MSc. Neural Network Report

Lin Li, Jianing Qiu, Xinyi Du, Yiming Jing
Course: CO395, Imperial College London

8th March, 2018

1 Execution

Concerning the dataset FER2013, our fully connected neural network achieves the overall classification rate of 45.28% after optimizing a set of hyper-parameters by the stage optimization. At the same time, we also experiment three convolutional neural networks with different architectures and achieve the best overall classification rate of 66.29% on the residual network.

2 Network Components

Generally, a typical feedforward neural network is assembled by an input layer, hidden layer(s) and an output layer. The hidden layer contains a set of neurons, i.e. linear function unit, followed by the activation function and the output layer is usually also connected to a classifier to produce the final classification. In this case, our fully connected neural network utilizes the rectifier linear unit(ReLU) immediately following each hidden layers, the Dropout layer as regularization and Softmax classifier as the final prediction producer connected to the output layer.

2.1 Linear Layer

Linear layers use the linear function as shown in equation (1) to perform simple matrix multiplication and addition on the input image x_i, whose pixels are normally streched out to a single column vector with the shape of $[D \times 1]$. In the forward pass, the input x_i will be transformed into a new vector with different values by two learnable parameters W and b. If the shape of W is $[K \times D]$ and that of b is $[K \times 1]$, then after the forward pass is finished, the shape of x_i would become $[K \times 1]$. In the backward pass, the partial derivatives of x_i, W and b can be derived by using simple calculus, which are shown in equation (2)-(4).

$$f(x_i, W, b) = Wx_i + b \tag{1}$$

$$\frac{\partial f}{\partial x_i} = W \tag{2}$$

$$\frac{\partial f}{\partial W} = x_i \tag{3}$$

$$\frac{\partial f}{\partial b} = 1 \tag{4}$$

2.2 ReLU Activation

Rectified Linear Unit (ReLU) activation function which has the form as equation (5) simply thresholds an activation at 0 in the forward pass. In the backward pass, the derivative of element less than 0 would be 0, and that of the rest would be 1 as shown in equation (6).

$$f(x) = max(0, x) \tag{5}$$

$$\frac{\partial f}{\partial x} = \begin{cases} 0, & x < 0 \\ 1, & otherwise \end{cases} \tag{6}$$

2.3 Dropout

The reason why dropout is implemented is that if training neural network has limited dataset, overfitting occurs at the times; hence dropout [1] has been raised. At training, half of the detectors will stop working, in this case, the generalization ability of this model will be improved. In more details, every neuron has a certain probability to be removed in order to prevent the case that the existence of one neuron does not depend on the others. This project uses inverted dropout as the regularization, the formula below is the possibility to keep neurons active:

$$q = 1 - p \tag{7}$$

where p is the probability of dropped neurons. Hence, the application of dropout can be state as:

$$o_i = m_i \times a(\sum_{k=1}^{d_i} w_k x_k + b) \tag{8}$$

Where: m_i is the i-th dimensional vector of Bernoulli distributed variables. $h(X) = XW + b$ is a linear transformation with d_i dimensional input X in a d_h dimensional output space. a(h) is the activation function.

In addition, the formulas for the train phase and test phase for the inverted dropout are:

$$o_i = \frac{1}{q} \times m_i \times a(\sum_{k=1}^{d_i} w_k x_k + b) \tag{9}$$

$$o_i = a(\sum_{k=1}^{d_i} w_k x_k + b) \tag{10}$$

In implementation, there are two outputs for both forward and backward cases, output of dropout and dropout mask. Dropout mask is using the binomial method to calculate m_i multiplies by the inverse of the keep probability $\frac{1}{(1-p)}$. The data structure of dropout mask is a matrix with 0 and 1, which satisfies normal distribution; the probability of a single being 1 or 0 is depending on probability factor, $1 - p$, and the matrix will be rescaled by $\frac{1}{1-p}$. The input is a detector matrix which is the same size with the mask, so that multiplying it with the mask, certain neurons will become deactivated corresponding to 0 in mask matrix. While backward propagation, to improve the performance of loss function, the derivation is used to optimize the weight factor.

2.4 Softmax Classifier

From the probabilistic view, the softmax function(equation 11), which mathematically squashes a K-dimensional vector f to a new K-dimensional vector P between zero and one that sum to one, produces the normalized probability assigned to the correct label y_i given the input, i.e. raw image, x_i and parameterized by W.(equation 12) To make this, the softmax classifier interprets the class scores inside the vector f as the unnormalized log probabilities. Therefore exponentiating these scores gives the unnormalized probabilities and the division operation performs the normalization so that the outputs of function sum to one.

$$P_k = \frac{e^{f_k}}{\sum_{n=1}^{K} e^{f_n}}, k = 1, \cdots, K \tag{11}$$

$$P(y_i|x_i; W) = \frac{e^{f_{y_i}}}{\sum_{n=1}^{K} e^{f_n}} \tag{12}$$

$$f_i = Wx_i \tag{13}$$

where x_i is the i-th input, y_i is the corresponding correct label for the input x_i, f_i is the i-th element of the vector of class scores f, which is calculated by the equation (13).

The right loss function for softmax classifier is the negative log likelihood function(equation 14), which can be easily simplified to the form of equation (15).

$$L = -\sum_{k=1}^{K} t_k log(p_k) \tag{14}$$

$$\begin{aligned} L = -log(p_y) &= -log(\frac{e^{f_y}}{\sum_{k=1}^{K} e^{f_k}}) \\ &= -f_y + log(\sum_{k=1}^{K} e^{f_k}) \end{aligned} \tag{15}$$

where L is the value of loss, t_k is the k-th value in the target vector[1](e.g. $[0, 1, \cdots, 0]$) and y is the real value of given label.

After obtaining the loss function, we can then compute how the loss changes with respect to the classes scores, i.e. the output of network. In other words, we are looking for $\frac{\partial L}{\partial f_k}$, which can be computed simply from equation (16) as below concerning two conditions:

$$\frac{\partial L}{\partial f_k} = \begin{cases} p_k, & k \neq y \\ p_k - 1, & k = y \end{cases} \tag{16}$$

3 Sanity Checks of FCNN

To check if the network works, we ran two experiments, overfitting a small dataset and learning a model to achieve at least 50% accuracy on the dataset CIFAR-10, before plunging into expensive optimization. The results are shown and discussed as below:

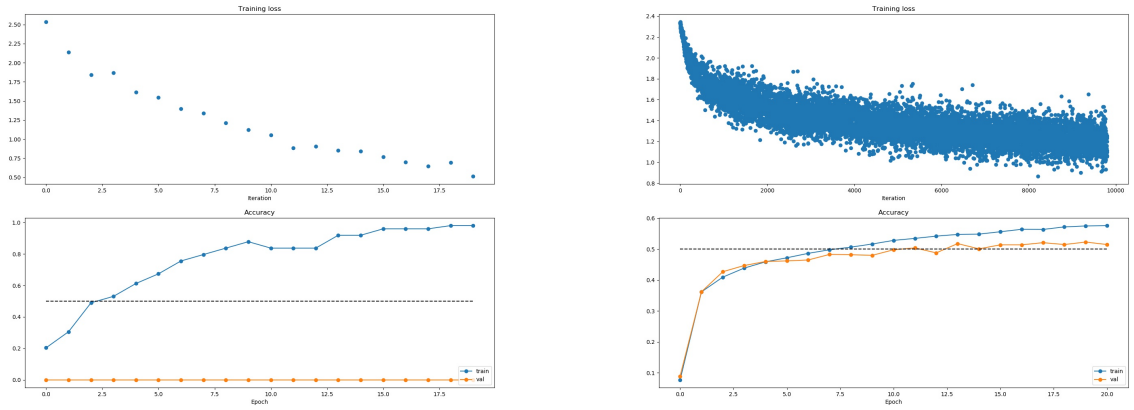
3.1 Overfitting a Small Dataset

In order to decrease the generalization of model, the architecture is designed as a very large network, two hidden layers with four hundred neurons inside each layer, compared to the size(50 images) of dataset and the dimensions($32 * 32 * 3$) of input images. Besides, no any regularizations, specifically L2 regularization and Dropout, are applied to the model in order to overfit data. The other hyper-parameters are just left as default, the update rule(SGD), learning rate(0.001) and number of epochs(20) etc., since they do not have a direct effect on the model's generalization ability.

3.2 50% Accuracy on CIFAR-10

The intuition is to improve the generalization ability of network by reducing the number of neurons in each hidden layer to one hundred so that the model can perform also well on the validation set. Moreover, another change compared to the parameters of overfitting one is annealing the learning rate using the step decay strategy with a decay rate of 0.95, while all other hyper-parameters are identical with the overfitting network's.

[1]the vector represents the label by one-hot encoding.



(a) The loss and accuracy of overfitting case

(b) The loss and accuracy of the case achieving 50% accuracy

Figure 1: The results of FCNNs on the CIFAR-10 dataset

4 Optimization of FCNN

The hyper-parameters of this network consist of learning rate, learning rate update schedule, momentum, stopping criterion, regularization strength for L2, dropout possibility and topology, i.e. the number of hidden layers and number of neurons in each layer. In practice, due to the limitation of time and computation resources, we only fine-tuned the parameters initial learning rate, regularization strength, dropout probability and the topology architecture, while the remaining parameters were chosen following the experiential advices.

4.1 Presupposed Hyper-parameters

Initial Architecture The initial architecture of the network is designed as a two-hidden-layers fully connected neural network with 1150 neurons in each hidden layer according to the expert's advices [2], the number of neurons in each layer is usually between the number of neurons in the input and output layers, and the results of experiments based on the advices. Basically, the number 1150 is just an average of the sizes of input(2304[2] in this case) and output(7 in this case) layers.

