INFOMCV Assignment 5
Action Recognition with Automatic Model Search

Lu Guowei (6374395)   liu zhaopeng (6516637)

April 14, 2019

1 Introduction

In task A, we created our own convolutional neural network with Keras. The structure of out CNN was inspired by $AlexNet$. We also applied the data augmentation methods to avoid overfitting. After that, we looked into details and analyzed our classification result. The accuracy is around 0.3.

In task B, we applied learning rate decay with an exponential function. We applied weight decay by setting a parameter of Keras. We used transfer learning from $MobileNet$. These methods significantly improved the accuracy to 0.615.

In task C, we designed and implemented our own automatic model search algorithm. The algorithm requires several training processes in each step of iteration, thus very computationally heavy. We ran the algorithm and found a pair of settings which could improve the accuracy to 0.691. We analyzed the limitation of this algorithm and assumed that the algorithm could be better if overfitting and underfitting is considered as input, but did not verify it.

2 Task A. Classification on Stanford-40

2.1 Network architecture

The CNN network we designed is shown in figure 1. There are 3 convolutional layers and 3 corresponding batch normalization layers and max pooling layers after them. There are also 2 dense layers on the top of the neural network. In total, there are 8 layers.

We referenced $AlexNet$, and manipulated it to satisfy the limitation of 8 layers to design our architecture.

Then, considering the scale of the data set, which contains 4000 training samples and 5532 test samples, we thought 3 Conv2D layers should be enough for feature extraction. We followed the design of $AlexNet$ whose filter sizes are $(11, 11, 96/s)$, $(5, 5, 384/s)$ and $(3, 3, 256/s)$, where s is a scalar to be selected. We could fine tune the variable s to reach the point where both underfitting and overfitting are prevented.

We fine tuned the neural network, and found 4 to be a good choice for the scalar s. Batch normalization was also found an effective way to suppress overfitting. As a necessity of feature extraction, we created two max pooling layers.

After the convolutional layers is a flatten operator. Its function is to flatten the 3-dimensional images into 1-dimensional series. After that, we added 2 dense and dropout layers to generate labels of 40 classes as output.

2.2 Accuracy

The first time we trained the model was with the original data and without any additional methods. We witnessed overfitting after several epochs. We distinguished overfitting by the phenomenon that training accuracy keeps increasing while test accuracy stops increasing, even decreases.

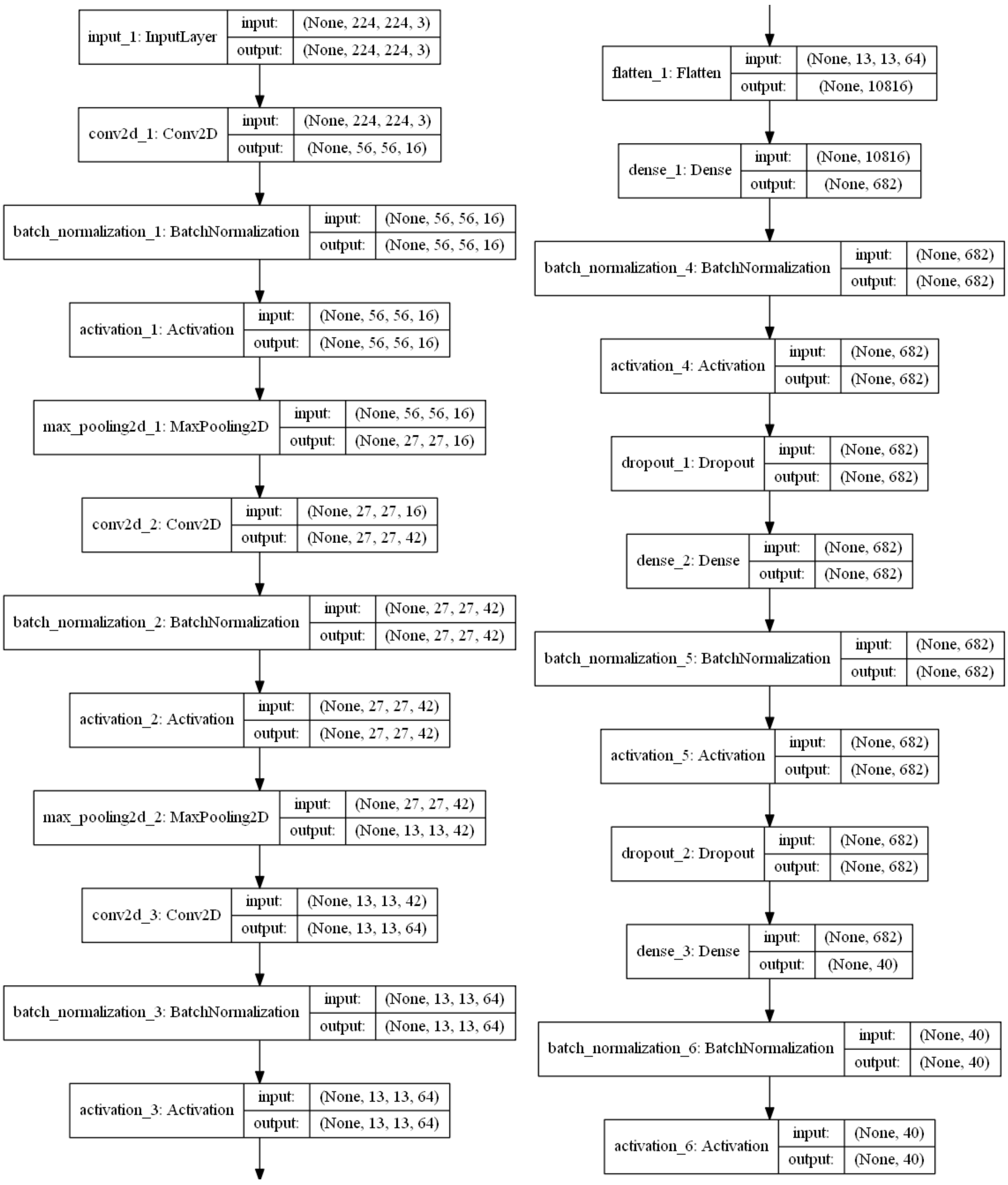
input_1: InputLayer | input: (None, 224, 224, 3) | output: (None, 224, 224, 3)
conv2d_1: Conv2D | input: (None, 224, 224, 3) | output: (None, 56, 56, 16)
batch_normalization_1: BatchNormalization | input: (None, 56, 56, 16) | output: (None, 56, 56, 16)
activation_1: Activation | input: (None, 56, 56, 16) | output: (None, 56, 56, 16)
max_pooling2d_1: MaxPooling2D | input: (None, 56, 56, 16) | output: (None, 27, 27, 16)
conv2d_2: Conv2D | input: (None, 27, 27, 16) | output: (None, 27, 27, 42)
batch_normalization_2: BatchNormalization | input: (None, 27, 27, 42) | output: (None, 27, 27, 42)
activation_2: Activation | input: (None, 27, 27, 42) | output: (None, 27, 27, 42)
max_pooling2d_2: MaxPooling2D | input: (None, 27, 27, 42) | output: (None, 13, 13, 42)
conv2d_3: Conv2D | input: (None, 13, 13, 42) | output: (None, 13, 13, 64)
batch_normalization_3: BatchNormalization | input: (None, 13, 13, 64) | output: (None, 13, 13, 64)
activation_3: Activation | input: (None, 13, 13, 64) | output: (None, 13, 13, 64)
flatten_1: Flatten | input: (None, 13, 13, 64) | output: (None, 10816)
dense_1: Dense | input: (None, 10816) | output: (None, 682)
batch_normalization_4: BatchNormalization | input: (None, 682) | output: (None, 682)
activation_4: Activation | input: (None, 682) | output: (None, 682)
dropout_1: Dropout | input: (None, 682) | output: (None, 682)
dense_2: Dense | input: (None, 682) | output: (None, 682)
batch_normalization_5: BatchNormalization | input: (None, 682) | output: (None, 682)
activation_5: Activation | input: (None, 682) | output: (None, 682)
dropout_2: Dropout | input: (None, 682) | output: (None, 682)
dense_3: Dense | input: (None, 682) | output: (None, 40)
batch_normalization_6: BatchNormalization | input: (None, 40) | output: (None, 40)
activation_6: Activation | input: (None, 40) | output: (None, 40)

