and momentum of 0.9.

3.3.2 Advantages

The architecture of AlexNet was first of its kind, utlilising consecutive convolution layers and the ReLU activation function to remove the vanishing gradient problem. At the time, the dropout layers were essential in addressing the generalization error of the neural network by letting the optimizer escape from local minima, however the dropout causes the model to require more iterations to converge. At this current time (3rd November 2021) the original paper has received 72209 citations of which 9044 are highly influential citations, highlighting the importance of this CNN in the machine learning community.

3.4 ResNet-14

As mentioned on the previous works, the Residual Network is a deep neural network that introduces the skip connection in order to reduce the vanishing gradient problem in a deep neural network. This skip connections would be used to skip the training of certain layers, which would then be used to connect with the output of the current block, to be used as an input of blocks ahead.

3.4.1 Identity and Convolutional Block

In Residual Network model, there are two types of blocks that help the skip connection function mentioned earlier. There is Identity block and Convolutional block. These two blocks function similarly, with the purpose to add the residue to the output, for the following blocks input.

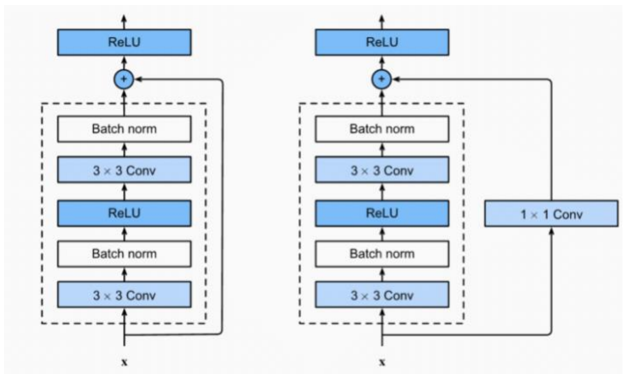

Figure 2. Left: Identity Block. Right: Convolutional block. Reference: https://www.analyticsvidhya.com/blog/2021/08/how-to-code-your-resnet-from-scratch-in-tensorflow/

The Identity Block in Residual Network would take the residue, and directly add that to the output, to be used for the following input. Whereas in the Convolutional block of the Residual Network, the residue would undergo a convolutional layer and a batch normalization, which would then be added to the output.

The Identity block consists of two conv2D layers, where each of the layer is followed by batch normalization layer and an activation 'relu' layer in between. Following this, the residue is then added without any convolution, which later then be added all together to an activation 'relu' layer. All the convolutional layers would have a kernel size of 3 with padding 'same' to ensure a maintained shape of the input.

In the Convolution Block, the layers are the same with the identity block. However, on the layer where the residue is added, a conv2D is used to introduce a convolutional layer to the residue.

3.4.2 Architecture

To initialize the model, the first layer introduced in the model would be an input layer, using a zero padding 2D layer to introduce the shape of the images which is 32x32x3.

In the ResNet-14, the identity and convolutional blocks can be combined to 5 major blocks. The first block is the only one that consists only one convolutional 2D layer, which have a filter of 64. The convolutional layer is then followed by batch normalization. The purpose of this is to help normalizes the input. Before introducing the following convolutional blocks, a maxpool 2D layer is also introduced here.

On the second block, it has two sub-blocks. Since the convolutional block is not needed yet, the two sub-block will only go through the identity blocks, two times. As for the third block, it will consist of 2 sub-blocks. The 2 sub-blocks being used here will consists of one convolutional block to start, followed by an identity block.

As for the fourth block, it will have 2 sub-block layers as well. Similar to the previous block, in here the sub-block would consists of only 1 convolutional sub-block layer as well and one identity sub-block layer.

As for the filters for the conv2D in these blocks, there is an increasing pattern. For each increase in block, the filter would be a double of the filter from the previous block, starting from the second block with a filter of 64. The third block would then have convolution and identity blocks with conv2D layers with 128 filter. However, in order to adjust to the small size of the images, instead of having deeper layers, a larger filter is used on the last block with 512 layers.

To finalize the model, an average pooling layer is used, prior to the flatten layer which is to create a fully connected network. Two dense layers are used, with 512 units for the first one, and the second one having a unit with the same amount of the classes available which in this case, is 100 classes.

In addition to the actual model, upon fitting the model for training, a function to control the learning rate is introduced, instead of keeping a constant learning rate when in comparison to the other model. The learning rate for the model on training would start at 0.001 and would decrease by a factor of 0.1 when the validation loss would stop improving for 5 consecutive epochs. The minimum learning rate it would go to would be 0.00001.

3.3.3 ResNet-14 Advantages

From the ResNet-14 model explained above, it shows how the model would use residuals from the input, and directly introduce it into the output, to create a generalization in the model to reduce the vanishing gradient, even though the model is increasing in complexity and layers.

Theoretically, the residual input would allow more information from the inputs of the earlier layers, to pass on through to the deeper network, to maintain the distribution of information for the gradient. By introducing these residuals information from the shallow layer to the deeper layer, more information should be able to be retained for the training, which would give a more accurate model with less test errors.