Learning Procedure Simply, the solver will stop learning early only if the accuracy for the validation set continuously decreases over five epochs, which approximately means the overfitting problem turns significant, or the training reaches the predefined maximum number of epochs. For the parameter updating, we firstly add the momentum(equation 17, 18), 0.9[3], to the original vanilla updating, stochastic gradient descent(SGD), and then apply the step decay, 0.9[4], as our learning rate update schedule to anneal the learning rate over time.

$$v_t = m \cdot v_{t-1} - (1-m)(lr \cdot dw_t) \tag{17}$$

$$w_t = w_{t-1} + v_t \tag{18}$$

4.2 Strategy and Practice

The core of our optimization strategy is random search, which is recognized as a better choice than searching on a grid. [4] Based on this insight, we develop the searching strategy from simple

[2]The dimension of input data is [48, 48, 1], which means a resolution of 48*48 with one channel of gray scale. The motivation for using gray scale is due to the fact that the all inputs are black-white photos.

[3]Follow the advices from the course materials

[4]Follow the advices from the note of course CS231n of Standford: http://cs231n.github.io/neural-networks-3/

random searching to the multi-stage one, which means the further searching range will be determined by the results of previous stage. In practice, we implement a script to search the local best parameter semi-automatically[5] for three stages:

1. the script does a quick search, 20(50) iterations[6] with 20(5) epochs, in a coarse range
2. the coarse range is shrunken based on the 'best' value and the distribution of performance in the previous stage and the strength of search is increased to 20(40) epochs.
3. the last stage perform a detailed search in the narrower range, determined by the results of second stage, with 40(50) epochs.

As shown above, the strength of search, the number of iterations and epochs, is not fixed, which is mainly affected by the specific situation and the limitation of available time and computation resources. For example, in the early stage for searching the learning rate, the number of iterations is relatively larger than that of dropout probability and regularization strength because there are more early stoppings happening when searching learning rate, while the training usually runs out of the maximum epochs when searching others.

Besides, during the searching, we mainly utilize the best accuracy occurring on the validation set to evaluate the performance of hyper-parameter out of convenience cause because the accuracy on the validation set has been already calculated and stored in the solver. Basically, we prefer the value within the range with lots of parameters achieving high performance as the central point of the range of next stage.

Finally, the best value for each parameter will be considered among the best values of each stage and the initial value, which means the optimized parameter maybe occur outside the last stage. Besides, there are also many tiny informal experiments, usually picking up some proposal values and comparing the performance with existing best ones, made to help correct the range of each stage.

Practice Following the aforementioned strategy, the optimization starts from the initial learning rate and then comes to the probability of dropout and the strength for regularization separately with the learning rate fixed at the 'best' value. Finally, the fine-tuning of network architecture is developed based on the previous best hyper-parameters, i.e. in this case without any regularizations. The procedures and results of optimization have been shown in the table 1. Moreover, the details of each stage experiment can be also found in the appendix A.

To be more specific, given the range, the sampling for learning rate and regularization strength has a look like equation (19), while the parameter dropout probability is instead searched in the original scale(equation 20). The corresponding criterion is if the parameter has a multiplicative effect on the parameter updating.[7] Apart from that, especially the criterion for better learning rate also consists of the frequency of appropriate update[8](equation 21) and the shape of loss figure, which are related to the speed of loss decreasing. A rough heuristic shows that the update should be around 1e-3. Intuitively, more updates are around this value, better the learning rate performs.

$$lr \ or \ reg = 10^{uniform(lower \ bound, \ upper \ bound)} \tag{19}$$

$$dropout = uniform(lower \ bound, \ upper \ bound) \tag{20}$$

$$\delta W = \frac{||W'||}{||W||} \tag{21}$$

where the lr is the learning rate, reg is the regularization strength, δW measures the update and W' is the updated weights.

[5]Semi-automatic staging means that the observer has to set the parameters for next stage searching manually, while the searching within each stage is done by the program. The detailed instructions can be found in the readme file.

[6]one random proposal of hyper-parameter for one iteration

[7]The detailed explanation can be found from the link: http://cs231n.github.io/neural-networks-3/

[8]In practice, the program calculates the frequency of updates and checks if within a range around 1e-3

	Stage 1				Stage 2				Stage 3			
	itr.	epoch	range	best	itr.	epoch	range	best	itr.	epoch	range	best
lr	100	20	(1e-5, 1e-1)	0.00191	200	20	(1e-4, 1e-2)	0.00197	100	40	(1e-3.35, 1e-2.35)	0.00334
dropout	50	20	(0, 1)	0.3670	50	40	(0.1835, 0.6835)	0.23524	20	50	(0.1835, 0.4594)	0.1850
reg	50	20	(1e-6, 1)	1e-6	20	40	(1e-6, 1e-1)	0.00207	20	50	(1e-6, 1e-1.842)	0.00091
layers	20	20	(7, 2304)	(1618, 1041)[9]	40	40	(400, 1800)	(1758, 1302)	30	50	(800, 1800)	(1510, 1361)
decay	0.9											
momentum	0.9											
batch size	100											
train set	26709											
valid set	2000											
test set	3589											

Table 1: Stage optimization for hyper-parameters

4.3 Results Analysis

After running all optimizations, we find a set of best parameters: learning rate 0.00344, dropout probability 0.23524, regularization strength 0.00091 and architecture [1150, 1150], while especially all searched topologies can not outperform the initial one. Then we test the performance of different potential combos among them, vanilla(no regularization)(45.28%), only using L2(45.28%), only using dropout(43.13%) and both used(43.27%), and decide the vanilla configuration as final version to be submitted. The reason why we can't decide the final architecture just based on the results of optimization is mainly the difference of number of epochs, e.g. the number of epochs for searching learning rate is sometimes just 20, while the others are usually 40 or 50.

Regularization Obviously, the regularizations, L2 and dropout, do not improve the performance of network concerning the overall performance or even the performance on each class. Moreover, the dropout even upgrades the classification rate of network by 2%. The main reason is that it performs very bad, 9.84%, when classifying emotion Disgust, especially predicting lots of positive samples as other classes, since the recall rate of that emotion for using dropout is super low, just 5.36%, compared to the 19.36% of same rate when using L2. Apart from that, the value of regularization strength is very close to the lower bound of the searching range, which usually means that a better value possibly exists outside the lower bound. But, as shown in the table 1, the lower bound, 1e-6, of regularization strength searching range is also very close to 0, which means the regularization may be not necessary. Those are also the reasons why not applying any regularizations.

But the application of regularization also has some benefits concerning the figures(3, 4) of loss and accuracy. Firstly, the gap between the accuracies on the training set and validation set turns narrower as the regularization strength increasing, i.e. greater dropout probability or L2 regularization strength. Besides, the figure of loss also looks thinner when the strength of L2 rises, i.e. fluctuation is reduced, which is mainly caused by the penalty effect on the large parameters introduced by the L2 regularization.

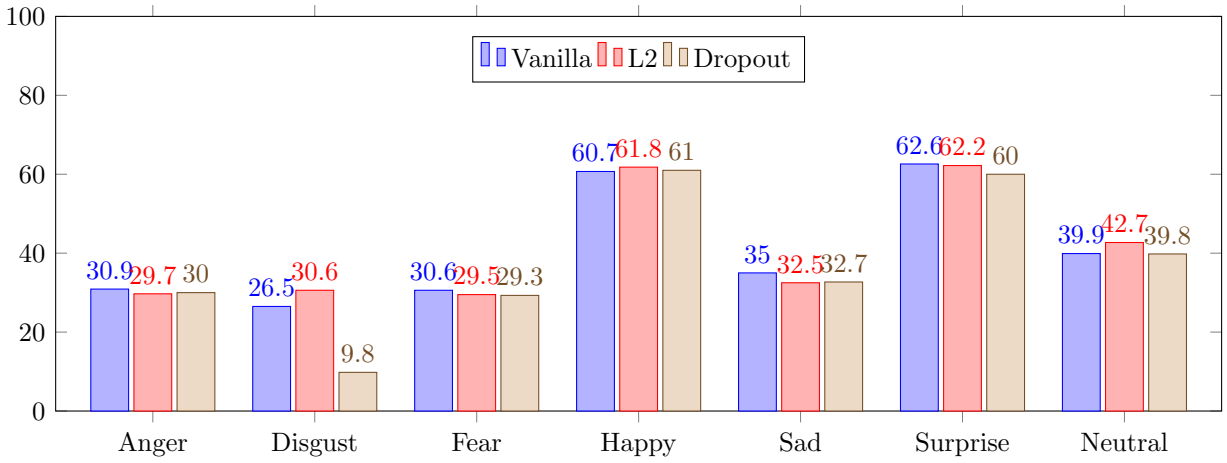
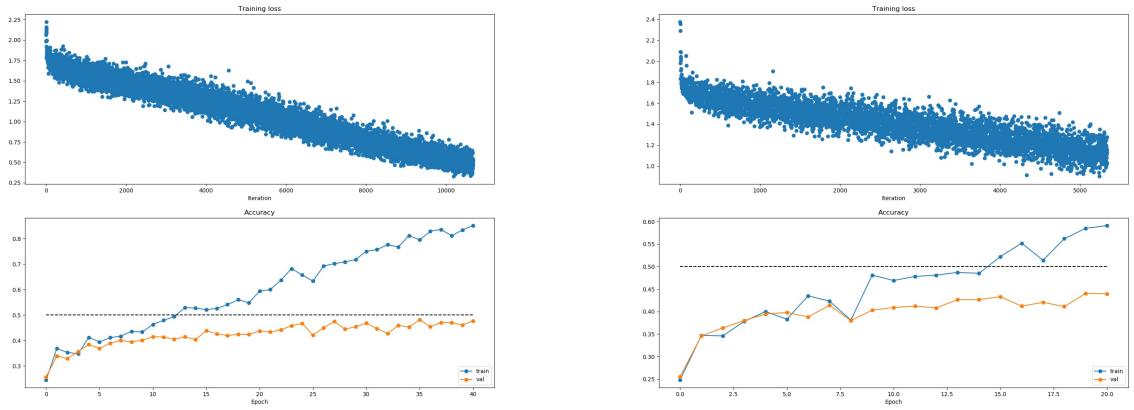
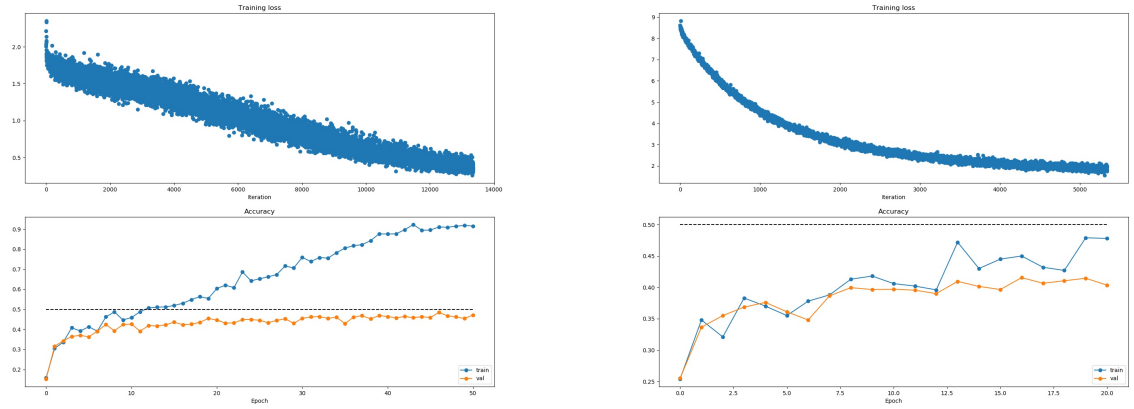