Figure 1: Neural Network for Task A

Then we applied the data augmentation methods to overcome overfitting. We compared its accuracy with the accuracy without data augmentation, and found it could improve the accuracy significantly. Specific numbers are shown below.

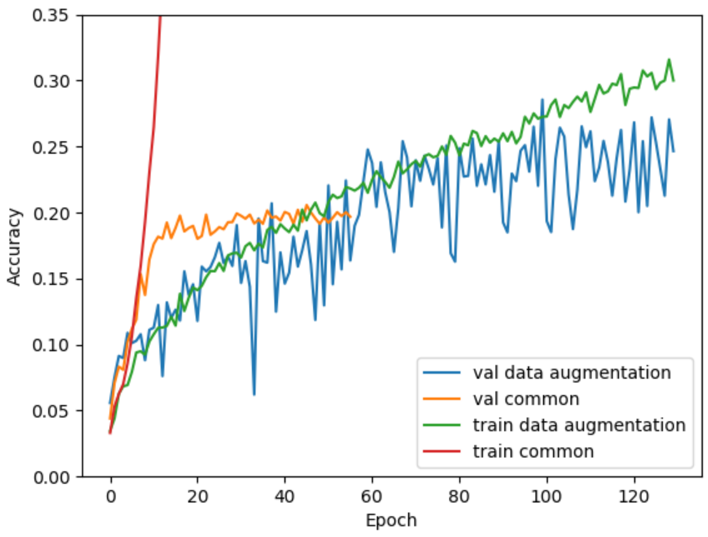

Figure 2: Accuracy at each epoch

Figure 2 shows an example, when data augmentation is applied, the accuracy is about 28%, overfitting began appearing after 101 epochs. If without data augmentation, the final accuracy is 20% and overfitting began appearing after 17 epoch.