The choice of the ResNet-14 algorithm here instead of other versions for Residual Networks (ResNet-50, ResNet-101, etc), is purely due to the simplicity of the dataset. Given the dataset only consists of 50 thousand images to train 100 classes, a Residual Network with very deep layers may suffer from overfitting as there is not enough data to train the model effectively and would reflect poorly on the test set.

4. Experiments

In this section, all three models will be discussed in terms of evaluating its performance in training and testing. Each model undergoes different types of tuning, to ensure that the model is tuned to reach its highest potential in terms of accuracy and predicting the correct labels.

All three model codes that are being compared for this experiment is run on the same environment to create consistency in order to be able to evaluate the performance of each model without any bias. Each model is run on a local computer with RTX 3080 GPU on the VSCODE software. In order to speed up the process of training the data, TensorFlow is installed to run on the GPU using CUDA. This method is proven to improve the runtime of each model significantly when in comparison to running it on a standard system, such as Google Colab. As an initial test comparison, the base CNN model was run on the Google Colab system. Upon running, the model manages to train with approximately 40 seconds per epoch, taking over 1 hour to complete training. When compared to running the same model on the local machine, each epoch was completed in just 12 seconds, finishing training in only 20 minutes. Based on this trail run, it can be decided that running the model on the local machine would be more efficient. Each model's runtime will be discussed further for comparison below.

4.1 Base CNN Model

4.1.1 Tuning the model

The Base CNN model is the baseline model for this report, as it is a simple convolutional neural network model that is not as deep as the state-of-the-art models, as described in the methods above. To achieve the maximum performance from the simple model, it was tuned based on the performance towards the validation set by observing the validation loss and accuracy. The variables that were tuned in this model was the filter sizes for each convolutional layer, placement of the maxpooling layer as well as the probability of the dropout layers.

Upon repetition of training, it was found that the architecture of the tuned variables mentioned on the method above, produced the highest possible accuracy for this model. The chosen architecture produces 2,202,468 total parameters, showing the simplicity of the model.

The base CNN model is compiled using the categorical cross entropy loss as this is a classification model. The categorical cross entropy loss is used when there are more than one label classes, which in this case there are 100 classes to label the image on. In addition, this loss function is used on datasets in where the labels are in the form of a one hot matrix. This was done towards the labels when the dataset was split and the function to_categorical was used to transform the labels to a binary class matrix. As for the optimizer used, the 'Adam' optimizer is used as it is shown to be the one of best optimizer for neural networks. As for the metrics used to measure the performance, accuracy metric is used to.

4.1.2 Base CNN Training

Upon training the model, two sets of data, the train set, and validation set is used. Both of these set of data have been pre-processed using image augmentation to generate altered images. The purpose of this image augmentation is to generate more variety on the images for the model to train on, giving it more information to retain by adjusting the rotation, width shift, and horizontal flip on the training images. The model is then fitted to train by running 100 epochs, with a batch size of 32. The model took an approximate of 20 minutes to train, with approximately 12 seconds to train per epoch.

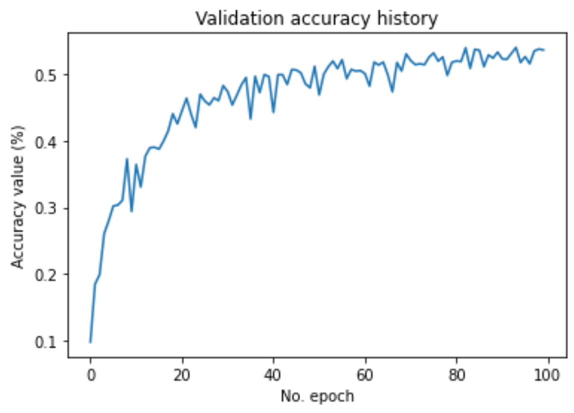

Figure 3: Validation Accuracy of Base CNN

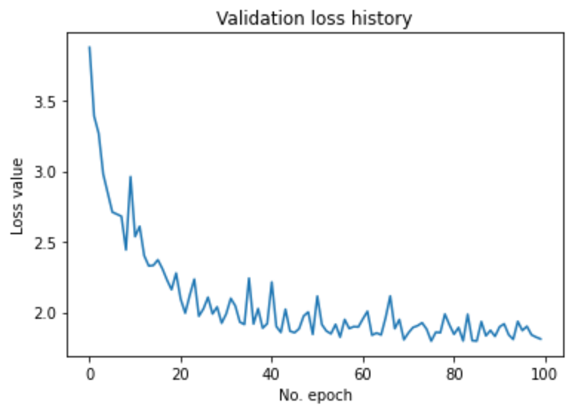

Figure 4: Validation loss of Base CNN

From figure 3 and figure 4, it shows the result of model on the validation set upon training. The base CNN model manages to achieve a validation accuracy of 53.6% and a validation loss of 1.81. In figure 2 it shows that the model trained and improved between the first epoch to epoch 30, where it starts to plateau and only improve slightly.

The base CNN model has a training accuracy of 61.9% and training loss of 1.35. The difference between the training accuracy and the validation accuracy suggests that the base CNN model suffers from slight overfitting. This shows that the model was able to learn and fit the labels on the train set too well, that it overfits when introduced to the validation set, causing it to decrease its accuracy on assigning the correct label to the images.

