CSCE 479/879 Homework 1 [Image Classification]

Beichen Zhang, Yinglu Jia, Bin Ma

February 27, 2021

Abstract

In this report, we built different neural networks (NNs) to solve image classification problems. The model architectures and hyperparameter training are the two most essential components in applying the deep-learning technique to the industry and research. Therefore, this study employed two datasets: FMNIST and CIFAR-100 to examine the impacts of different model structures and hyperparameters on image classification. In the first part of the report, we trained and tested fully-connected NNs on FMNIST to investigate the performances of models with different number of layers and neurons. The best performed model achieved an accuracy of 88% on the test dataset. In the second part, a self-defined convolutional neural network (CNN) structure and the residual network (ResNet) were trained and tested on CIFAR-100. We tuned different architectures and hyperparameters to improve the models' performance. As a result, our model on CIFAR-100 dataset achieved 42% test accuracy, which is not affordable to a real application. The primary reason of the poor performance could be the serious over-fitting issue that limits the possibility of using more complex models.

Part I
Image Classification on FMNIST

1 Introduction

Fashion-MNIST is a dataset of Zalando's article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28×28 grayscale image, associated with a label from 10 classes1. Fashion-MNIST is intended to serve as a direct drop-in replacement of the original MNIST dataset for benchmarking machine learning algorithms.

In this report, we use fully-connected neural networks (NNs) to solve image classification problems on datasets of Fashion MNIST. We tired different architectures and parameters to improve the models' performance. As a result, our model on Fashion MNIST classification is considerable solid, achieving a accuracy of 88%.

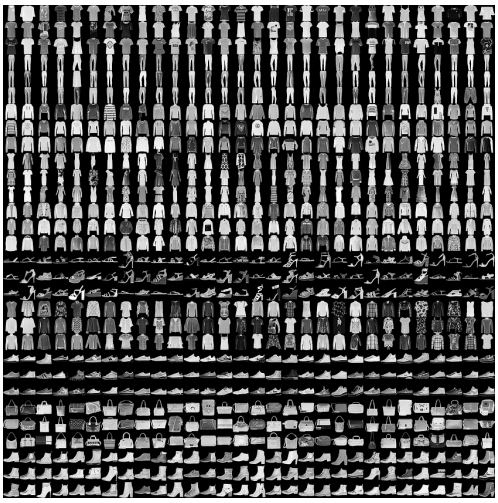


Figure 1: Fashion-MNIST

2 Problem Description

The first one is to classify images provided in dataset of Fashion-MNIST. The dataset contains 70,000 grayscale images, with 60,000 as training images and 10,000 as test images. Each grayscale image consists of 784 pixels with 28*28 as dimension. In each pixel, intensity described with an integer, range from 0 to 255. All images had been classified into ten different classes. The labels of the classes are integers, from 0 to 9. Each class represents one type of fashion objects, as listed in Figure 2.

Figure 2: Labels of 10 classes in Fashion-MNIST

Label Value	0	1	2	3	4	5	6	7	8	9
Meaning	T-shirt/ top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot

3 Approaches

The dataset will be split into training data and test data by 7:3. We are going to apllied different architectures with various number of neurons and layers. Also, different learning rate will be applied too.

Table 1: Tested fully connected NNs.

models	num of neurons of each layer	batch size	num of epoch
model 1	10/10	200	500
model 2	20/10/10	200	500
model 3	200/20/10/10	200	500
model 4	400/10/10	200	500
model 5	600/10/10	200	500
model 6	800/10/10	200	500

4 Experimental Setup

4.1 Architecture

Fully-connected NNs were adopted in our classification task. In fully-connected NNs, each neuron in one layer is connected to all neurons in the next layer, and each connection has its own weight. The "fully-connectedness" of these networks makes them prone to over-fitting data. So, regularization is usually required. Typical ways of regularization are to include a complexity penalty on the weights in the loss function. In order to do experiment on how the architectures are influencing the performance. We designed six different fully connected NNs 1. All the hidden layes were fully connect with ReLU, and the activation function of the output layer is set to softmax.

