Report of Final Project: Image Classification

I. Introduction

In recent years, with the development of the field of computer vision, there is an increasing demand for object recognition. Deep Convolutional Neural Networks (CNNs) have made great progress in areas such as image classification and object recognition [1].

The main goal of this project is to build, train, and evaluate a CNN model that can identify whether an image belongs to the tomato, cherry, or strawberry categories. By tuning the CNN model with various hyperparameters to achieve higher accuracy. Meanwhile, to compare with the CNN model, we constructed a simple multilayer perceptron (MLP) model as a baseline.

At first, we performed EDA and preprocessing on the dataset and constructed a Baseline MLP model and a Basic CNN model. A 5-fold cross-validation was used for CNN optimization, and the parameters were tuned for the Basic CNN model with 7 aspects of optimization. A good validation accuracy was finally obtained. The entire project was constructed using Pytorch and executed in a reasonable runtime(Less than 1 hour with the GPU).

II. Problem Investigation

The dataset used in this project is an image categorization dataset consisting of a total of 4,500 RGB images in 3 classes, 1,500 images per class. The 3 categories are tomatoes, cherries and Strawberry, most image size is 300*300. Additionally, images are sourced from Flickr.

A. Conduct exploratory data analysis (EDA).

First, do a general overview of the data set to ensure that the data meets the given conditions, such as whether the classification and size are correct.

By manually scrolling through all the training data, it can be found that there are some noisy pictures in the image set, as shown in Fig. 1. One kind of image is misclassified images, such as tomato images are misclassified as "cherry", and tomato images are misclassified as "strawberry", and so on. Another kind of noise is that there are some pictures that are not related to the classification. For example, pictures of dogs exist in the cherry classification, cakes exist in the tomato classification, etc.

There are also some pictures where more than two categories appear at the same time. For example, as shown in Figure 2, this picture is classified into the cherry category, but there are two fruits, cherry and strawberry, in the same picture. This kind of picture will have a negative effect on the feature learning of the model during training. Because the characteristics of strawberries learned in the cherry category will cause category confusion, making the classification blurrier and reducing the classification efficiency of the model. These noisy images should be removed during the data preprocessing stage.

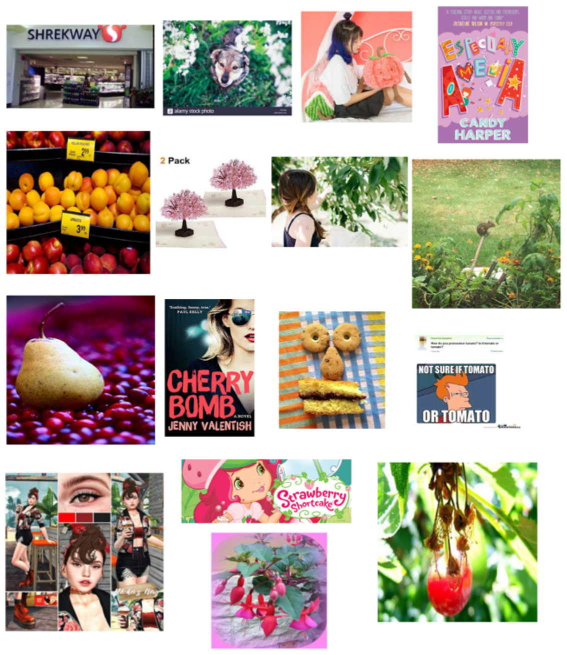


Fig. 1. Some noise pictures

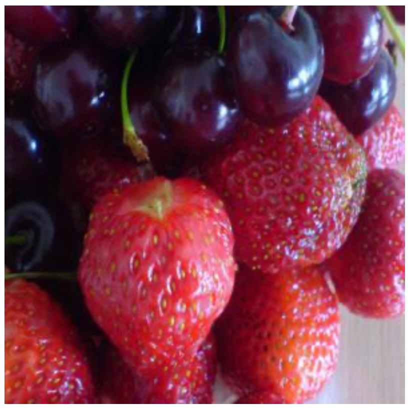
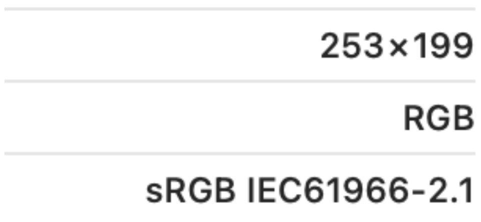