2.3 Analysis and conclusion

2.3.1 Result analysis

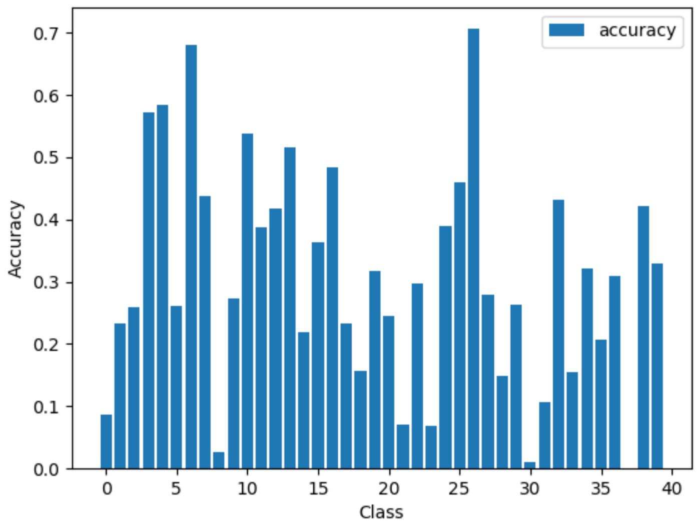

Figure 3: Accuracy of each category

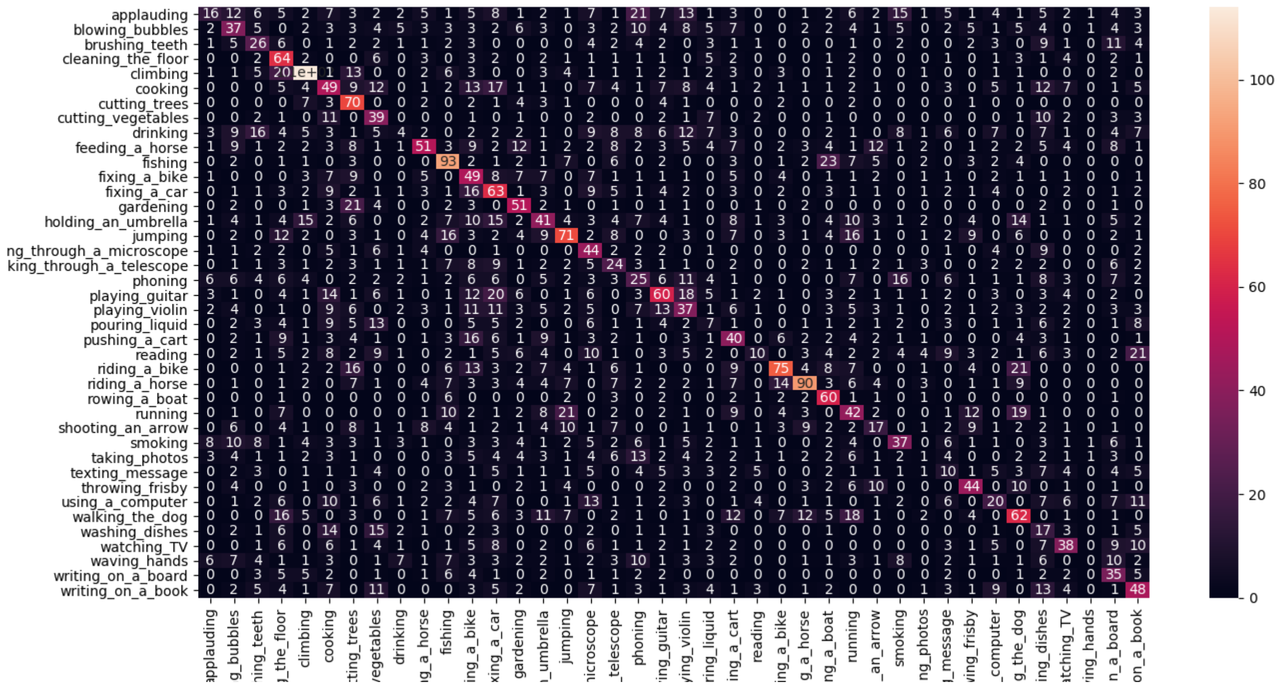

Figure 4: Accuracy of each category

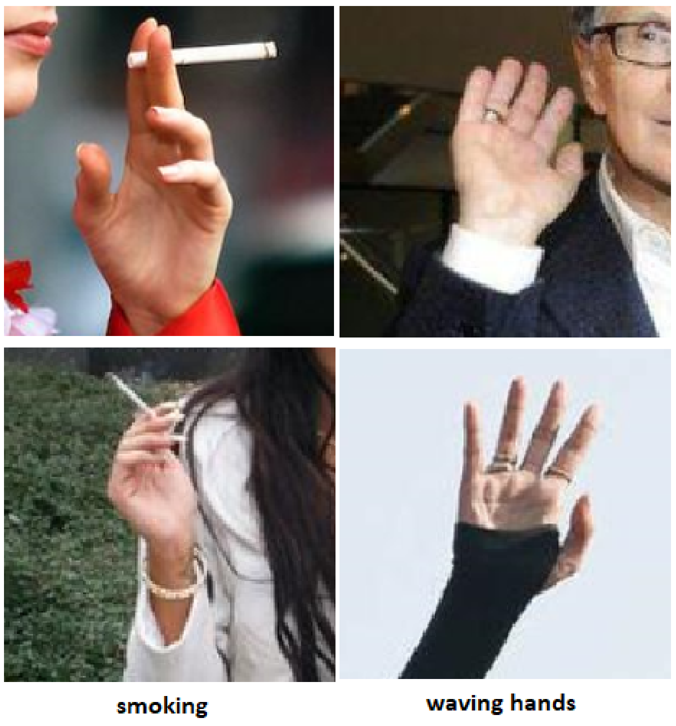

Figure 5: Smoking vs waving hands

Figure 4 is a confusion matrix showing the result of each category. The most accurate two classes are $fishing$ with 93% accuracy and *riding_horse* with 90% accuracy. We looked into the specific images in the data set and tried to find the reoson.

We found that both these two classes have obvious features (human and water, human and horse).

However, owning distinguishable feature is not the only condition. For example, *riding_bike* is also obviously featured by a human and a bike. But their result accuracy are actually not high (accuracy: 75%). The reason is that there is another category named $fixing\_bike$ which is similar.

The two categories with the lowest accuracy are *taking_photos*, 0%, *waving_hands*, 0%, we checked them and found that, *waving_hands* is quite similar with *smoking* and *applauding*, as shown in Figure 5. Another reason is that many people use phones to take photos which could confuse the network.