Figure 2: F1 rate of each class concerning the use of regularizations



(a) Low dropout rate (b) High dropout rate

Figure 3: The loss and accuracy of using high and low dropout rate on FER2013



(a) Low L2 regularization (b) High L2 regularization

Figure 4: The loss and accuracy of using high and low L2 regularization rate on FER2013

Performance The model can achieve the overall accuracy of 45.28%, whereas the performances over the different emotions varies a lot. Basically, the network performs better on classifying the emotions like Happy and Surprise but worse on recognizing the emotions Anger, Fear, Sad, Neutral and worst one Disgust. Some explanations can be found from the confusion matrix, which reveals an imbalanced, especially very insufficient samples of emotion Disgust, distribution of the classes of samples. Besides, the model performs very well for the emotion Happy, which also owns largest amount of samples. The problem is that the imbalance maybe also occurs on the training dataset so that the parameters belonging to particular neurons are significantly affected. Specially, the precision rate of emotion Disgust exceeds the average level a lot, which means the samples predicted as Disgust are usually with the labels of Disgust.

Another discovery observed from the figure 6 of loss and accuracy is that the loss value can be eliminated to a very low level with the accuracy on training set increasing to a very high level, whereas the accuracy on validation set is stuck around 45%. It is caused by the different computational attributes of loss and accuracy. Considering the equation (15), the loss value depends on the normalized probability, confidence of prediction, of actual class. Therefore, it's possible that, for example, the samples of Happy have already been predicted by the model very well but it still receives lots of samples of Happy and increases its confidence on predicting this emotion, while the accuracy can't be improved as this increasing.

Emotion	Anger	Disgust	Fear	Happy	Sad	Surprise	Neutral	Act. Sum
Anger	138	0	57	114	80	19	59	467
Disgust	10	9	6	16	4	2	9	56
Fear	67	0	139	98	77	46	69	496
Happy	51	1	40	633	72	24	74	895
Sad	79	0	80	148	212	30	104	653
Surprise	23	0	39	45	17	255	36	415
Neutral	57	2	51	138	96	24	239	607
Pre. Sum	425	12	412	1192	558	400	590	3589

Table 2: Confusion matrix of the optimized FCNN

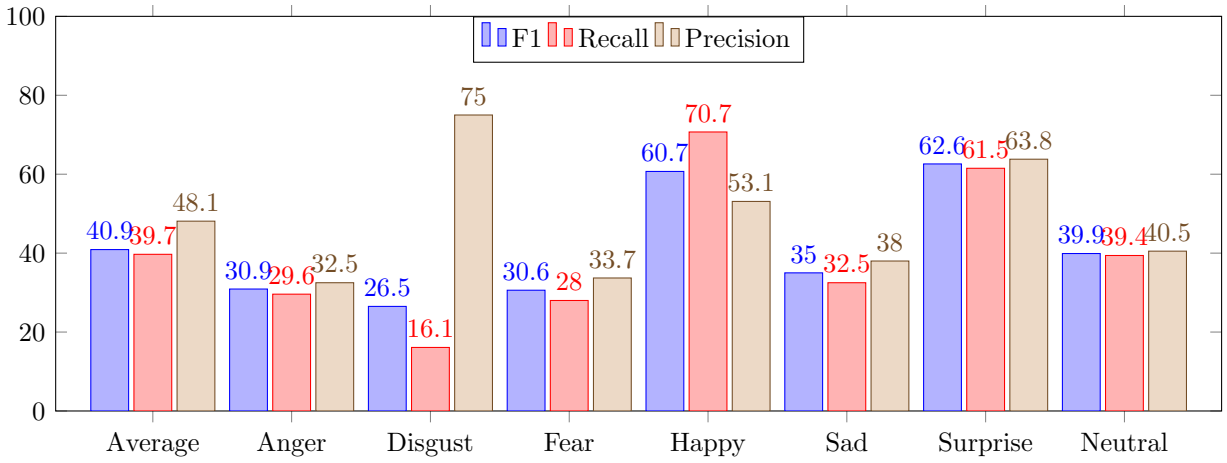


Figure 5: Performance concerning each class and average for optimized FCNN

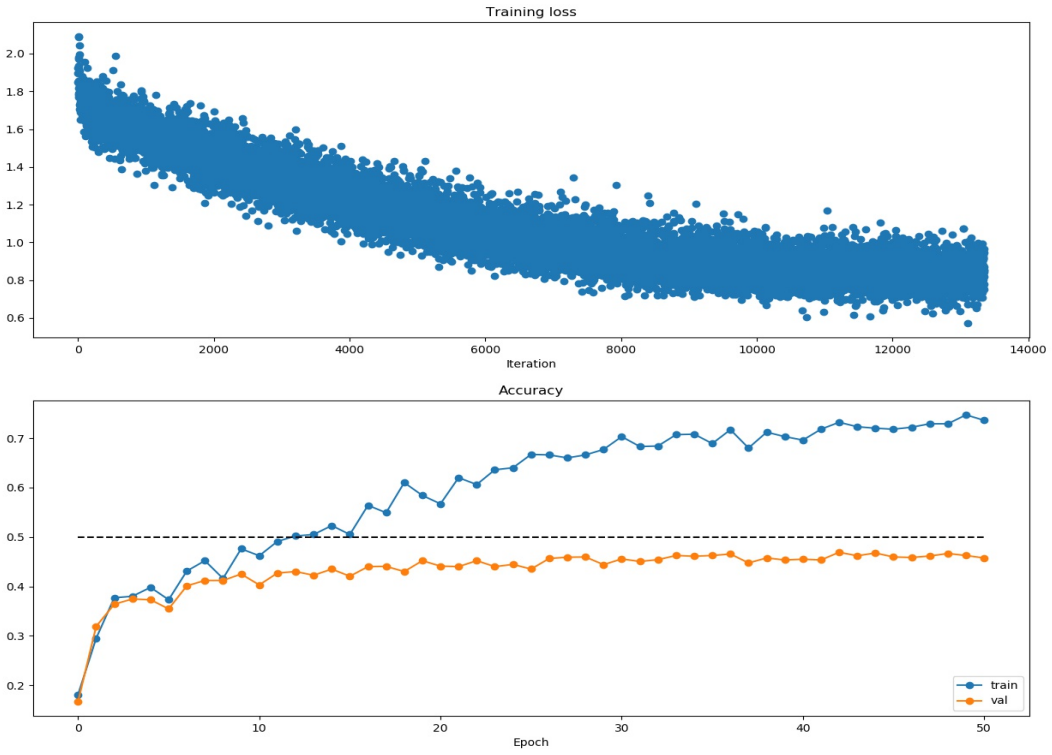


Figure 6: The loss and accuracy of optimized FCNN

5 Convolutional Neural Network

There are three different convolutional neural networks tried in this coursework. The architecture of these networks can be found in Appendix B. Among them, two CNNs use the basic architecture with the depth of 4 and 6 convolutional layers respectively. The rest one is inspired by the ResNet [3] and in order to trade off the training time and the performance, we only used 3 ResNet blocks such that the depth of the residual neural network could be limited to 20 core convolutional layers.

5.1 Justification of the choice of the hyperparameters

We first started from a shallow convolutional network, so there are only 4 convolutional layers in the first constructed network. The detailed architecture can be found in Appendix B.1, which is simply 2 *conv-relu-conv-relu-maxpooling-dropout* blocks, and on top of them is *fullyConnectedLayer-relu-dropout-fullyConnectedLayer-softmax*. Based on the performance of this network (the best accuracy is 0.6066 on the test set), we increased the depth of the second one to 6 convolutional layers in order to improve the capacity of the network, the architecture of which is shown in Appendix B.2. Other adjusted hyperparameters are listed below, as well as the reasons for the changes.

1. Filters: The second network has two extra convolutional layers so that more features can be learned. The number of filters in each conv layer is 128 with the same kernel size of (3×3) as the previous kernels in this network and the ones in the first network.

2. Stride: Max poolng layer is added after each *conv-relu* block in the second network (6 max pooling layers in total), and in order to reduce the speed of losing the learned features, the stride of the first two pooling layers is set to 1, while the rest is set to 2.

3. Zero padding: All conv layers in the second network use the zero padding of size 1 so that the input and output volumn will have the same spatial size, while in the first network, zero padding of size 1 and no zero padding are used alternately.