Fig. 2. Images that cause category confusion

In addition, there are some images whose size is not 300*300, as shown in Fig. 3. Since the content of these images is correctly categorized, it is only necessary to change the size during data preprocessing, and there is no need to delete them.


253×199

RGB

sRGB IEC61966-2.1

Fig. 3. Image size not 300*300

There are also some images in the training data that have some of the features missing, some are overall dark colors, and some of the classified items have special shapes or are only partial. These data are still retained because other features can still be used as the basis for classification. For example, dark-colored images still have obvious shape features for classification.

B. *Data Pre-processing:*

First, remove the noisy image. After the above manual filtering, 51 noise pictures were eliminated. In the remaining training data, there are 1479 categories of "cherry", 1487 categories of "strawberry", and 1483 categories of "tomato".

Second, resize and scale the image to 300*300, and convert all images to tensor objects. General images use RGB channels with values ranging from 0 ~ 255. This range is a bit large for training data and needs to be normalized. According to the formula normalized_value = (value - mean) / std, the values of the 3 channels can be ranged between [-1,1] to improve the overall performance. These preprocessing steps are shown in the code in Figure 4.

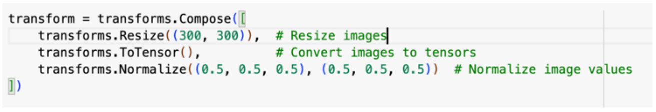

```
transform = transforms.Compose([
    transforms.Resize((300, 300)),  # Resize images
    transforms.ToTensor(),          # Convert images to tensors
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Normalize image values
])
```

Fig. 4. Preprocessing steps

III. Methodology

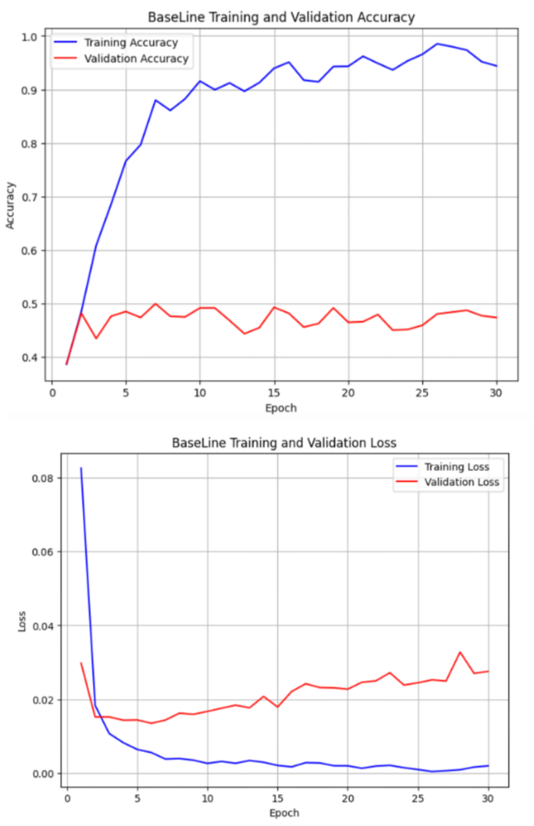
In order to achieve the expected results of the project, we constructed a simple multilayer perceptron (MLP) model as a baseline model in the first step. Then, we built our own CNN model and optimally tuned it with multiple different parameters. The models were trained in a reasonable amount of time (less than 1 hour when running on the GPU).