In order to discuss how the number of layers will affect the training, we designed model 1, model 2 and model 3. Model 4, model 5 and model 6 will reflect the number of neurons impact. Also, we applied both with early stop and without early stop on each model. Learning rate is changed among 0.5, 0.01 and 0.001, when we are training model 2 and model 3.

5 Experimental Results

Generally, the test accuracy is shown in the Table 2.

5.1 The number of layers

The first model only have one 10-neuron hidden layer, and we add on 20-neuron layer on top of that. Furthermore, we add another 200-neuron layer on the second model.

In this case, model 2 displays the best result on training data (Figure 4).This indicates that it is not "the more complex of the model, the better result there will be". Actually, model 3 which is the most complex performs worst in this case.

Table 2: Test Accuracy

models	early stopping	learning rate	test accuracy
model 1	yes	0.001	0.81
model 1	no	0.001	0.86
model 2	yes	0.001	0.70
model 2	no	0.001	0.86
model 2	yes	0.01	0.80
model 2	no	0.01	0.76
model 2	yes	0.5	0.1
model 2	no	0.5	0.09
model 3	yes	0.001	0.88
model 3	no	0.001	0.79
model 3	yes	0.01	0.33
model 3	no	0.01	0.1
model 3	yes	0.5	0.1
model 3	no	0.5	0.1
model 4	yes	0.001	0.87
model 4	no	0.001	0.87
model 5	yes	0.001	0.88
model 5	no	0.001	0.88
model 6	yes	0.001	0.88
model 6	no	0.001	0.82

Figure 3: model 1 early stopping vs no early stopping

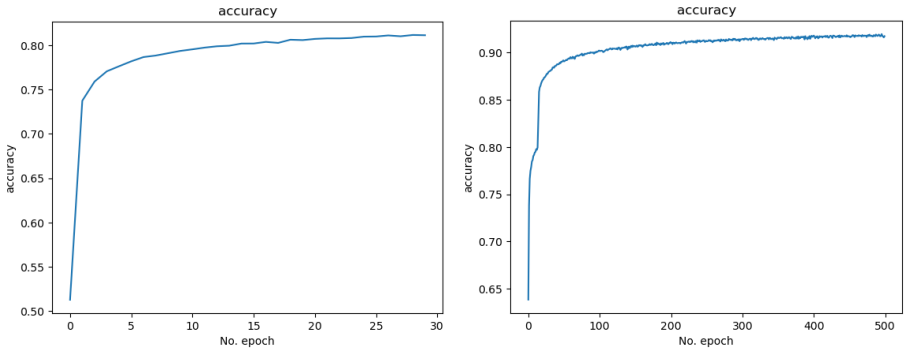
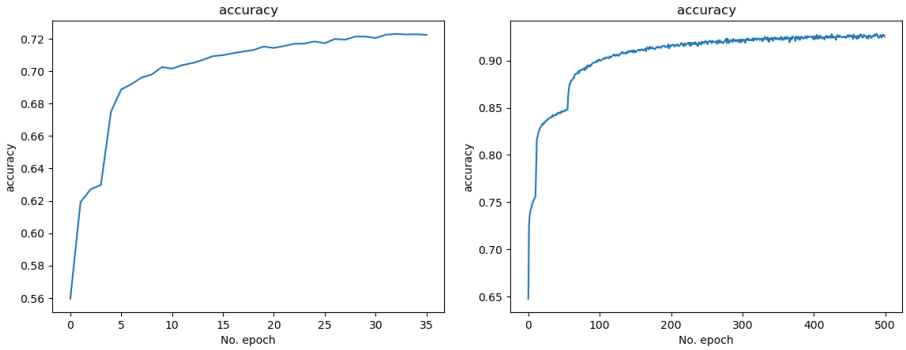


Figure 4: model 2 early stopping vs no early stopping



5.2 Early stopping

We can also get the information of early stopping in each model. validation accuracy while training is the monitor and $min_delta = 0.001$. In order to avoid the local optimum, we set a patience = 10. In this case, the training with early stopping only train the data with less than one hundred epoch, while the total epoch number is set to be 500. For model 1 and 2, it shows that as the training epoch increases, the training accuracy increase too. But the improvement stay very trivial. Considering the time and other resources, early stopping will be favored. For model 3, as the training epoch increases to a large number, the performance is getting worse. Here, the early stopping is necessary.