The performance of the second convolutional neural network is surprisingly worse than the first one (the best accuracy is 0.5592 on the test set), which indicates simply stacking more convolutional layers might lead to a poor capability of prediction. Thus, in order to prevent the prediction from deteriorating with the increase of the depth of the network, we referred to the ResNet and built a residual network with only 3 residual blocks (21 convolutional layers in total with 20 being the core convolutional layers). This residual network is trained from scratch on the fer2013 dataset and the details of its architecture can be found in the Appendix B.3.

5.2 Performance of the Network

Table 3 shows the confusion matrix of the built residual neural network, whose overall classification rate is 0.6629. Classification and F1 rate of each class can be found in Figure 7.

Emotion	Anger	Disgust	Fear	Happy	Sad	Surprise	Neutral	Act. Sum
Anger	297	2	45	20	56	11	36	467
Disgust	18	29	0	2	5	1	1	56
Fear	65	2	226	16	97	38	52	496
Happy	23	0	19	760	14	19	60	895
Sad	80	0	69	29	355	14	106	653
Surprise	9	0	35	17	8	337	9	415
Neutral	45	2	36	53	89	7	375	607
Pre. Sum	537	35	430	897	624	427	639	3589

Table 3: Confusion matrix of the residual network

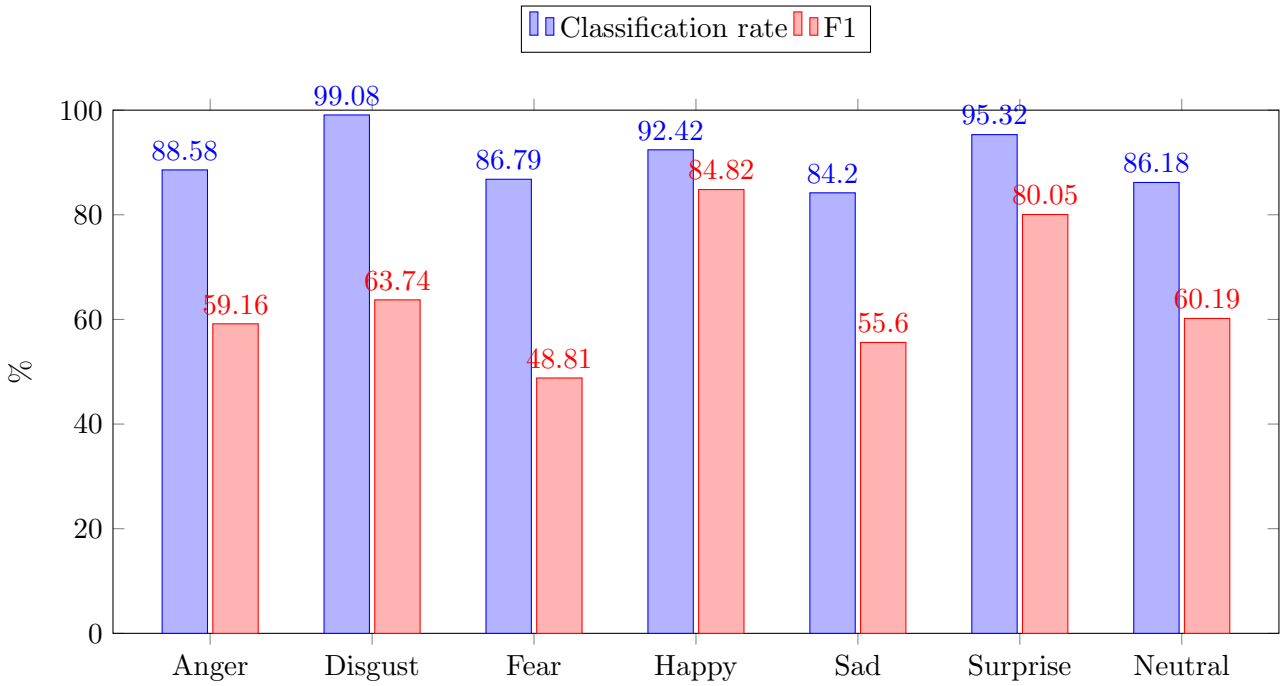


Figure 7: Classification and F1 rate of each class on the residual network

5.3 Comparison between CNN and Feedforward NN

One of the reasons why the overall performance of the trained CNNs, especially the residual network, is better than the Feedforward NN is that CNNs preserve the majority of the spatial correlation of images, i.e., the hierarchical structure of images (e.g., faces are made up of eyes, which are made up of edges, etc.)[10]. Hence CNNs can learn much more meaningful features from the images than Feedforward NN, e.g., an happy mouth normally has a V-like shape (this spatial information is lost in Feedforward NN since it stretches the pixels into one single column vector). In addition, Feedforward NN uses linear function to do the forward pass and in this coursework raw pixels are used to train the network. However, raw pixels normally can not be linearly separated, which may lead to the pool performance of the Feedforward NN. Moreover, Feedforward NN is simply matching the input image with the remembered template so that its generalization is worse than the CNNs. Besides the above general discussions about causes that may lead to different performance of the CNNs and feedforward NN, we also list and analyze several other used techniques that may also have impact on the performance of these networks:

1. Real-time data augmentation: Real-time data augmentation is applied to all CNNs during training, which on the one hand provides more data to the networks, and on the other hand avoids overfitting and helps the learned model generalize better. This techniques is not used in the Feedforward NN.

2. Depth and L2 regularization: The CNNs are much deeper than the Feedforward NN (only 2 hidden layers with neurons 1758 and 1302 respectively), especially for the residual neural network which has 21 convolutional layers and in total 272,583 trainable parameters. L2 regularization is applied to both residual network and Feedforward NN to control overfitting. However, this technique does not make significant difference on Feedforward NN.

3. Optimization algorithm: All CNNs use the *Adam* [11] as the optimizer, while the Feedforward NN uses the *Stochastic Gradient Descent (SGD)* [12] with momentum.

[10]The details can be found from the link: http://cs231n.github.io/neural-networks-1/

[11]The detailed explanation of Adam can be found from the link: http://ruder.io/optimizinggradient-descent/index.html#adam

[12]The detailed explanation of SGD can be found from the link: http://ruder.io/optimizinggradient-descent/index.html#momentum

4. Mean subtraction: In order to accelerate the convergence and improve the accuracy, the mean of data fed into the residual network is subtracted. However, in Feedforward NN, mean subtraction is not used.

5. Learning rate schedule: During the training of residual network, the learning rate will be multiplied by 0.1 after every 40 epochs starting from the initial setting of 0.001, while for the feedForward NN, the learning rate will be multiplied by 0.95 after every epoch starting from the initial setting of 0.00344.

6 Additional Questions

6.1 Question 1

It would not be the case where one is better than the other, both algorithms are designed for different implementations; in general, comparing these two approaches, more precise results always lead to better performances. Neural network is known as a good strategy to deal with complicated data, and can represent approximately to any function, however, the decision tree differs from neural network in the following aspects:

1. Size of Data Set:

   Neural network works better with large dataset, while decision tree has advantages of dealing with small dataset; for instance, neural network has limited performance if the number of data is below one thousand.

2. With Respect of Feature Engineering:

   Neural network requires more features in dataset, for example, missing value imputation, transform from categorical to numerical dataset, data scaling and initial weights; in most of the cases, decision tree model does not require such scenarios.

3. Parameter Adjustment:

   Decision tree model benefits with much easier parameter adjustment than neural network, the most common way to adjust parameters in neural network is trail-and-error, this makes neural network as a black-box model. To deal with this situation,visualization tools become more and more popular to make the model intuitive.

Concluded from above, larger data-set size, more data-set features, no requirement for easier parameter adjustments, will result the better performance in error observation; otherwise, decision tree would be a better approach.

6.2 Question 2

In order to adapt new classes, the changes to be made in the two models, Decision Trees and Neural Networks, are quite different concerning the architectures of models and involved procedures. However, the Neural Network model requires more efforts than Decision Trees model to implement such adaption.

For the Decision Trees model, firstly new decision trees will be trained for added classes respectively. Then the prediction procedure will be also expanded due to the new included decision trees, while the specifications depend on the specific algorithm to integrate the prediction results of all decision trees. Apart from that, the deep effect occurs on the structures of existing decision trees during training and is determined by the degree of noise in the added data. For example, when training one particular decision tree, the new data, even without any noises, will affect the calculation of information gain and thus change the behavior at specific node, like if creating new child node or the selection of best attribute. Generally, more noise inside the new data, more changes in the trees' structures happen.

For the Neural Network model, specifically the typical fully connected neural network, the first one shallow change goes for the output layer and the classifier connected to it, which usually means

the increase of neurons inside the output layer and the classifier's input data and output classifications. Again, the significant changes happen inside the architecture of network. In general, all hyper-parameters are possible to be affected by the extended dataset and need to be refine-tuned, which potentially involves the number of hidden layers, the number of neurons inside each layer, the learning rate, the regularization strength etc.

References

[1] Zhou ZH. Ensemble methods: foundations and algorithms. CRC press; 2012 Jun 6.

[2] Heaton J. Introduction to neural networks with Java. Heaton Research, Inc.; 2008.

[3] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. InProceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 770-778).

[4] Bergstra J, Bengio Y. Random search for hyper-parameter optimization. Journal of Machine Learning Research. 2012;13(Feb):281-305.