4.1.3 Base CNN Results

Base CNN on Test Set			
Precision	*Recall*	*F1-Score*	*Accuracy*
0.55	0.54	0.54	0.5443

Table 1: Base CNN results on Test set

As it can be seen from table 1, the base CNN model managed to achieve a result better than expected as it was a very simple model architecture. The model managed to get a 54.43% accuracy on the test set. This accuracy achieved on the test set is similar to the accuracy that the model was able to achieve on the validation set, showing that the model is able to perform relatively consistent as the amount of data is increased. The similarity from the precision, recall, F1-score and accuracy is as expected due to the balanced dataset as each class have the same amount of images to train and test upon.

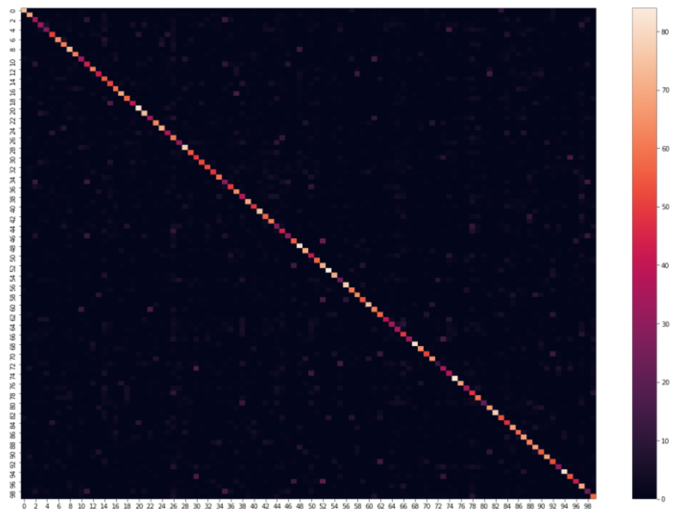
Figure 5: Base CNN confusion matrix

The confusion matrix on figure 5 is a clear representation on how the model managed to correctly label each image to its proper label. The axis on the confusion matrix is between the image and the label. The clear line shows that on majority of the occasion, the model managed to label the image correctly, with the model performing better on some images in comparison to others. An example of this is shown below on figure 6, showing the model correctly label the pixelated image of a lion, as a lion.

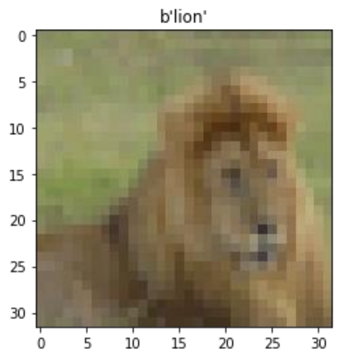

Figure 6: Model labelling a picture of Lion

4.2 AlexNet Base CNN

4.2.1 Tuning the Model

With the AlexNet Base CNN model, there are not many variables on the architecture that can be tuned as the model is based on a published architecture, and it was best to keep it as similar as possible, in order to be able to create a comparison. The method we use to tune the model is the same as the previous model, using validation set of data to measure the performance of the model, and manually tune the model with repetition whilst changing certain variables.

As the chosen AlexNet model is a simple model with only 5 convolutional layers, there was no change on the amount and level of layers. The kernel size for the input layer here is also kept the same, as the one used in the published article, as it assists on showing how the chosen dataset type (different in image size), may affect the performance of the model.

One of the variables that is changed from the base AlexNet model, was the learning rate of the stochastic gradient descent optimizer upon compiling the model. In the publish article and creator of the AlexNet model, a changing learning rate during the training was used starting from 0.01 and decreasing by a division rate of 10. As for our version of AlexNet, a constant learning rate was used, at the value of 0.02 which shows to give the best performance and result for the model with the given dataset. As for the loss and metrics used on compiling the model, the same were used as the base model, with loss function of categorical cross entropy and the accuracy metric.

4.2.2 AlexNet Base Training

Similar to the previous base CNN model, for the AlexNet based neural network, the model is trained using the same train and validation set of the dataset. The two set of the data have undergone the same pre-processing method as well for this model. To train the model, it is fitted to train on the same 100 epochs and batch size of 32 as well. As for this model, the runtime for each epoch is approximately 13 seconds, giving a total training time of approximately 22 minutes.