5.2.1 Learning rate

We also tested different learning rate with the number of 0.001, 0.01 and 0.5 on model 2 and 3.

Figure 6 and 7 show the great difference on different learning rates, especially when the earlystopping was not applied. The right picture in Figure 6 indicates

Figure 5: model 3 early stopping vs no early stopping

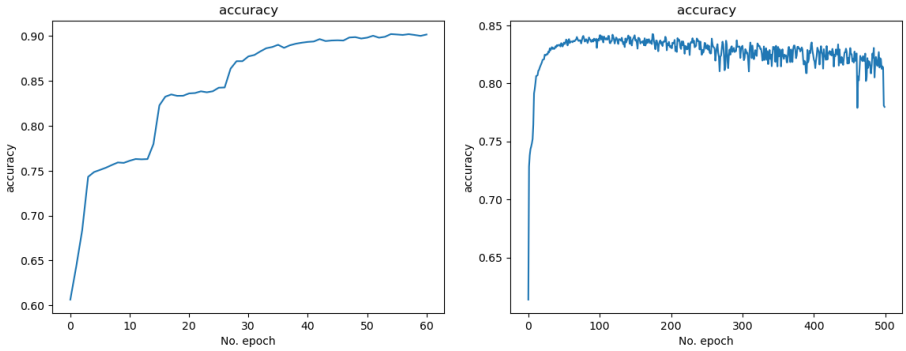
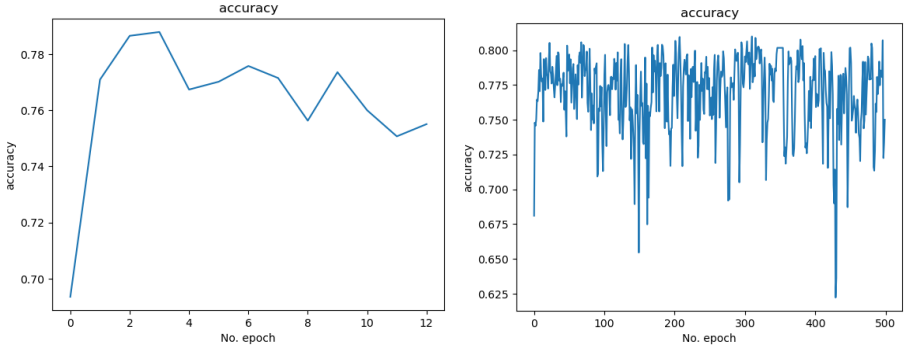


Figure 6: model 2 learning rate 0.01 early stopping vs no early stopping



that the learning rate is too large, and the training accuracy is waggling up and down a lot. And Figure 7 shows 0.5 is way too large for the learning rate. The training accuracy can not be optimized, because the the learning rate is so big that the improvement is waggling away for the best solution.

6 Discussion

Generally, this dataset is very suitable for data training. All of the parameters of the structure and the process will affect the outcome of a NNs. However, there is no direction of the effect. Sometimes adding a layer will help, but sometime don't. The final test accuracy is around 0.88. The models we design will get very close the best accuracy when the learning rate is suitable.