A. *MLP Baseline Model.*

In this simple baseline model, we construct a standard neural network without CNN. The entire data set was split into a training set and a validation set at a ratio of 8:2, and the training set was shuffled.

This MLP model consists of an input layer (number of neurons 300*300*3, depending on the image size and 3 RGB channels), an output layer (number of neurons 3, depending on the 3 classifications), and two hidden layers (number of neurons 512 and 256). The activation function is ReLU, the loss function is cross-entropy loss, the optimizer chooses Adam (lr=0.001), and the mini-batch size is 100. In order to shorten the execution time, 30 epochs are used. In addition, a softmax function was applied to the output for classification to improve the accuracy of the prediction results.

Fig. 5 shows the classification accuracy and loss vs. epoch of the MLP Baseline model for the training and validation data sets. It can be seen that although the training data exceeds 90% accuracy after more than 10 epochs, the highest accuracy is only 49.89% for the validation data and the loss function tends to increase after 3 epochs. This means that the model is overfitting. Also, it may be due to the fact that the MLP does not extract the features in the image well, thus leading to a low accuracy in the validation set.



```
Validation Accuracy 0.49887640449438203
```

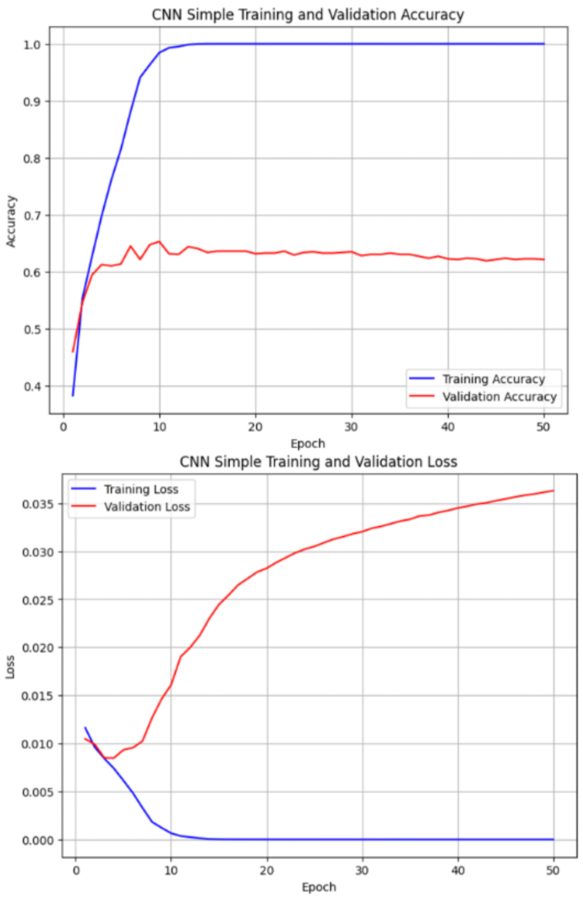
Fig. 5. Accuracy and loss of MLP on training and validation sets

B. *Basic CNN Model*

Based on the lecture and the code shown in the tutorial, we created a basic CNN model.

The basic CNN model still splits the dataset into training and validation sets in the ratio of 8:2 and the batch size is chosen to be 100. Three convolutional layers are temporarily created with the number of nodes being 32, 64 and 128 respectively. The convolution kernels are all 3*3 with a padding of 1. Meanwhile, the pooling layer is added after each convolution with pooling window sizes of 2*2, 2*2, and 3*3, and sliding steps of 2, 2, and 3. Two fully connected layers are created with the number of nodes being 128 and 3. The activation function is also ReLU, the loss function is cross-entropy loss, the optimizer chooses Adam (lr = 0.001), and runs for 50 epochs.