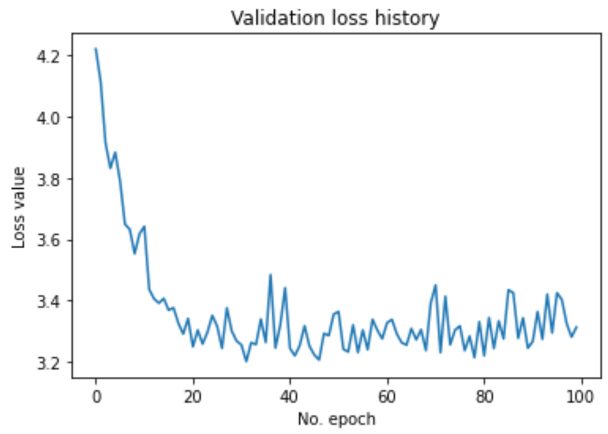

Figure 7: AlexNet Based model validation loss

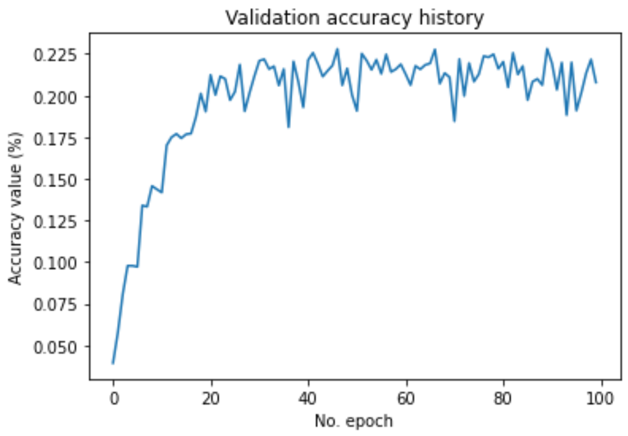

Figure 8: AlexNet Based model validation accuracy

The performance of the model upon the validation set can be clearly summed up on figure 7 and 8. The figure shows how the model was only able to achieve a low validation accuracy of 20.79% on its final epoch. However, it shows on the graph that the model has managed to achieve accuracy as high as 22% on other epochs. This depicts the unstable performance of the model on training as the accuracy of each increasing epoch would spike upwards and downwards. The same can be said on the validation loss, with a high validation loss on the final epoch of 3.31.

However, in this model, the training accuracy that it managed to achieve is only 17.9%, a significant amount lower than the validation accuracy. A possible explanation for the lower training accuracy is due to the two dropout function being used in the model. Since dropout is a regularization technique, it creates a loss of information for certain images/samples. Given that the images being used here is smaller with only 32x32 and only 500 images per class to train on, compared to the very large dataset ImageNet used to develop the model, this loss of information may have been significantly too severe for the model to be able to reconstruct the images properly. This shows on the large training loss 3.54.

4.2.3 AlexNet Base Results

AlexNet based CNN on Test Set			
Precision	*Recall*	*F1-Score*	*Accuracy*
0.21	0.21	0.18	0.214

Table 2:AlexNet based CNN results on Test set

The table above shows the result of the AlexNet based model performance on the dataset's test set. The model was only able to achieve 21.4% of accuracy, which is significantly less when in comparison to the base CNN model used previously. This poor performance of the model may be due to the small dataset being used in this report. The AlexNet model was initially constructed to perform on a large dataset such as ImageNet with 224 x 224 size images. The model used here is not designed to perform on a dataset with image size of only 32 x 32. This shows on the input convolutional layer, where the kernel is left as 11 x 11. However, this result is as expected since the actual AlexNet model on ImageNet was only able to achieve an accuracy of 37.5%, making it a low performing model.

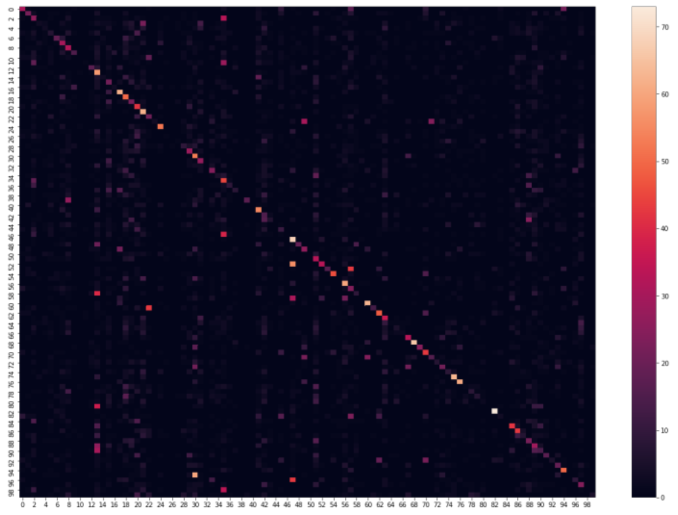
Figure 9:AlexNet based confusion matrix

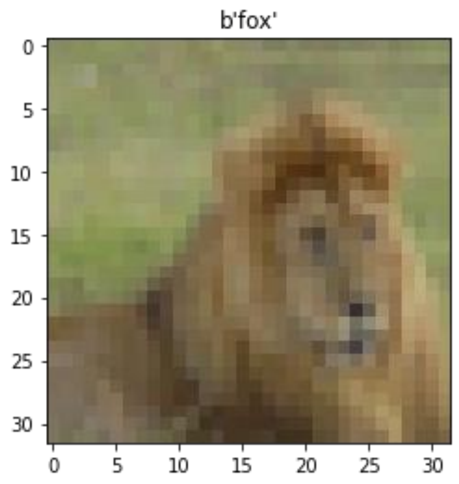

Figure 10: AlexNet model labelling a picture of lion as fox

The above figures depict a clearer explanation on the poor performance of the AlexNet based model. As it can be clearly seen on the confusion matrix figure 9, the model suffers from mislabeling a significant number of images. An example of this is shown next to it on figure 20. The model manages to label a picture of lion, as fox. This shows the model fails to correctly label images with similar features. In this case, fox and lions when analyzed on a pixel-by-pixel level, may have similar facial structure. The model was not able to pick up and differentiate on this similarity, and assumed that it is a feature from the same image.