We conclude that the degree of difficulty for the network to classify a specific image depends on multiple factors. Owning features that are easy to identify and being unique are among the factors.

2.3.2 Data augmentation

Data augmentation is an efficient way to overcome overfitting. We applied rotation, width shift, height shift zoom and horizontal flip, and improved the accuracy from 20% to 28%.

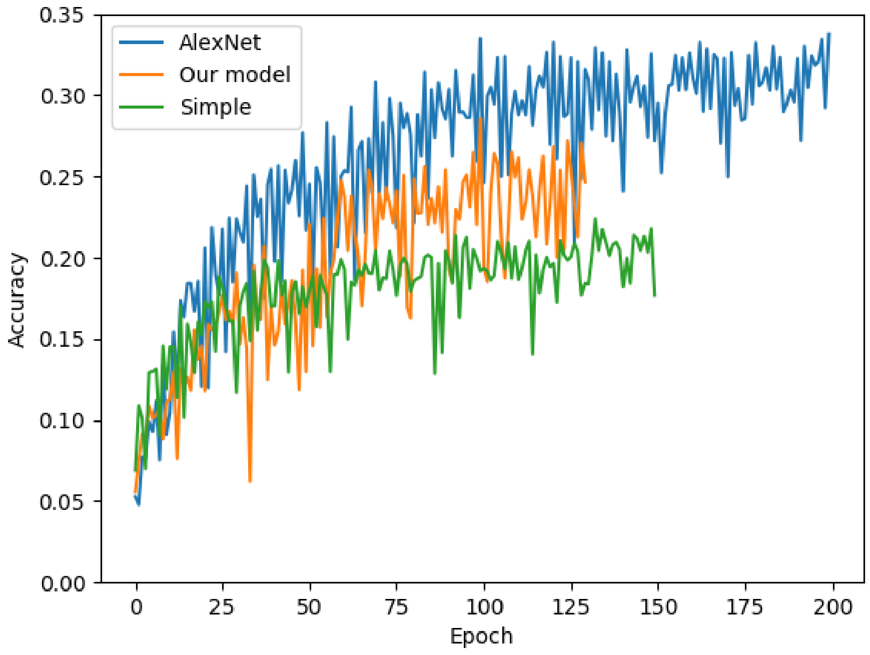

Figure 6: Accuracy among three models

2.3.3 Selection of CNN model

Before deciding on *Alexnet*, we compared three different CNN models. Comparison is shown in Figure 6. The green one is a simple CNN with 2 Conv2D layers, the blue one is a typical $AlexNet$ network with 5 conv2d layers and orange one is the model we designed which has 2 conv2d layers. In our experiment, we first applied a complicated network ($AlexNet$) and a simple network, then based on the requirement, we designed our custom model and fine tuned the parameters. We gained a good accuracy as well as controlling the computation consumption.

However, we found a phenomenon which is hard to explain. We compared one complicated network with one simple one. Theoretically, the more complicated a neural network is, it should be easier to overfit. But in our experiment, the simple network reaches the point of overfitting earlier, which is counter-intuitive.

3 Task B. Using additional methods

3.1 Learning rate decay

We used an exponent function of training iteration i to set the decaying learning rate r. We also limited the learning rate with a minimum value r_{min}. By doing this, we avoid learning rate to become too small. So the learning rate will start with r_{max}, then decay, and eventually get stable at r_{min}.

$$r = \max(r_{max}e^{-\gamma i}, r_{min})$$

r_{max}, r_{min} and γ are configurable parameters. We set $r_{max} = 0.01$, $\gamma = 0.015$ and $r_{min} = 0.001$. At the beginning, $r = 0.01$, it decays during the process of training and gets stable at 0.001.

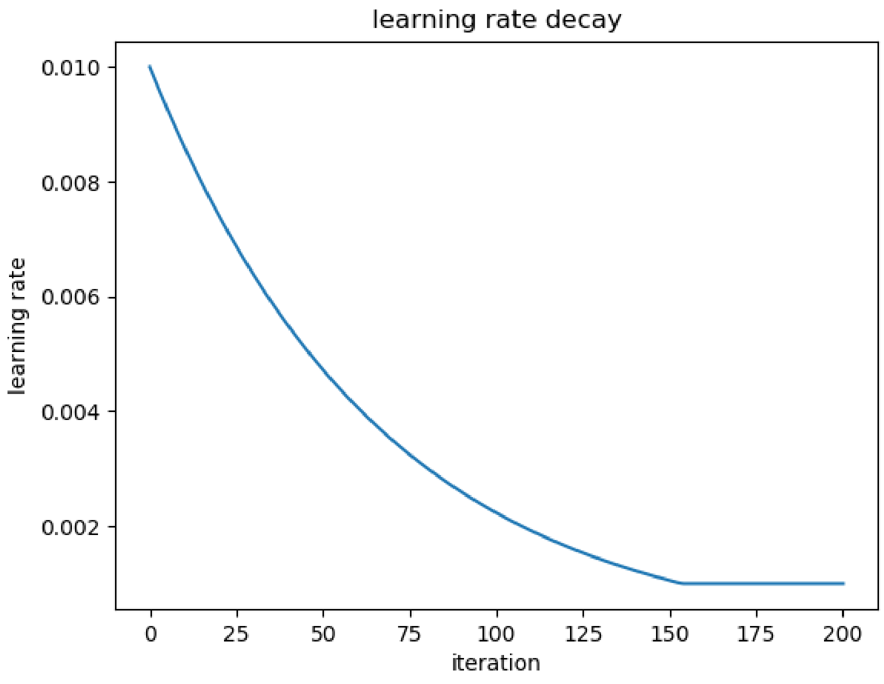

Figure 7: Learning rate decay.