Figure 7: model 2 learning rate 0.5 early stopping vs no early stopping

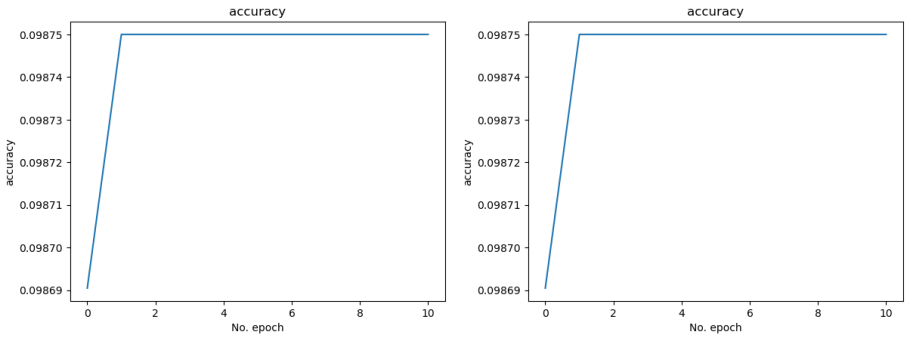
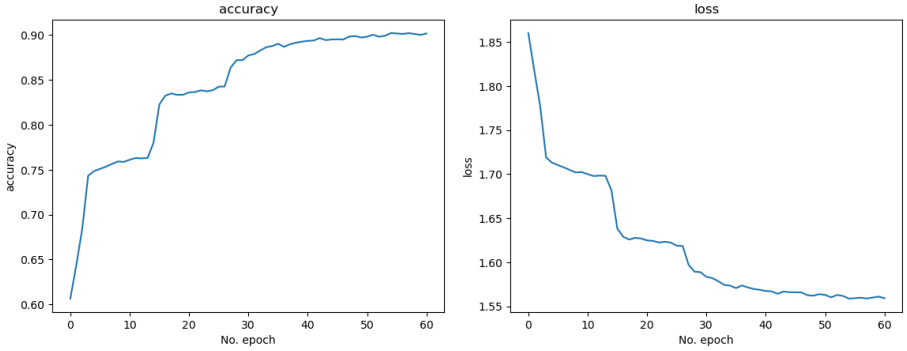


Figure 8: model 5 training accuracy and training loss



7 Conclusions

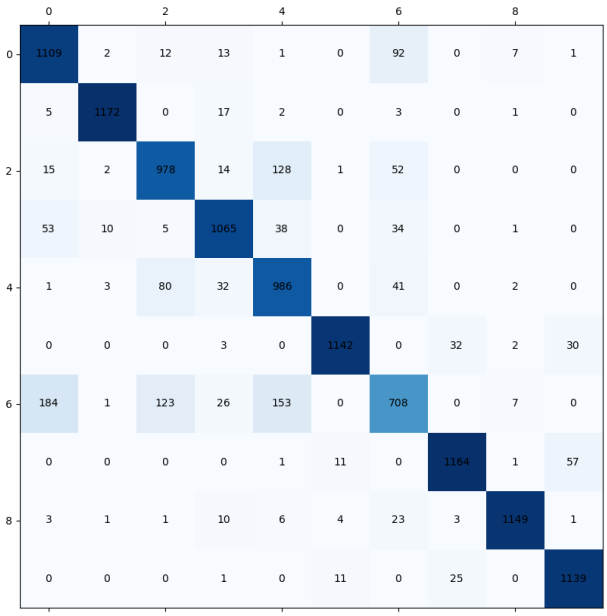
We got different structures that will give the best 0.88 accuracy for the test date. Among them, model 3 with early stopping give us the best result.

Part II
Image Classification on CIFAR-100

1 Introduction

Neural Network (NN) is an algorithm that is modeled from the idea of neural cells in the human brain. It had been widely used in many areas, such as image classification, speech recognition, game AI, and so on [3].In this report,

Figure 9: model 5 confusion matrix



we used a self-defined convolutional neural network (CNN) structure and the residual network (ResNet) to solve image classification problems on CIFAR-100 [2]. Classification of CIFAR-100 dataset is technically difficult considering the numerous classes the images are labeled with. In thus, it is widely used in theory as a benchmark of the model developed. In this report, we trained and tested various architectures by applying generalization, tuning hyperparameters, and increasing depth of the model. The properties of the CNN were further discovered by comparing the performance between models with different architectures and hyperparameters. As a result, our best model had a test accuracy of 42%, which is far from real application. We found the defeat of this model comes from a serious over-fitting issue, which needs further attention to address this problem in future.

2 Problem Description

In this homework, we studied a image classification problem on CIFAR-100 dataset. CIFAR-100 dataset has 100 classes (fine label) including various features, such as animals, plants, vehicles, which can be also grouped into 20 su-

perclasses (coarse label). We apply the fine label in this study. Our objective is to investigate the impacts of different hyperparameters on model performance. Because it is a image classification problem, we employed the convolutional neural network (CNN) as the main structure. We trained and tested a architecture with a self-defined unit and compared it with a ResNet model. For hyperparameter tuning, we primarily focused on addressing the over-fitting problem and exploring the impacts of various activation functions, learning rates, and optimizers.