4.3 ResNet-14

4.3.1 Tuning the Model

The ResNet model is a model with plenty of variables and layers in its architecture that can be tuned in order for the model to perform well on the dataset being used. The number following the ResNet represents the number of convolution layers being used in the model. A further explanation of the layers has been explained on the method section previously. The famous Residual Networks are ResNet-34, ResNet-50 and many deeper models. Due to the simplicity of the dataset being used, a simple version of the Residual Network has been used with only 14 main layers. Hence, the name ResNet-14.

The tuning of this model is done in a similar way as the previous models, by using both training and validation set upon training, and adjust the variables accordingly to improve the model. The first variable being tuned in this model is the amount of layers being used. A model of ResNet-34 and ResNet-20 was tried on the dataset, however with the deeper model, it shows that the training tends to overfit significantly, where the models was only able to achieve around 49%. The layers were reduced to ResNet-14 where it shows to have given the best result. Another advantage from using less layers is that there is a significant decrease in running time.

In addition, the filter for the model is adjusted from the actual residual network design. In a typical residual network model, each filter increase would be a double of the previous filter. However, in this model, it has been tuned where there is a direct increase in filter from 128 to 512. By skipping the filter amount of 256, it has improved the performance of the model.

The residual network model in general suffers a significant overfitting due to the complexity of the model. In order to adjust the model to reduce overfitting, a layer of dropout with value of 0.2 have been introduced to create a generalization layer after the last identity block. Different value for the dropout probability has been tried, with 0.2, 0.5, 0.8 and 0.9. In addition, additional dropout layers were introduced on different levels of the model, however it seems to have decreased the performance of the model.

The last parameter being tuned for the model is on how the learning rate is being used upon training the model. Different from the previous models where the learning rate remains constant throughout training, in the ResNet-14 model, a function to adjust the learning rate is introduced as mentioned on the methods. This helps stabilize the model even further as the model learns more in training.

4.3.2 ResNet-14 Model Training

The ResNet-14 model is trained using the same dataset as the previous models, that are split into training and validation sets. However, as for the ResNet-14 model, the image augmentation for the training set is only limited to horizontal flip, differ from the more complex image augmentation used for the previous models. The model is fitted to train on the same amount of epochs at 100 and batch size of 32. With those given variables, the ResNet-14 model is very complex that it increased the runtime significantly. Each epoch takes a range of 30 to 33 seconds to run, resulting in a total time of 52 minutes to train the model entirely.

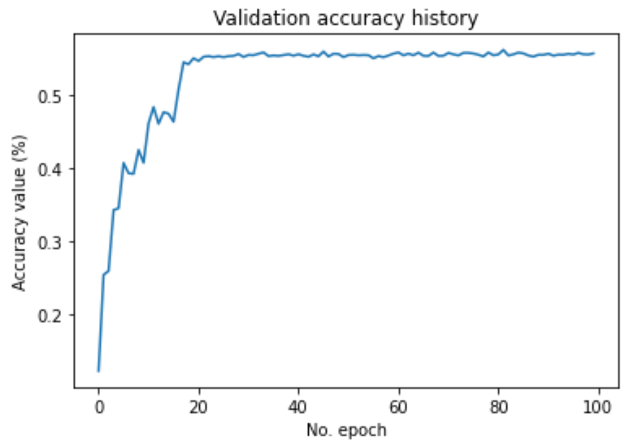

Figure11: ResNet-14 model validation accuracy

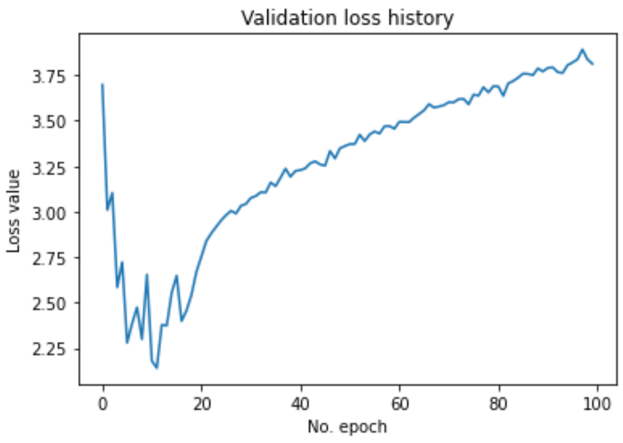

Figure 12: ResNet14 model validation loss

The ResNet-14 manage to achieve the highest level of accuracy on validation, with a maximum accuracy of 56.1% reached on epoch 82. As it can be seen from figure 11, it shows how the decreasing learning rate impacts the model training. With each reduction of learning rate, the model shows to stabilize and plateau. It can be seen how the model manage to train and learn in a fast rate in the beginning, with reaching above 50% validation accuracy on the first 20 epochs.

Upon training the model, it was visible that the model suffers from overfitting significantly. The ResNet-14 model managed to achieve a 99% training accuracy. However, only a validation accuracy of 56.1%. This can be seen on the validation loss graph in figure 12. In this graph, it shows how the validation loss decrease in the beginning, as both validation and training accuracy would increase in a parallel manner. However, as it approaches epoch 20, the validation accuracy stops improving significantly, as where the training accuracy kept on improving. This pictures in the significant increase of validation loss in figure 12.