Appendices

A Results of stage optimization

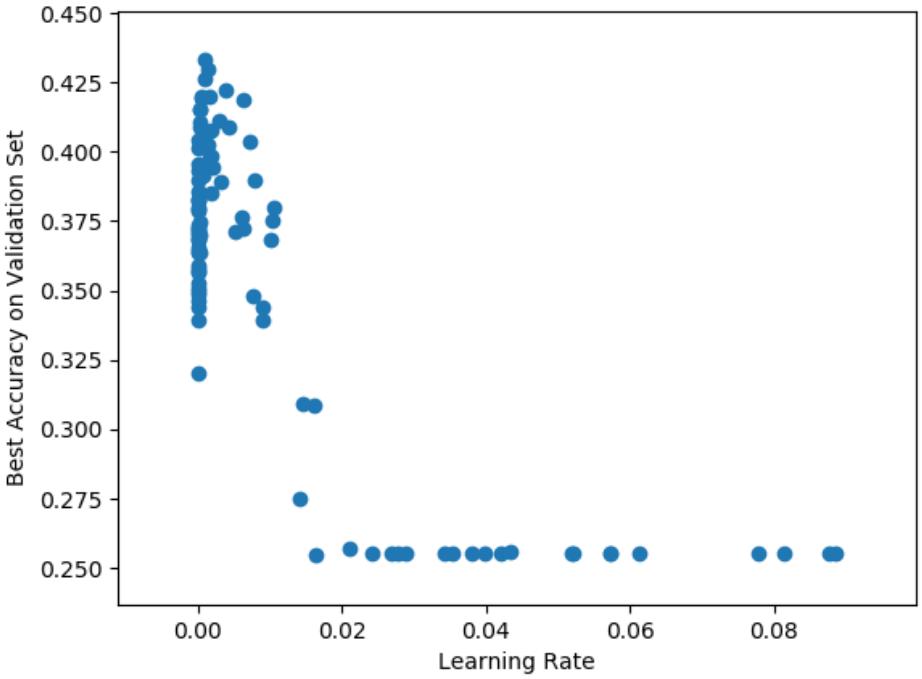


Figure 8: Learning rate stage 1 within the range (-5, -1) concerning accuracy

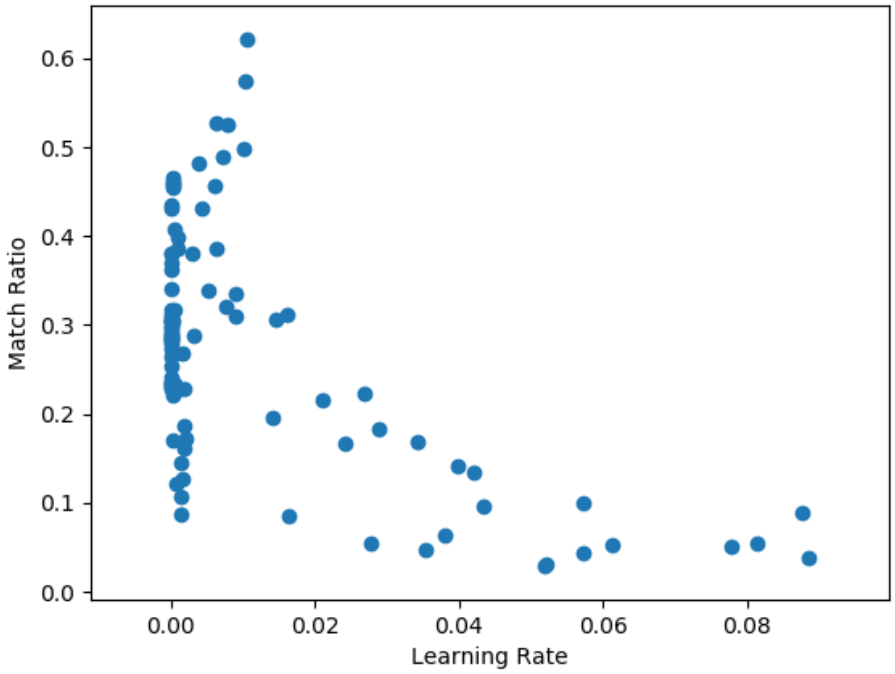


Figure 9: Learning rate stage 1 within the range (-5, -1) concerning update match ratio

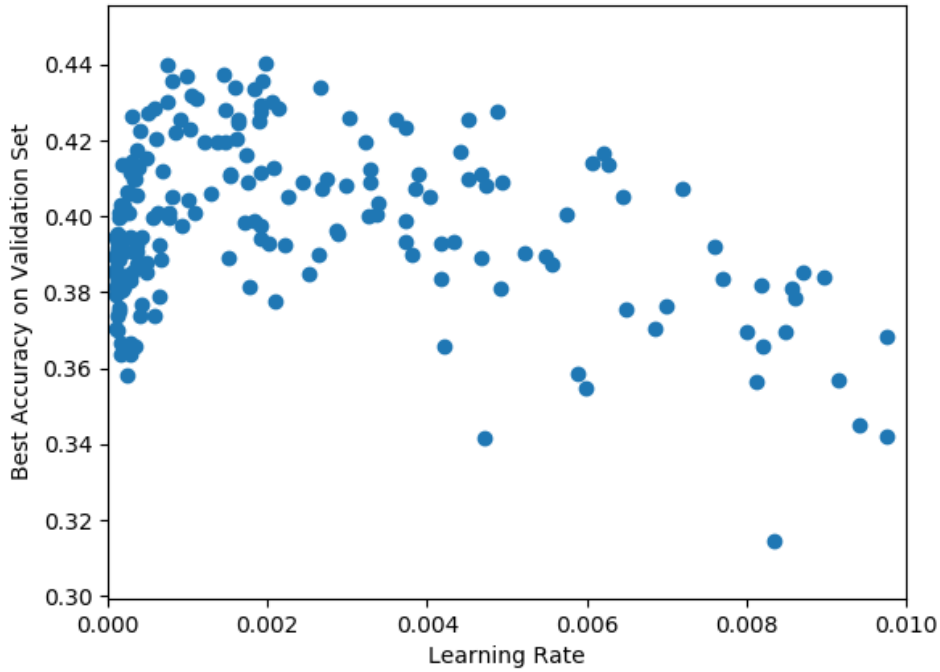


Figure 10: Learning rate stage 2 within the range (-4, -2) concerning accuracy

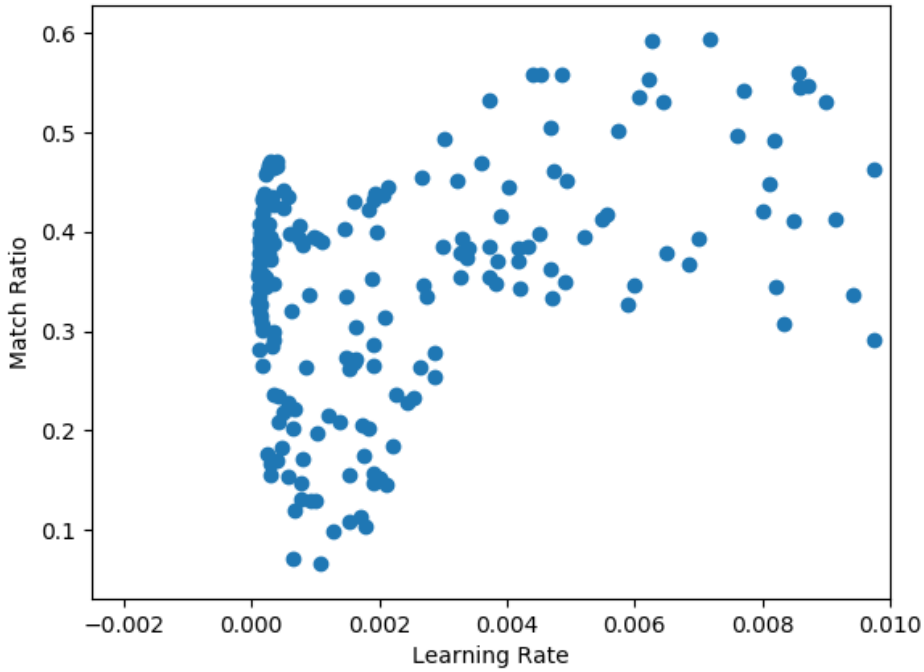


Figure 11: Learning rate stage 2 within the range (-4, -2) concerning update match ratio

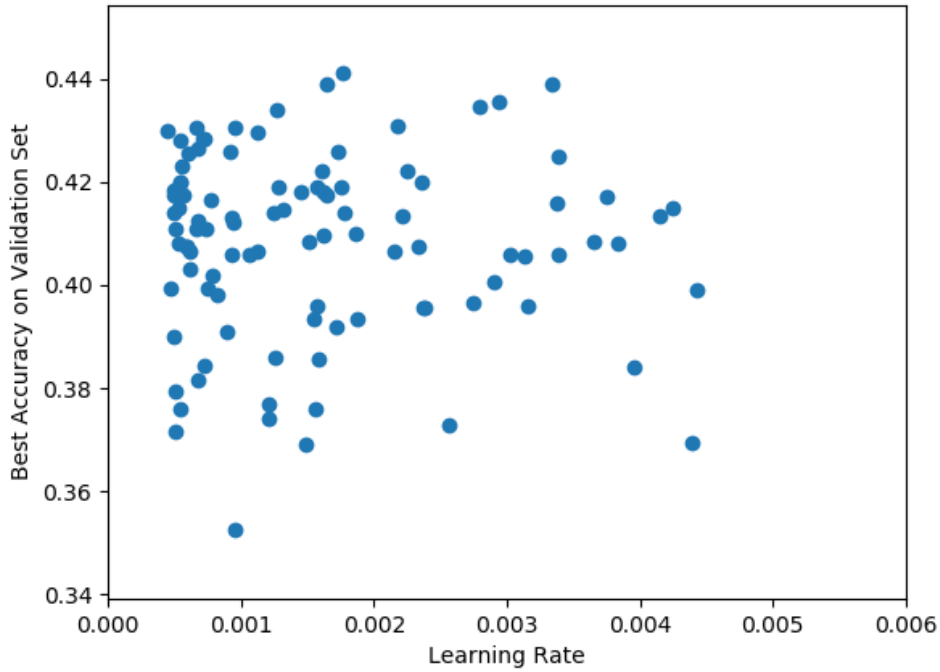


Figure 12: Learning rate stage 3 within the range (-3.35, -2.25) concerning accuracy