As Figure 8 shows, when we applied the exponent learning function instead of constant learning rate, the learning efficiency is boosted. The fluctuation of the blue anf orange line (with exponent learning rate) is more significant than the green and red line (with constant learning rate, 0.001), but at the end, they had similar accuracy.

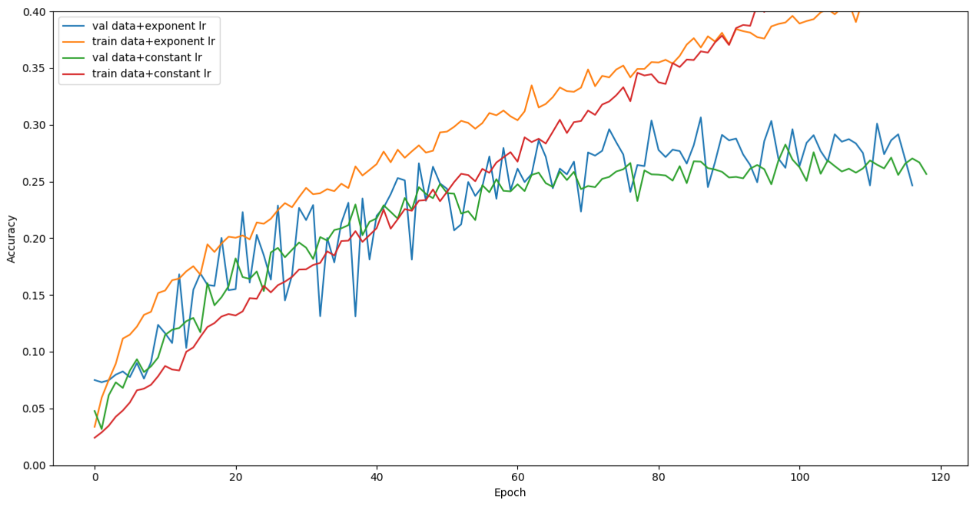

Figure 8: Accuracy between exponent lr and constant lr(0.001).

3.2 Weight decay

In theory, weight decay is a method to avoid overfitting. We applied a early stopping strategy when overfitting is detected. Thus weight decay can lengthen our training process before early stopping.

Figure 9 shows the number of epochs before early stopping with different weight decay values. We can see that when we increase the value of weight decay, overfitting is suppressed and the model is trained for more steps. In this experiment, we did not apply data augmentation method.

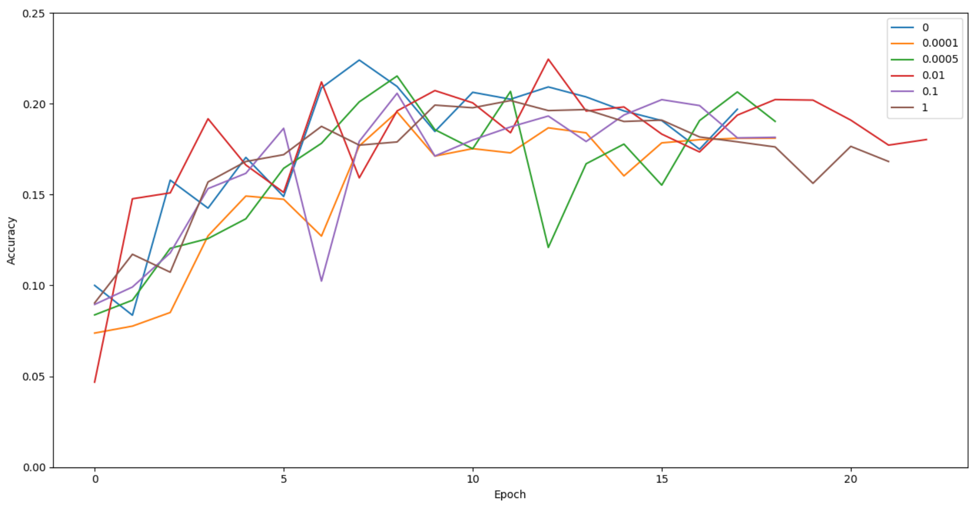

Figure 9: Epoch length before early stopping.

3.3 Transfer learning

We tried several popular CNN architectures, including $VGG16$, $VGG19$ and $MobileNet$. After comparison, we chose $MobileNet$ as our transfer learning model. The main reasons are its simplicity and performance.

We employed $MobileNet$ to extract image features, then added a flatten layer and two dense layers on top of it. We set the weights in all layers of $MobileNets$ fixed and only trained our additional layers. Data augmentation is also applied. Table 1 shows the highest accuracy:

Epochs	Time per Epoch	loss	accuracy	val loss	val accuracy
185/200	102s	1.544	0.555	1.481	0.615

Table 1: The highers accuracy after 200 epoch.

As shown in Figure 10, after 200 epochs, there was still no overfitting. However, the training was time-consuming, each epoch took 102 seconds. So we stopped the training after 200 epochs. We decided to find another smart way to train this model through Task C.