3 Approaches

3.1 Data source

The CIFAR-100 dataset (Fig.10) consists of 60000 32×32 colour images in 100 classes, with 600 images per class. There are 50000 training images and 10000 test images. We load 5000 training images from the dataset as 4-dimensional NHWC ($N \times H \times w \times c$) tensor. Here N is the number of training image (6000), $H \times w$ is the size of the image (32×32). $c = 3$ represents 3 color panes. The 100 classes are listed on the website[1].

Then the former 90% dataset is used as training data, while the last 10% is saved for testing. The training dataset is further split into two parts with 80% training data and 20% validation data.

3.2 Preprocessing

The original data is integrate in 8 bit. We firstly need to transfer them into float, otherwise they cannot be used to train the model. Then they were normalized by dividing 255, which is the maximum value of color digits, to expresses a [0,1] representation. The original 100 classes were labeled with 100 numeric values from 0 to 99. However, considering there are no natural ordering relationships among these classes, using numeric categorical labels is not logic. So we converted the integer labels to one-hot vectors.

3.3 Hyperparameters of neural network

To build a well performed architecture of neural network, We started from a simple one, as shown in Fig.11, which includes two convolutional+pooling layers for this problem. The convolutional layers were followed by at least one connected layer and softmax as the output layers.

In the training process, a small number of 360 images in each batch was used to fasten calculation. At the same time, this choice of batch size kept an appropriately enough images for each class (3.6 images on average) at each learning epoch to stably decrease loss function. A enough epoch number of 100 was set with early stopping, which was used to avoid over-fitting and to determine the

Figure 10: The CIFAR-100 dataset

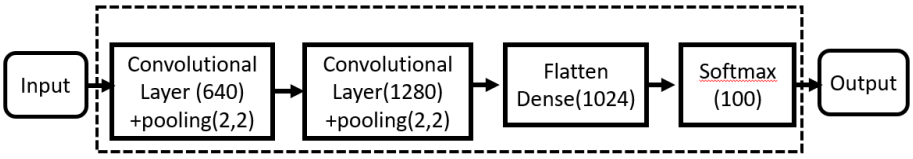


Figure 11: General architecture of the CNN.

converging point quickly. The cross-entropy loss on training set was used as the monitor rather than validation accuracy to get informative Learning curve as shown in Fig.12

The learning curve shows that this model was seriously over-fitting, since the training accuracy if far to large compared with validation accuracy. The test accuracy was only 27%.

3.3.1 Generalization

To solve the over-fitting problem in the last model, we tried to apply generalization methods, including L2 regularization and dropout with a rate of 0.2. Besides, to maker the NN faster and more stable, batch normalization layers were included before the convolutional layers through normalization of the input layer by re-centering and re-scaling. The changed model is shown in Fig.13.

The learning curve (Fig.14) was obtained after training this generalized model. The over-fitting problem might be alleviated since the training and validation accuracy were close (0.01). However, with this very small accuracy, it is apparent that this model was inversely under-fitting. The minor difference between training and validation accuracy also might be caused by the inadequate learning capability of the model.

3.3.2 Activation functions and optimizer

Hence, we compared different choices of activation functions (ELU and ReLU) and optimizers (Adam and SGD). Based on model Fig.11, we tested their performances. Their performance are shown in Tab.3. As a result, ReLU was both more accurate and faster than ELU. Adam was less accurate than SGD, but it is much more efficient than SGD. Therefore, we stayed on applying ReLu and Adam in the model.