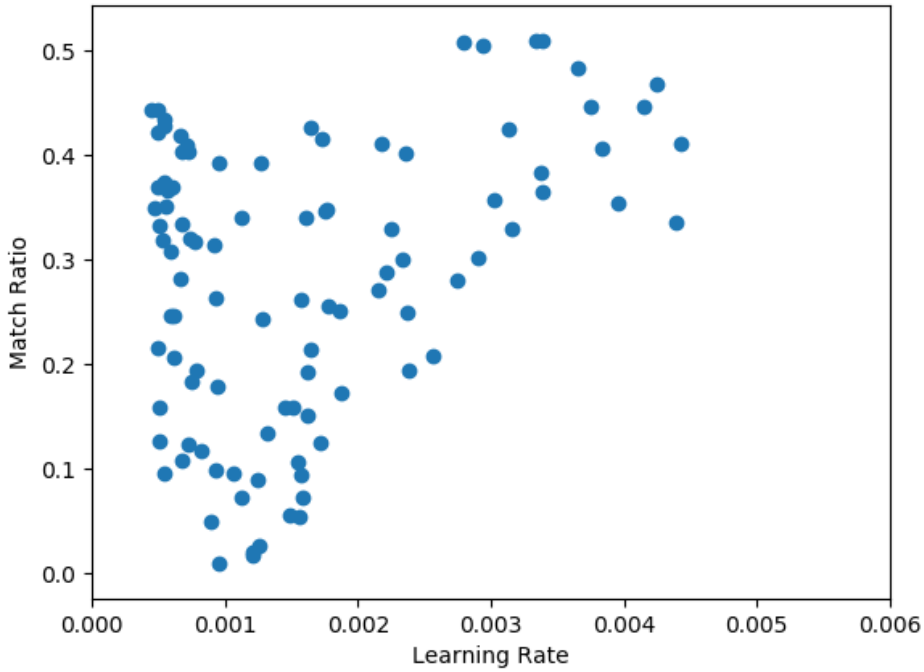


Figure 13: Learning rate stage 3 within the range (-3.35, -2.25) concerning update match ratio

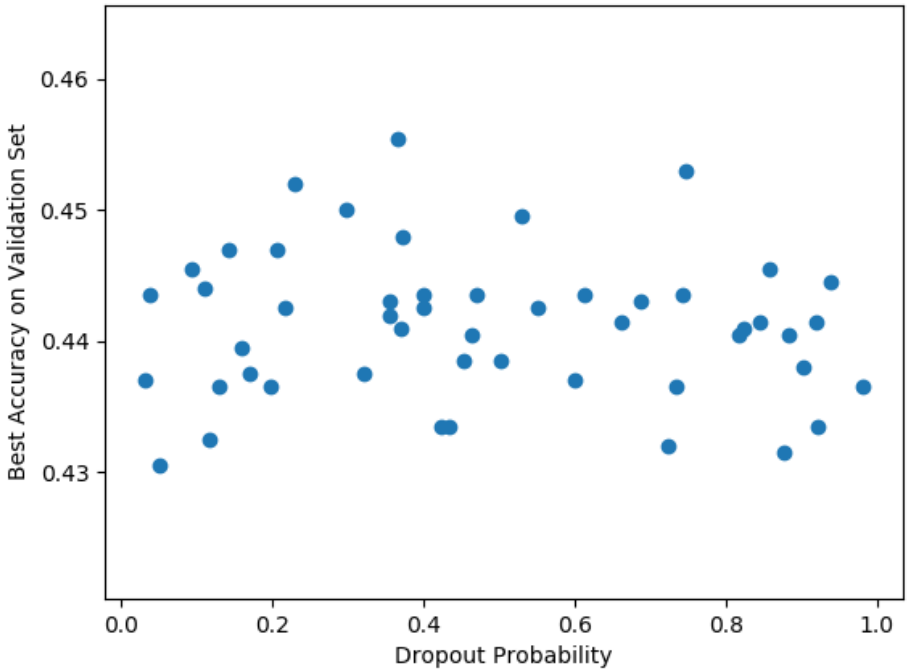


Figure 14: Dropout searching stage 1 within the range (0, 1)

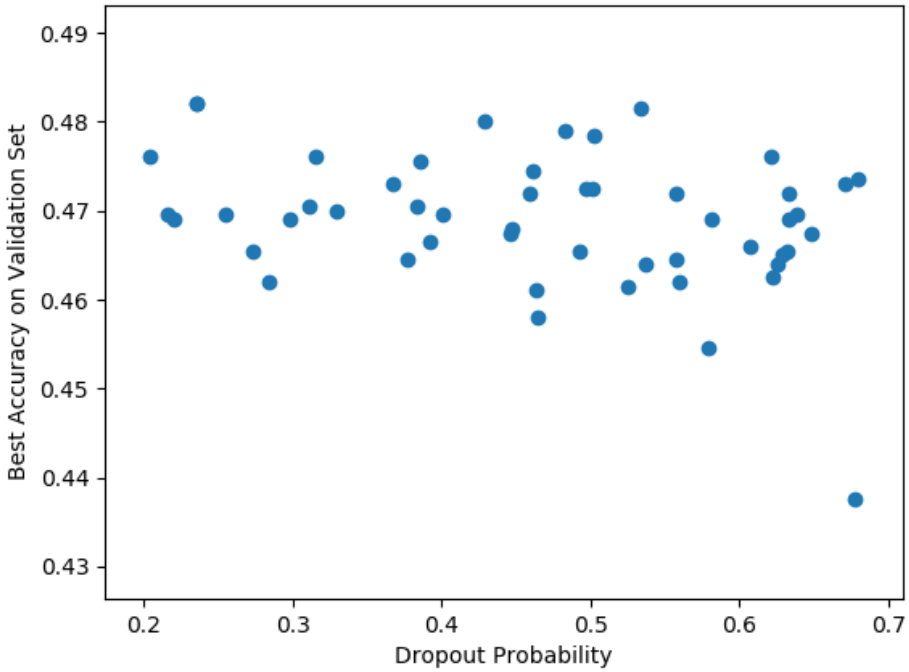


Figure 15: Dropout searching stage 2 within the range (0.1835, 0.6835)

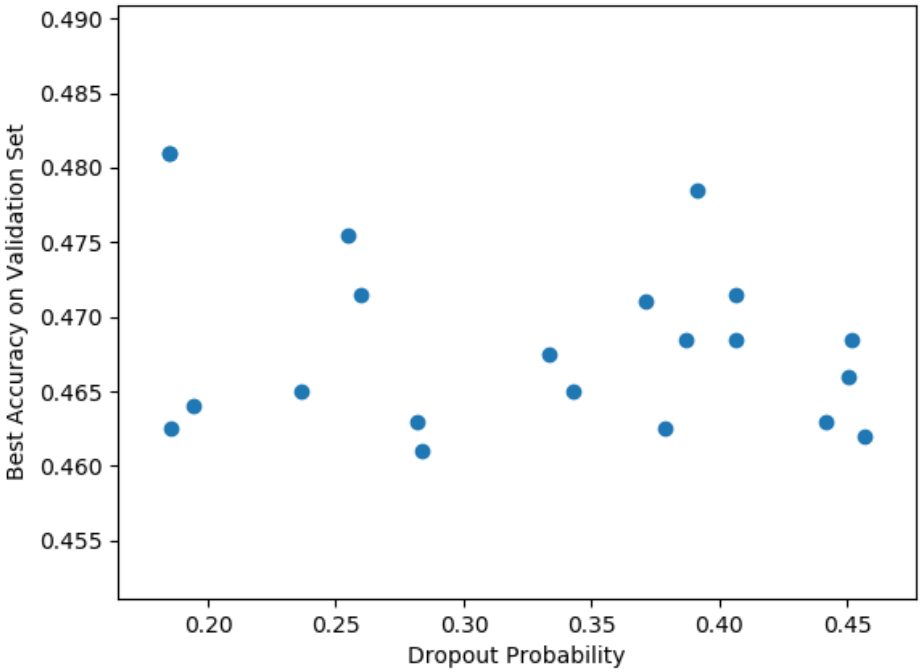


Figure 16: Dropout searching stage 3 within the range (0.1835, 0.4594)

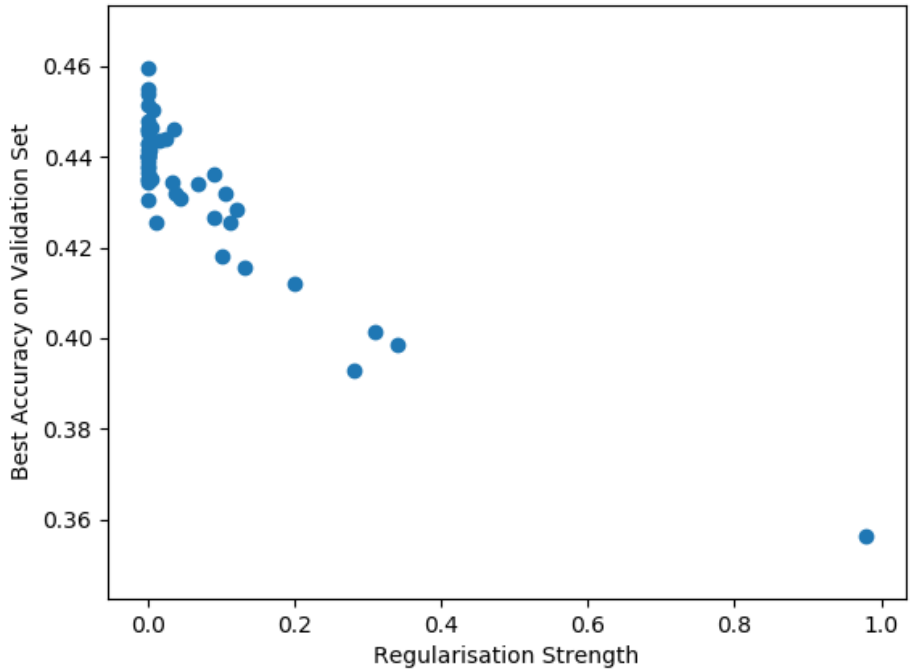


Figure 17: Regularization strength stage 1 searching within the range (-6, 0)