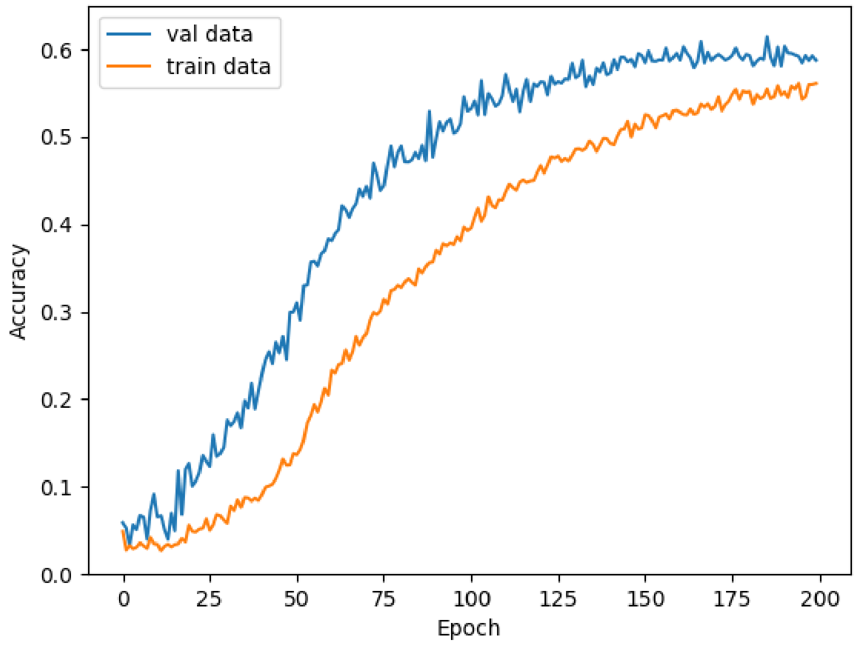

Figure 10: Epoch length before early stopping.

Figure 11 is the confusion matrix of prediction from transfer learning model. Most classes have positive improvement. Specifically, there is an improvement of accuracy in $riding\_bike$ and $fixing\_bike$, which was a problem during Task A.

3.4 Analysis and conclusion

3.4.1 Learning rate decay

As shown in Figure 8, suitable learning rate function is able to improve the learning efficiency. In our selected learning rate decay function, the initial learning rate is 0.01, after 150 epoch, it keeps

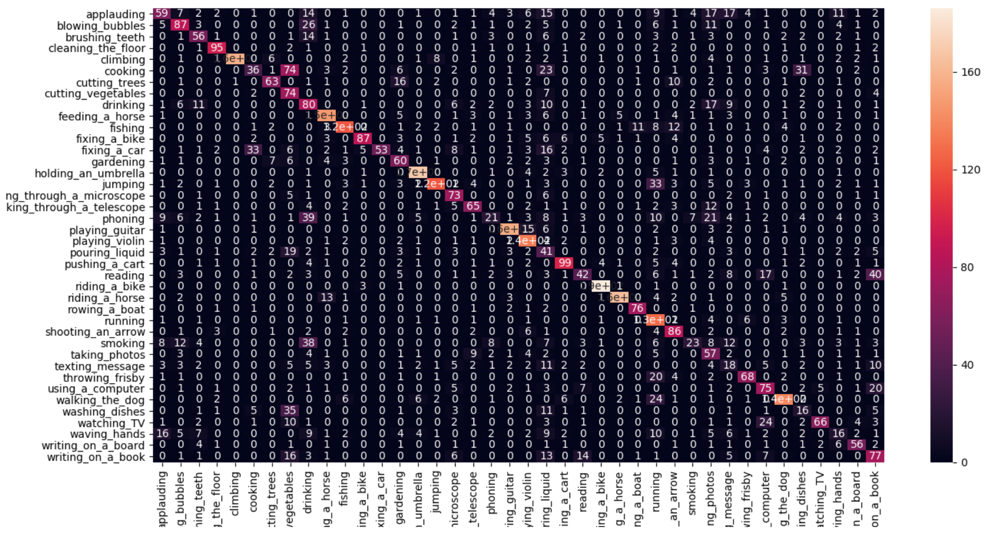
Figure 11: Accuracy of each category.

0.001 as a constant learning rate, based on this requirement, we defined our exponent function. We also tried cyclical learning rates, the result was similar with exponent one.

3.4.2 Weight decay

In theory, weight decay method is a way to suppress overfitting. But in our experiment, the improvement was not obvious.

3.4.3 Transfer learning

We found from our result that transfer learning is a effective way to improve the accuracy.

However, we compared its result with our own custom neural network, and found several interesting issues. As Figure 11 shows, there are many False Positive situations in *cutting_vegetables* and *drinking* classes. So, although this model can improve the accuracy of these two classes, but at the price of a decrease of the accuracy of other classes such as *cooking*.

As shown in Figure 12, the accuracy of *cooking*, *cutting_trees* and *smoking* is better in our model than the transfer learning model. So, maybe a combined model is an idea to solve this issue. After the optimization of Task C, we can see Figure 14, the model itself fixed this issue.

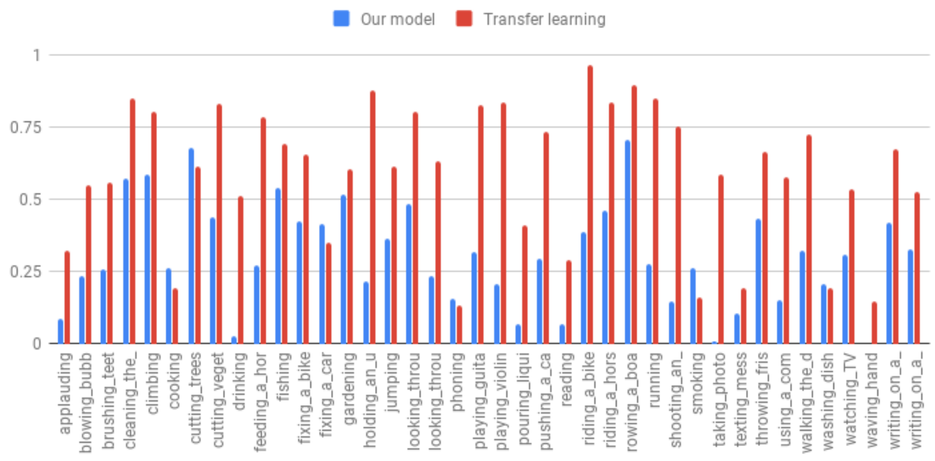

Figure 12: Comparison of accuracy of each category with and without transfer learning.