3.3.3 Depth of the model

After getting the best set of regularization, activation function and optimizer, we employed the combination of 2 convolutional layers together with maximum pooling, batch normalization and dropout as a unit, which was represented by

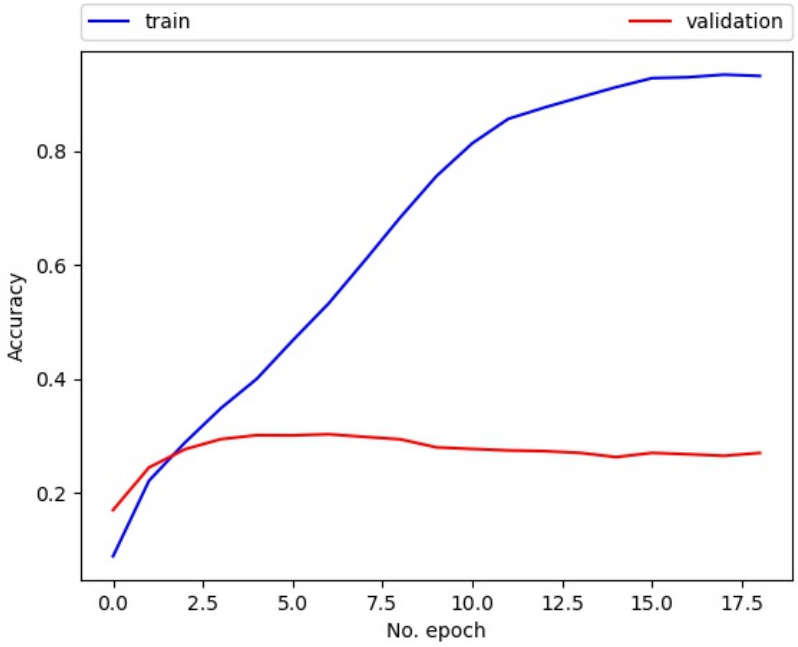


Figure 12: Learning curve of model in Fig. 11

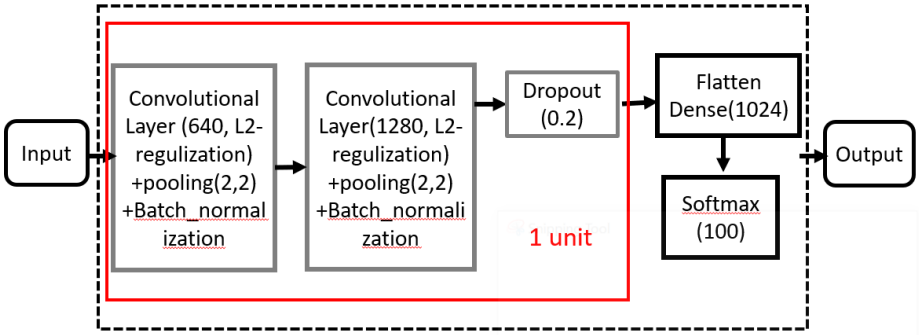


Figure 13: A self-defined unit in CNN.

Table 3: Selection of activation functions and optimizers

Activation functions	ReLU	ReLu	ELU
Optimizer	Adam	SGD	Adam
Time per step	144	544	429
Test accuracy	0.27	0.31	0.24