The model training results are shown in Fig. 6. The accuracy of the validation set has improved greatly (49.89% -> 65.28%). However, it is easy to see from the graph that there is still an overfitting problem. Firstly, the training accuracy reaches a very high value (100%) after 10 epochs, and the accuracy of the validation set tends to decrease after about 15 epochs. Therefore, for the current model, the first step is to stop the epoch early to reduce overfitting. Based on the performance of the validation set, we finally chose 15 as the epoch value for further training.



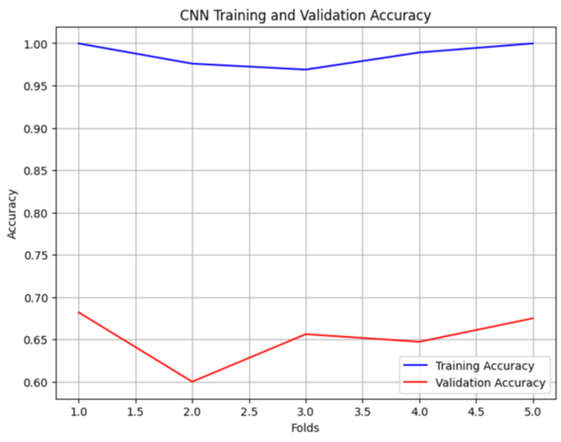
```
accuracy 0.6528089887640449
```

Fig. 6. Accuracy and loss of Basic CNN on training and validation sets

C. *Optimize 1: 5-fold cross-validation*

Use sklearn.model_selection to perform a 5-fold cross-validation of the dataset. The advantage of this is that the average accuracy of the validation set of 5 times can be used instead of the optimal accuracy, thereby obtaining a more accurate performance evaluation. Moreover, during the hyperparameter tuning process, it can better measure the performance between different model parameters and improve the generalization ability of the model.

Fig. 7 shows the results of 5-fold cross-validation of the Basic CNN Model. It can be seen that the average accuracy of the verification set is 65.21%.



```
Average validation accuracy: 0.6520617787945046
```

Fig. 7. Average accuracy of Basic CNN

D. *Optimize 2: The regularisation strategy*

Since the Basic CNN model has a serious over-fitting problem, dropout is added as a regularization strategy to reduce over-fitting.

Regularization plays a key role in reducing overfitting and dropout is the most commonly used regularization technique. It randomly drops a predesigned portion of neuron activity at each iteration to regularize the network [2].

We tested three groups of Dropout in each convolutional layer, respectively p=0.2, p=0.3 and p=0.4, and the other parameters remained unchanged. As shown in Fig. 8, Mean accuracies were 66.73% (p=0.2), 68.24% (p=0.3), and 67.61% (p=0.4) respectively.

Compared to Basic CNN, the average accuracies are all improved, which means that Dropout does reduce overfitting. The highest average accuracy was found at p=0.3, so p = 0.3 was chosen for all following optimizations.

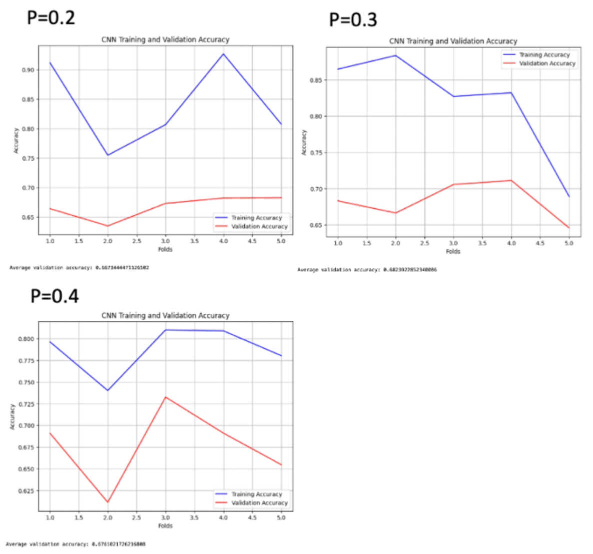


Fig. 8. Average accuracy of different dropout

E. *Optimize 3: Hyper-parameter Settings*

Hyperparameters are the parameters that need to be set when training the model, such as Epochs, Batch Size, Learning Rate, Dropout Rate, and so on.