4 Task C. Finding the best architecture

4.1 Automatic model search algorithm

We built our own method to search for the best model automatically. In our implementation, we set the number of layers in the model fixed, which are two dense layers and the corresponding dropout layers and normalization layers. But we left the number of parameters in the layers changeable for the searching algorithm to decide.

In summary, the algorithm automatically selects two numbers, each represents the number of parameters in one dense layer.

The algorithm initializes by randomly guessing the 2 numbers, n_1 and n_2. Then tries to find the best combination iteratively.

In each iteration, the algorithm tries 5 neighbouring combinations, d is the step size.

$$\begin{pmatrix} & (n_1-d,n_2) & \\ (n_1,n_2-d) & (n_1,n_2) & (n_1,n_2+d) \\ & (n_1+d,n_2) & \end{pmatrix}$$

(n_1, n_2) is then set to one of the candidates according to the their accuracy. The algorithm iterates until maximum accuracy is found or a limitation of iterations is reached. But it is possible that the algorithm gets trapped in local maximum. So we randomly reinitialize the iterations once a local maximum is detected

We also limit (n_1, n_2) by a boundary with intuition and experience. The input of the dense layers has 50176 parameters, the output has 40 parameters. And in task B, we witnessed overfitting with both dense layers owning 1024 parameters. So we limit n_1 and n_2 in the range of $(128, 2048)$ for task C.

Basically, every iteration requires 5 training processes. We optimized it with cached training results, but it still requires a large amount of time.

4.2 Result

The searching process ended after 40 iterations, we recorded n_1, n_2 and accuracy in each iteration.

With 40 steps of iteration, the algorithm found $(1491, 615)$ to be the best setting for (n_1, n_2) with accuracy of 0.691 (temporarily, this number can be better with more iterations).

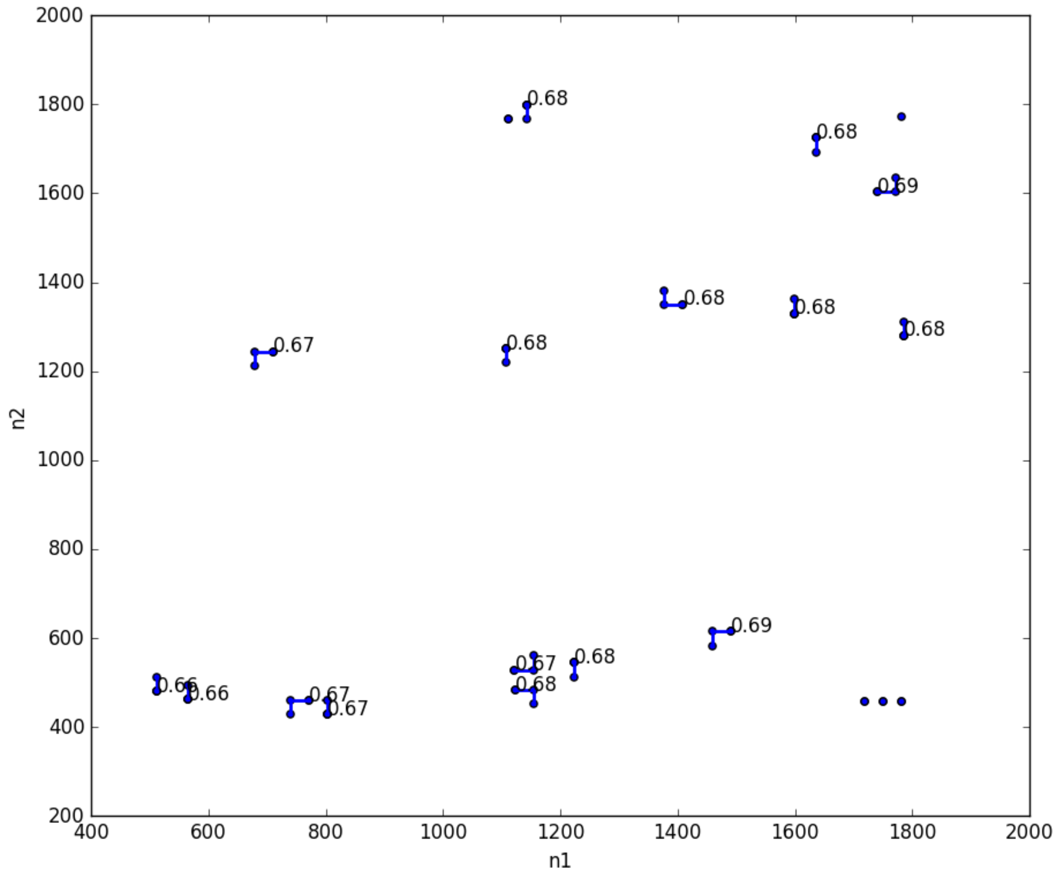

Figure 13: Automatic model searching process.

$(n_1, n_2) = (1491, 615)$ fits our intuition. The first dense layer should be larger because it extracts more specific information. The second dense layer is responsible for extracting more abstract feature, thus smaller.