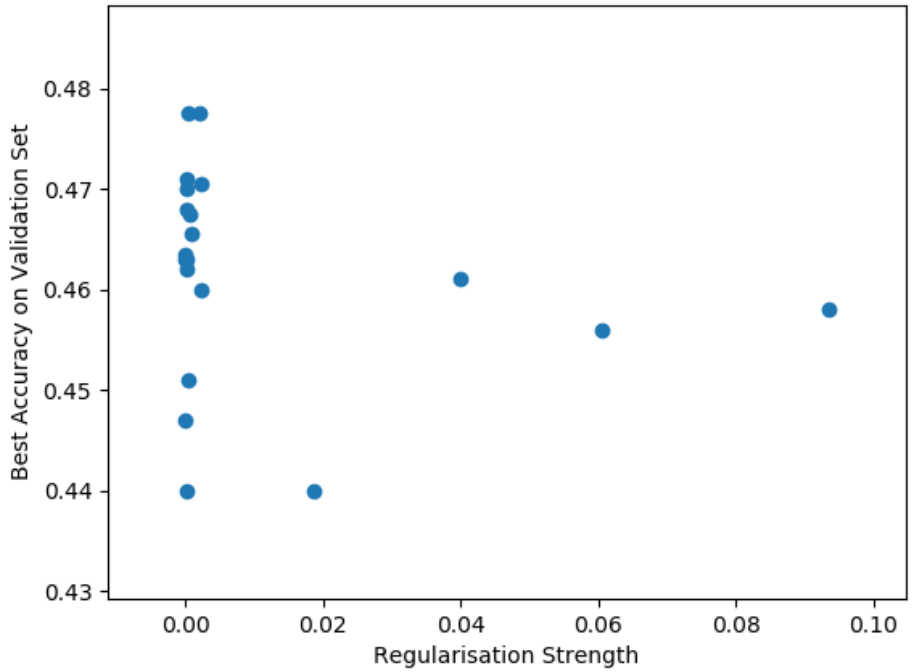


Figure 18: Regularization strength searching stage 2 within the range (-6, -1)

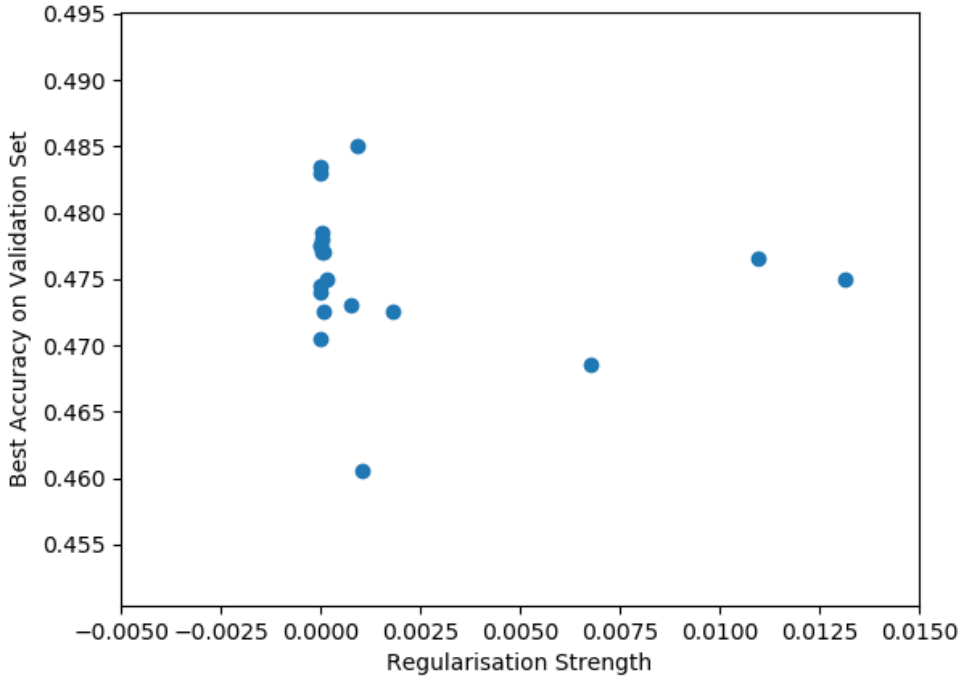


Figure 19: Regularization strength searching stage 3 within the range (-6, -1.842)

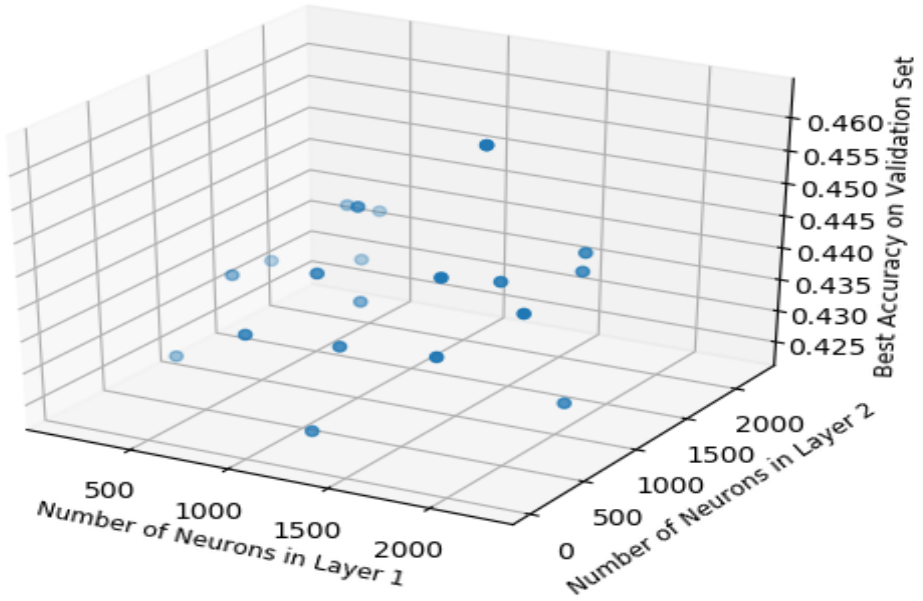


Figure 20: Architecture searching stage 1 within the range (7, 2304)

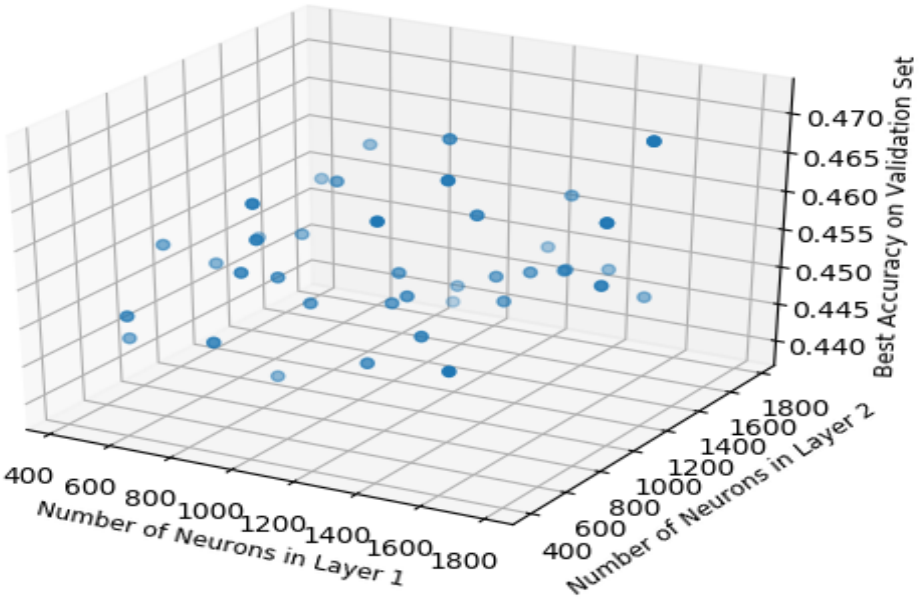


Figure 21: Architecture searching stage 2 within the range (400, 1800)

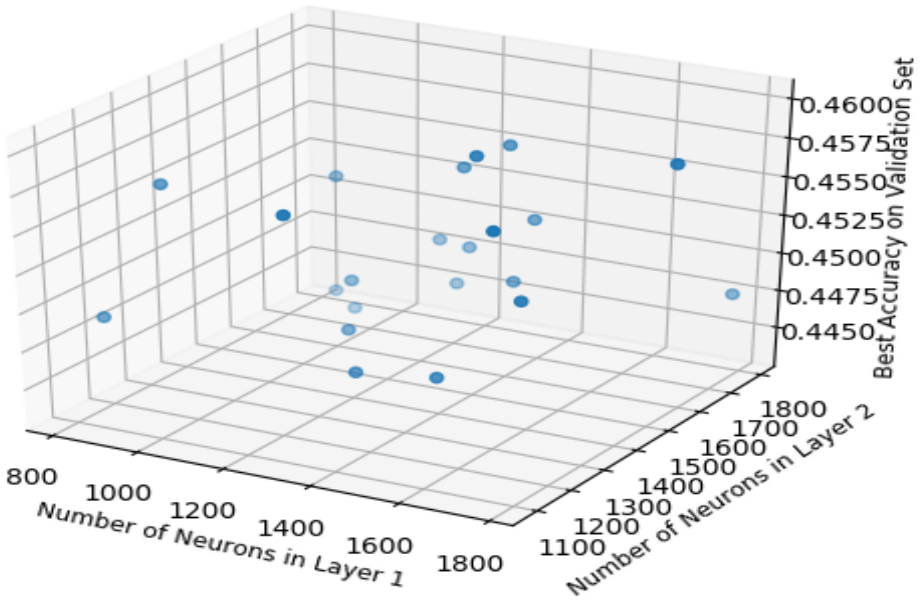


Figure 22: Architecture searching stage 3 within the range (800, 1800)