An attempt to solve this overfitting by using regularization, addition of dropout layers and change in learning rates was attempted. However, by solving the overfitting, the model shows to have decrease the validation accuracy as well. Therefore, it was decided to create a model that suffers from overfitting but was able to generate a well enough accuracy.

4.3.3 ResNet-14 Results

Resnet-14 model on Test Set			
Precision	*Recall*	*F1-Score*	*Accuracy*
0.57	0.56	0.57	0.565

Table 3: ResNet-14 model on Test Set

As shown on table 3, the ResNet-14 have managed to achieve a high accuracy of 56.5% accuracy, the highest out of the three model. The model manages to achieve a similar accuracy on the test set, when in comparison to the validation set. This shows the consistent performance of the model, as the amount of data is increased. The model performed as expected, with the given simple dataset to train on.

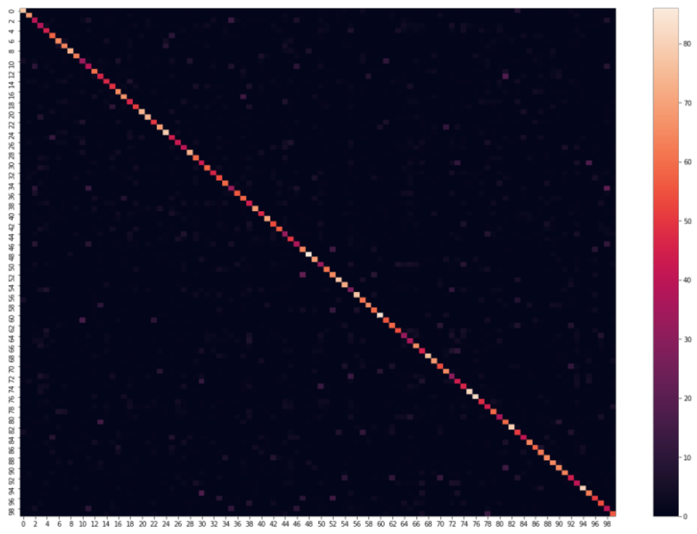

Figure 13:ResNet-14 confusion matrix

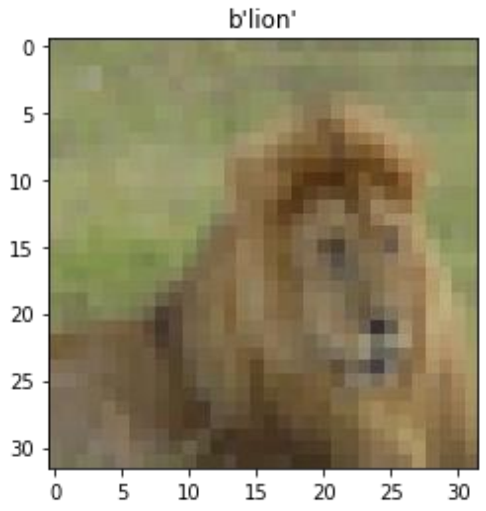

Figure 14: ResNet14 labelling a picture of lion

The confusion matrix on figure 13 shows how the model is able to correctly label the images on a repetitive manner. It clearly shows a line on the respective image and label, where there is very little errors on each label being given. This performance can be seen on the figure 14 next to it, where it was able to correctly label a picture of a lion.

4.4 Comparison and Evaluation

	Accuracy			*Training Time*
	Training	*Validation*	*Test*	
Base CNN	61.9%	53.6%	54.4%	20 m
AlexNet-Based CNN	17.9%	22.8%	21.4%	22 m
ResNet-14	99.8%	56.1%	56.5%	52 m

Table 4: Model Performance Comparison

From the table above, it can be seen how each model perform on its training, validation and test set along with its runtime. It can be seen, as previously discussed, that the base CNN model and ResNet-14 model suffers from a significant overfitting, with it being greater in the ResNet-14. From the table we can see that the base CNN and the ResNet-14 has managed to achieve a similar result, with a difference of 2% where the ResNet-14 performed better. This higher accuracy is due to a deeper neural network, with the use of residuals to avoid vanishing gradient. This allows the model to retain more information as it uses more parameters to train the model. However, the training time for the ResNet-14 is more than twice longer with the local computer environment. This shows a significant tradeoff for a very long computing time, which in return only gives marginal improvement of a simpler neural network.

As it can be shown and previously discussed, even the simple base model in here suffers from overfitting. Various steps and tuning for each model have been attempted to solve for this issue. However, as it can be seen that the overfitting exists on the simplest model and even more significant on the more complex model, the models being implemented here can be improved by being given more dataset to train form. By introducing a larger dataset, the model would have more information to train with, giving a more accurate prediction. This solution would be more advantageous for the more complex model, such as the ResNet-14 model, as it was able to achieve 99.8% training accuracy.