4.3 Analysis and conclusion

4.3.1 result

As shown in Figure 13, the continuous iterations typically ends within 2 3 steps. Which means it is trapped in local maximum every 2 3 iterations. We randomly restart the searching process to get out from local minimum. But in order to get a more convincing result, more iterations need to be run. That requires a lot more computation and time.

But even with only 40 iterations, we could still see some trends in the graph. In general, the accuracy of points in the right part of the graph is larger than those in the left, which means the number of first dense layer n_1 wants to be larger to get a larger accuracy.

Figure 14 is the test result of the model selected by our automatic model search algorithm, which has 1491 parameters in the first dense layer and 615 parameters in the second dense layer, with overall accuracy being 0.691.

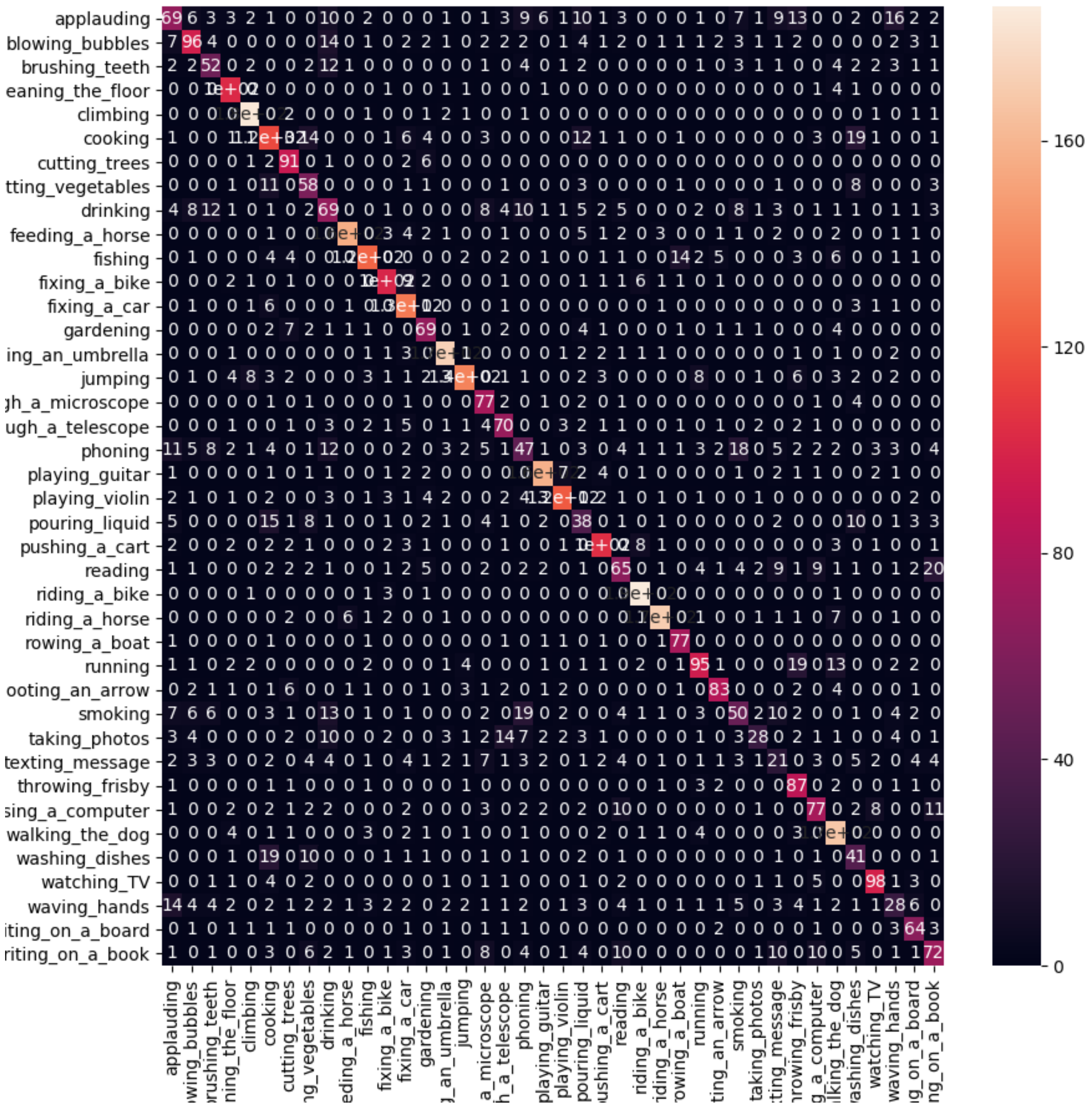

Figure 14: Test result of automatically selected model.

4.3.2 Analysis and limitations

Our algorithm only considered the training accuracy itself, but ignored the facts of overfitting and underfitting. So we designed another model search algorithm, with overfitting and underfitting as additional input.

The new algorithm would also try to find the best settings iteratively. In each iteration, it would first train the model with current (n_1, n_2), and decide if it's overfitting or underfitting.

It would decrease n_1 or n_2 when overfitting and increase n_1 or n_2 when underfitting. If neither overfitting nor underfitting is the case, it would move to the candidate setting with the largest accuracy.

We expect the new algorithm to make use of more information and be less possible to get trapped in local maximum. But we did not verify our idea because of the limited computation power and time.

Another limitation of our algorithm is that it can only slightly manipulate the model by changing the parameter numbers, but cannot add or delete layers.

5 Reference

[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton ImageNet Classification with Deep Convolutional Neural Networks