B Architecture of CNNs

B.1 The architecture of the 4-conv-layer convolutional neural network

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 48, 48, 32)        320
_________________________________________________________________
activation_1 (Activation)    (None, 48, 48, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 46, 46, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 46, 46, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 23, 23, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 21, 21, 64)        36928
_________________________________________________________________
activation_4 (Activation)    (None, 21, 21, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 10, 10, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6400)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3277312
_________________________________________________________________
activation_5 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 3591
_________________________________________________________________
activation_6 (Activation)    (None, 7)                 0
=================================================================
Total params: 3,345,895
Trainable params: 3,345,895
Non-trainable params: 0
_________________________________________________________________
3589/3589 [==============================] - 11s 3ms/step
Test loss: 1.0654972126703801
Test accuracy: 0.6065756478459764
```

Figure 23: The architecture of the 4-conv-layer convolutional neural network

B.2 The architecture of the 6-conv-layer convolutional neural network

```
=================================================================
conv2d_1 (Conv2D)            (None, 48, 48, 32)        320
_________________________________________________________________
activation_1 (Activation)    (None, 48, 48, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 47, 47, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 47, 47, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 47, 47, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 46, 46, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 46, 46, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 46, 46, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 46, 46, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 23, 23, 64)        36928
_________________________________________________________________
activation_4 (Activation)    (None, 23, 23, 64)        0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 11, 11, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 11, 11, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 11, 11, 128)       73856
_________________________________________________________________
activation_5 (Activation)    (None, 11, 11, 128)       0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 5, 5, 128)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 5, 5, 128)         147584
_________________________________________________________________
activation_6 (Activation)    (None, 5, 5, 128)         0
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 2, 2, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
activation_7 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 3591
_________________________________________________________________
activation_8 (Activation)    (None, 7)                 0
=================================================================
Total params: 552,679
Trainable params: 552,679
Non-trainable params: 0
_________________________________________________________________
3589/3589 [==============================] - 18s 5ms/step
Test loss: 1.1495479338014316
Test accuracy: 0.5592086932334637
```

Figure 24: The architecture of the 6-conv-layer convolutional neural network

B.3 The architecture of the residual neural network

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 48, 48, 1)	0	
conv2d_1 (Conv2D)	(None, 48, 48, 16)	160	input_1[0][0]
batch_normalization_1 (BatchNor	(None, 48, 48, 16)	64	conv2d_1[0][0]
activation_1 (Activation)	(None, 48, 48, 16)	0	batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, 48, 48, 16)	2320	activation_1[0][0]
batch_normalization_2 (BatchNor	(None, 48, 48, 16)	64	conv2d_2[0][0]
activation_2 (Activation)	(None, 48, 48, 16)	0	batch_normalization_2[0][0]
conv2d_3 (Conv2D)	(None, 48, 48, 16)	2320	activation_2[0][0]
batch_normalization_3 (BatchNor	(None, 48, 48, 16)	64	conv2d_3[0][0]
add_1 (Add)	(None, 48, 48, 16)	0	activation_1[0][0] batch_normalization_3[0][0]
activation_3 (Activation)	(None, 48, 48, 16)	0	add_1[0][0]
conv2d_4 (Conv2D)	(None, 48, 48, 16)	2320	activation_3[0][0]
batch_normalization_4 (BatchNor	(None, 48, 48, 16)	64	conv2d_4[0][0]
activation_4 (Activation)	(None, 48, 48, 16)	0	batch_normalization_4[0][0]
conv2d_5 (Conv2D)	(None, 48, 48, 16)	2320	activation_4[0][0]
batch_normalization_5 (BatchNor	(None, 48, 48, 16)	64	conv2d_5[0][0]
add_2 (Add)	(None, 48, 48, 16)	0	activation_3[0][0] batch_normalization_5[0][0]
activation_5 (Activation)	(None, 48, 48, 16)	0	add_2[0][0]
conv2d_6 (Conv2D)	(None, 48, 48, 16)	2320	activation_5[0][0]
batch_normalization_6 (BatchNor	(None, 48, 48, 16)	64	conv2d_6[0][0]
activation_6 (Activation)	(None, 48, 48, 16)	0	batch_normalization_6[0][0]
conv2d_7 (Conv2D)	(None, 48, 48, 16)	2320	activation_6[0][0]
batch_normalization_7 (BatchNor	(None, 48, 48, 16)	64	conv2d_7[0][0]
add_3 (Add)	(None, 48, 48, 16)	0	activation_5[0][0] batch_normalization_7[0][0]

activation_7 (Activation)	(None, 48, 48, 16)	0	add_3[0][0]
conv2d_8 (Conv2D)	(None, 24, 24, 32)	4640	activation_7[0][0]
batch_normalization_8 (BatchNor	(None, 24, 24, 32)	128	conv2d_8[0][0]
activation_8 (Activation)	(None, 24, 24, 32)	0	batch_normalization_8[0][0]
conv2d_9 (Conv2D)	(None, 24, 24, 32)	9248	activation_8[0][0]
conv2d_10 (Conv2D)	(None, 24, 24, 32)	544	activation_7[0][0]
batch_normalization_9 (BatchNor	(None, 24, 24, 32)	128	conv2d_9[0][0]
add_4 (Add)	(None, 24, 24, 32)	0	conv2d_10[0][0] batch_normalization_9[0][0]
activation_9 (Activation)	(None, 24, 24, 32)	0	add_4[0][0]
conv2d_11 (Conv2D)	(None, 24, 24, 32)	9248	activation_9[0][0]
batch_normalization_10 (BatchNo	(None, 24, 24, 32)	128	conv2d_11[0][0]
activation_10 (Activation)	(None, 24, 24, 32)	0	batch_normalization_10[0][0]
conv2d_12 (Conv2D)	(None, 24, 24, 32)	9248	activation_10[0][0]
batch_normalization_11 (BatchNo	(None, 24, 24, 32)	128	conv2d_12[0][0]
add_5 (Add)	(None, 24, 24, 32)	0	activation_9[0][0] batch_normalization_11[0][0]
activation_11 (Activation)	(None, 24, 24, 32)	0	add_5[0][0]
conv2d_13 (Conv2D)	(None, 24, 24, 32)	9248	activation_11[0][0]
batch_normalization_12 (BatchNo	(None, 24, 24, 32)	128	conv2d_13[0][0]
activation_12 (Activation)	(None, 24, 24, 32)	0	batch_normalization_12[0][0]
conv2d_14 (Conv2D)	(None, 24, 24, 32)	9248	activation_12[0][0]
batch_normalization_13 (BatchNo	(None, 24, 24, 32)	128	conv2d_14[0][0]
add_6 (Add)	(None, 24, 24, 32)	0	activation_11[0][0] batch_normalization_13[0][0]

```
activation_13 (Activation)      (None, 24, 24, 32)   0           add_6[0][0]
conv2d_15 (Conv2D)              (None, 12, 12, 64)   18496       activation_13[0][0]
batch_normalization_14 (BatchNo (None, 12, 12, 64)   256         conv2d_15[0][0]
activation_14 (Activation)      (None, 12, 12, 64)   0           batch_normalization_14[0][0]
conv2d_16 (Conv2D)              (None, 12, 12, 64)   36928       activation_14[0][0]
conv2d_17 (Conv2D)              (None, 12, 12, 64)   2112        activation_13[0][0]
batch_normalization_15 (BatchNo (None, 12, 12, 64)   256         conv2d_16[0][0]
add_7 (Add)                     (None, 12, 12, 64)   0           conv2d_17[0][0]
                                                                 batch_normalization_15[0][0]
activation_15 (Activation)      (None, 12, 12, 64)   0           add_7[0][0]
conv2d_18 (Conv2D)              (None, 12, 12, 64)   36928       activation_15[0][0]
batch_normalization_16 (BatchNo (None, 12, 12, 64)   256         conv2d_18[0][0]
activation_16 (Activation)      (None, 12, 12, 64)   0           batch_normalization_16[0][0]
conv2d_19 (Conv2D)              (None, 12, 12, 64)   36928       activation_16[0][0]
batch_normalization_17 (BatchNo (None, 12, 12, 64)   256         conv2d_19[0][0]
add_8 (Add)                     (None, 12, 12, 64)   0           activation_15[0][0]
                                                                 batch_normalization_17[0][0]
activation_17 (Activation)      (None, 12, 12, 64)   0           add_8[0][0]
conv2d_20 (Conv2D)              (None, 12, 12, 64)   36928       activation_17[0][0]
batch_normalization_18 (BatchNo (None, 12, 12, 64)   256         conv2d_20[0][0]
activation_18 (Activation)      (None, 12, 12, 64)   0           batch_normalization_18[0][0]
conv2d_21 (Conv2D)              (None, 12, 12, 64)   36928       activation_18[0][0]
batch_normalization_19 (BatchNo (None, 12, 12, 64)   256         conv2d_21[0][0]
add_9 (Add)                     (None, 12, 12, 64)   0           activation_17[0][0]
                                                                 batch_normalization_19[0][0]
activation_19 (Activation)      (None, 12, 12, 64)   0           add_9[0][0]
average_pooling2d_1 (AveragePoo (None, 1, 1, 64)     0           activation_19[0][0]
flatten_1 (Flatten)             (None, 64)           0           average_pooling2d_1[0][0]
dense_1 (Dense)                 (None, 7)            455         flatten_1[0][0]
==================================================================================================
Total params: 273,959
Trainable params: 272,583
Non-trainable params: 1,376
__________________________________________________________________________________________________
3589/3589 [==============================] - 23s 6ms/step
Test loss: 1.2483886403936502
Test accuracy: 0.6628587350568986
3589/3589 [==============================] - 22s 6ms/step
Confusion matrix: [[297   2  45  20  56  11  36]
 [ 18  29   0   2   5   1   1]
 [ 65   2 226  16  97  38  52]
 [ 23   0  19 760  14  19  60]
 [ 80   0  69  29 355  14 106]
 [  9   0  35  17   8 337   9]
 [ 45   2  36  53  89   7 375]]
```

Figure 25: The architecture of the residual neural network