The AlexNet-based CNN model here shows how neural networks that are designed for a different dataset will not be able to directly be re-implemented on a different dataset with significant property difference. In order for this to work, further changes to the model needed to be made in order to adjust the values to the smaller image size, such as changing the input convolutional layer kernel size to a 3 x 3. Another solution for this would be to train the AlexNet model on its intended dataset such as ImageNet and use transfer learning to introduce it to the CIFAR-100 dataset for classification.

The three models chosen shows the performance difference and similarity of models with different levels of complexity and depth neural network. On the given CIFAR-100 dataset, it can be seen that the simple model of base CNN is able to keep up in accuracy with a complex model of the ResNet-14, with a significantly less computing power and time needed.

In this report, the best model out of the three can be clearly be decided as the ResNet-14 with the highest accuracy. However, as due to the small difference between ResNet-14 and base CNN and significant difference in training time, it can be argued that the base CNN is a close second, especially with a limited computing power.

5. Conclusion

The three models used in the dataset manages to show the difference in performance of models with different complexity. As expected, the most complex model ResNet-14 managed to achieve the best accuracy out of the three model. However, surprisingly the simple base model was able to achieve a similar accuracy that is not far off from the best model. The results from this experiment shows that a deeper neural network will perform better in classification. In addition, the help of the residual networks, it helps the deeper neural network to perform better by reducing the vanishing gradient problem.

As it is derived from the two out of the three model, the main issue on running neural network models on the dataset is that the size of the dataset is causing the models to overfit as it does not have enough information to train on. Increasing the dataset set, by giving the model more images per class label would be a significant help for future improvements to help reduce overfitting and increase accuracy. Another approach for future work that can be used to help improve the performance is to use a different metric to measure the model's performance. Since there is 100 class to label, using the accuracy metric where it measures the amount of time it correctly labels an image, which means placing the correct 1 label out of 100 is too difficult. A better metric from Keras such as TopKcategoricalAccuracy, where it measures the K specified number of top elements being correctly predicted is a fairer approach to measure accuracy in a dataset with large number of classes.

References

1. Krizhevsky A, Sutskever I, Hinton, G. ImageNet Classification with Deep Convolutional Neural Networks [Internet]. 2012 Available from: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

2. He K, Zhang X, Ren S, sun J. Deep Residual Learning for Image Recognition [Internet]. 2015 [cited 31 October 2021]. Available from: https://arxiv.org/pdf/1512.03385.pdf

3. Jay P. Understanding and Implementing Architectures of ResNet and ResNeXt for state-of-the-art Image… [Internet]. Medium. 2018 [cited 5 November 2021]. Available from: https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624

4. Rosebrock A. Keras Conv2D and Convolutional Layers - PyImageSearch [Internet]. PyImageSearch. 2018 [cited 5 November 2021]. Available from: https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/

5. Shorten C. Introduction to ResNets [Internet]. Towards Data Science. 2019 [cited 31 October 2021]. Available from: https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4
6. Shinde Y. How to code your ResNet from scratch in Tensorflow? - Analytics Vidhya [Internet]. Analytics Vidhya. 2021 [cited 31 October 2021]. Available from: https://www.analyticsvidhya.com/blog/2021/08/how-to-code-your-resnet-from-scratch-in-tensorflow
7. Team K. Keras documentation: Conv2D layer [Internet]. Keras.io. 2021 [cited 5 November 2021]. Available from: https://keras.io/api/layers/convolution_layers/convolution2d/
8. LeCun Y, Botton L, Bengio Y, Haffner P. Gradient Based Learning Applied to Document Recognition [Internet]. Proc. Of the IEEE, 1998. Available from: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

Appendix

- Running the code:
  1. Download the dataset from the website https://www.cs.toronto.edu/~kriz/cifar.html
     a. Scroll down on the website and find the CIFAR-100 python version and download the file.
  2. Place the downloaded dataset, the three ipynb file and the ResNet14_model.h5 file on the same folder.

3. Each ipynb represents different algorithms, that takes the same procedure to run (except for the Best_algorithm_RESNET14.ipynb as it has saved trained model and can skip training).
4. Choose an algorithm ipynb file to run, and open it.
5. If the code is run on colab, mount the driver by un-commenting the second code block (as instructed on the code file). If it is run on a local machine, skip this step.
6. On the third code block, replace the path the path of the folder where the dataset, ResNet14_model.h5 file and all the ipynb is placed.
7. Run every code under the pre-processing heading, with the data augmentation as its last code.
8. To run the main algorithm:
    a. If the chosen file is the Best_algorithm_RESNET14.ipynb, a .h5 file have been prepared to skip the training. In this algorithm, to save time, it is possible to skip down to the code block under "Loading the saved model" and run it to load the .h5 file, and skip the training code chunk. After this, all the remainder code blocks should be run to get the results (except for the codes of plotting the graph, as this needs the history of the training to run).
    b. If the chosen file is the other ipynb algorithm file, run every code chunk until the end individually.