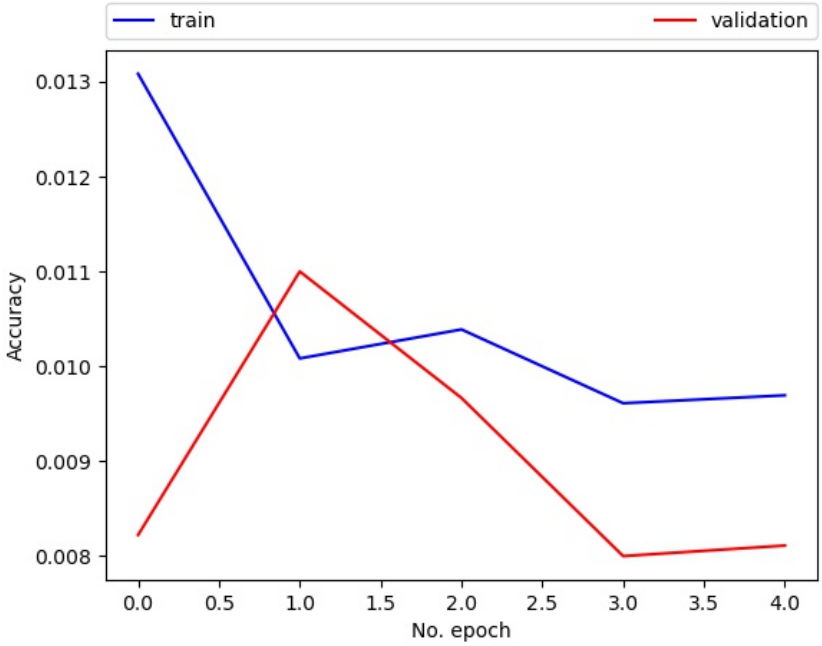


Figure 14: Learning curve of the model based on Fig. 13

Table 4: Test accuracy of CNN with different number of unit

Number of unit	1	2	3	4
Test accuracy	0.27	0.37	0.42	0.39

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 16, 16, 64)        9408
_________________________________________________________________
batch_normalization (BatchNo (None, 16, 16, 64)        256
_________________________________________________________________
activation (Activation)      (None, 16, 16, 64)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 8, 8, 64)          0
_________________________________________________________________
ru__res_ne_t34 (ru_ResNET34) (None, 8, 8, 64)          74240
_________________________________________________________________
ru__res_ne_t34_1 (ru_ResNET3 (None, 8, 8, 64)          74240
_________________________________________________________________
ru__res_ne_t34_2 (ru_ResNET3 (None, 4, 4, 128)         230912
_________________________________________________________________
ru__res_ne_t34_3 (ru_ResNET3 (None, 4, 4, 128)         295936
_________________________________________________________________
ru__res_ne_t34_4 (ru_ResNET3 (None, 2, 2, 256)         920576
_________________________________________________________________
ru__res_ne_t34_5 (ru_ResNET3 (None, 2, 2, 256)         1181696
_________________________________________________________________
global_average_pooling2d (Gl (None, 256)               0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 100)               25700
=================================================================
```

Figure 15: The ResNet-based architecture with 6 RUs.

the red square in Fig.13. By sequentially adding the number of the self-defined unit, we ended up with various CNNs of different depths. The models with one to four units were separately trained and compared.4. As shown in the table, the model with three self-defined units had the best performance with a test accuracy of 0.42.

3.3.4 Comparison with ResNet

To evaluate the performance of the self-defined unit, we also compared it with a CNN architecture based on ResNet that contains six RUs. Figure 15 shows the structure of the ResNet-based model. The six RUs are in size 64, 18, 256 with two repeats. The model applied the same regularization setting that included the L2 norms in the convolutional layers of RUs and the dropout layer with the ratio of 0.2 before the softmax output layer. However, the test accuracy ended up with 0.41 that was slightly lower than the self-defined architecture. In summary, the final model we set up is shown in Fig.16.

Model: "sequential"

Layer (type)	Output Shape	Param #
zero_padding2d (ZeroPadding2	(None, 40, 40, 3)	0
conv2d (Conv2D)	(None, 40, 40, 40)	1120
batch_normalization (BatchNo	(None, 40, 40, 40)	160
conv2d_1 (Conv2D)	(None, 40, 40, 80)	28880
max_pooling2d (MaxPooling2D)	(None, 20, 20, 80)	0
batch_normalization_1 (Batch	(None, 20, 20, 80)	320
dropout (Dropout)	(None, 20, 20, 80)	0
conv2d_2 (Conv2D)	(None, 20, 20, 160)	115360
batch_normalization_2 (Batch	(None, 20, 20, 160)	640
conv2d_3 (Conv2D)	(None, 20, 20, 320)	461120
max_pooling2d_1 (MaxPooling2	(None, 10, 10, 320)	0
batch_normalization_3 (Batch	(None, 10, 10, 320)	1280
dropout_1 (Dropout)	(None, 10, 10, 320)	0
conv2d_4 (Conv2D)	(None, 10, 10, 640)	1843840
batch_normalization_4 (Batch	(None, 10, 10, 640)	2560
conv2d_5 (Conv2D)	(None, 10, 10, 1280)	7374080