- Training accuracy and loss of each algorithm:

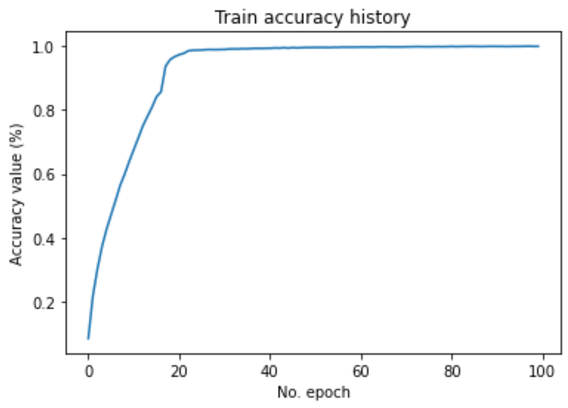

Figure 15: Train accuracy of ResNet-14

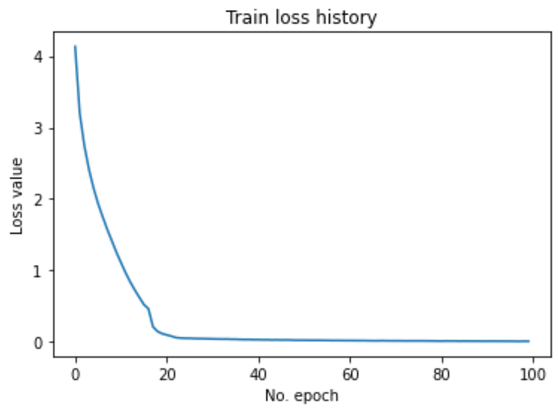

Figure 16: Train loss of ResNet-14

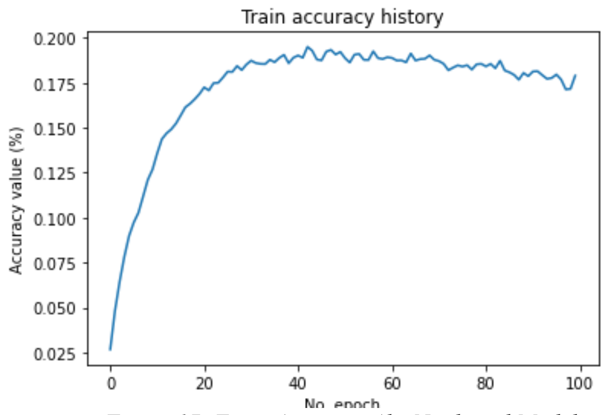

Figure 17: Train Accuracy AlexNet-based Model

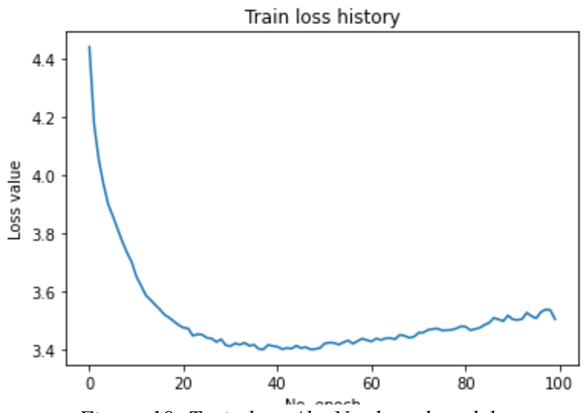

Figure 18: Train loss AlexNet-based model

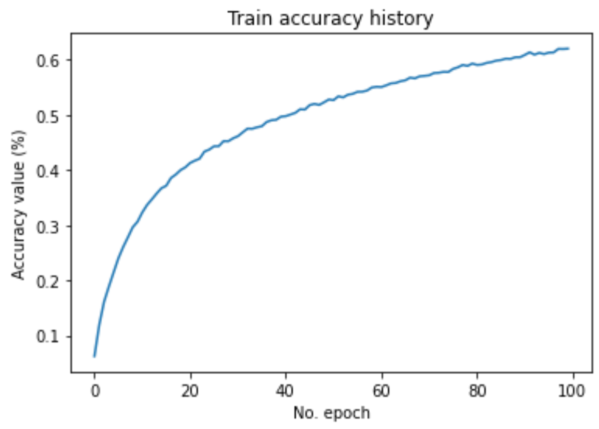

Figure 19: Base CNN train accuracy

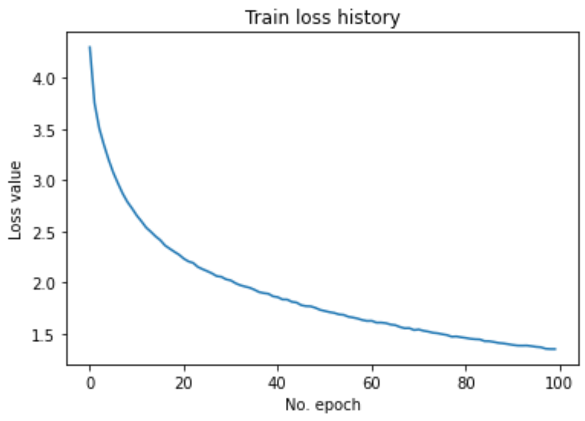

Figure 20: Base CNN train loss

- Results:

	Accuracy			Loss		*Training Time*
	Training	*Validation*	*Test*	*Training*	*Validation*	
Base CNN	61.9%	53.6%	54.4%	1.355	1.812	20 m
AlexNet-Based CNN	17.9%	22.8%	21.4%	3.537	3.313	22 m
ResNet-14	99.8%	56.1%	56.5%	0.01	3.813	52 m