max_pooling2d_2 (MaxPooling2	(None, 5, 5, 1280)	0
batch_normalization_5 (Batch	(None, 5, 5, 1280)	5120
dropout_2 (Dropout)	(None, 5, 5, 1280)	0
flatten (Flatten)	(None, 32000)	0
dense (Dense)	(None, 1024)	32769024
dense_1 (Dense)	(None, 600)	615000
dense_2 (Dense)	(None, 100)	60100

Figure 16: The final constructed architecture of CNN

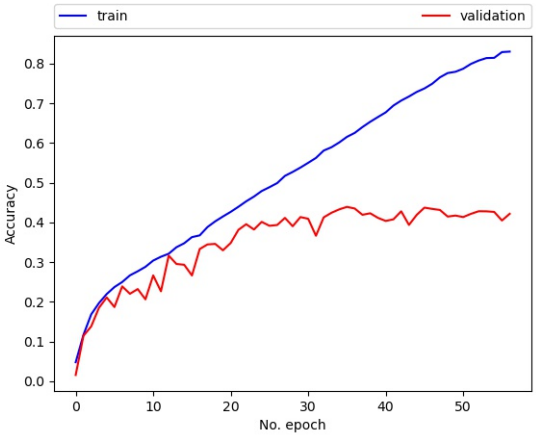


Figure 17: Learning curve of the final constructed architecture of NN

3.4 Performance measurement

After training the model, we calculated the categorical accuracy together with 95% confidential interval (CI) on test set to measure the performance since the class labels will be one-hot vectors. The CI was applied by Equation (1) with an assumption of the error following the Gaussian distribution, where p is the test accuracy. Accuracy is a percent value of right classifying instants in total predictions. Moreover, the confusion matrix was calculated in a classification problem, providing more comprehensive understandings of whether the model misclassified the images. Each row of the matrix represents the instances in a predicted class, while each column represents the instances in an actual class.

$$CI = p \pm z_{\frac{\alpha}{2}} \sqrt{\frac{p(1-p)}{n}}, \alpha = 0.05 \tag{1}$$

4 Experimental Results

The learning curve of the final model is shown in Fig.17. Base on the learning curve, the model is still over-ftting since the validation accuracy is smaller than training accuracy. However, compared with the model in Fig.11, this model is less over-fitting.

The calculated test accuracy is [40.9%, 43.6%] with 95% confidence, which is the best result we achieved. The confusion matrix is shown in Fig.18

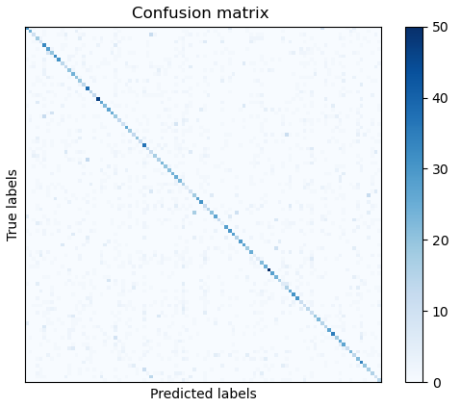


Figure 18: Confusion matrix of final model

5 Discussion

The main problem of the final architecture is seriously over-fitting. The accuracy is too poor to be used in application. Based on the confusion matrix calculated above, even though this model has a potential of finding right classes, it still randomly confuses with other classes.

With hyperparameter turning and comparison with the mutual architecture based on ResNet, we realized that not only the structure of neural networks will determine the model performance, but also the various tuning tricks and preprocessing techniques. Without a proper preprocessing step and fine tuning, the ResNet model could not achieve an expected performance.

This model need to be further developed. The most important thing is to solve the over-fitting problem. Otherwise, the model can not afford a deeper model. One method we can work on in the future is to develop a generalized loss function. Besides, we can also try a larger dropout rating and adaptive learning rate.

6 Conclusions

In summary, we developed several CNN models to classify CIFAR-100 dataset. By applying preprocessing, hyperparameters tuning, adding generalizations and the number of layers, we built a final model with 6 convolution layers. This model had a poor performance on the test set with a accuracy of 42%. The major problem of the model is serious over-fitting, which prohibits the possibility of

applying a deeper structure. In the future, we will try some other generalization approaches such as a generalized loss function to solve this issue.

References

[1] URL: `https://www.cs.toronto.edu/~kriz/cifar.html`.

[2] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[3] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.