Most of the experiments and comparisons of hyperparameters are described in their respective chapters, e.g., the Dropout Rate tunning is in §III.D, while the Learning Rate is in §III.G, and so on. This section focuses on the tuning of the number of convolutional layers.

Based on the Base CNN model (3 convolutional layers with 32, 64 and 128 nodes respectively), the number of convolutional layers was tuned and testing. Try building the model with 2 convolutional layers, 4 convolutional layers and 5 convolutional layers. The number of nodes in the 2 convolutional layer is 32 and 64, the number of nodes in the 4 convolutional layer is 32, 64, 128, and 256, and the number of nodes in the 5 convolutional layer is 32, 64, 128, 256, and 512. and the corresponding pooling layer is configured for each convolutional layer.

Figure 9 illustrates the impact of different convolutional layers on the accuracy of the validation set. The average accuracy of the validation set is 56.30% (2 convolutional layers), 75.39% (4 convolutional layers), and 78.26% (5 convolutional layers). And the validation set accuracy of base CNN model with 3 convolutional layers is 68.24%. It can be seen that as the number of convolutional layers increases, the average accuracy of the validation set also increases. At the same time, adding more convolutional layers may also mean an increase in runtime, which is one of the reasons why the number of convolutional layers cannot be too large. Finally, five convolutional layers are chosen to get better generalized model.

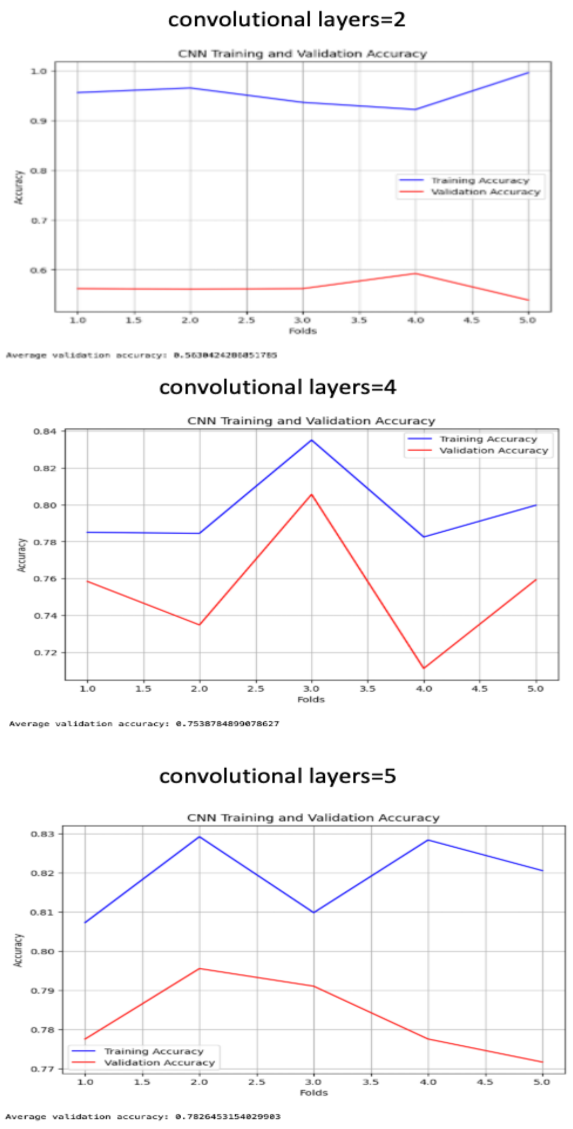


Fig. 9. Average accuracy for different number of convolutional layers

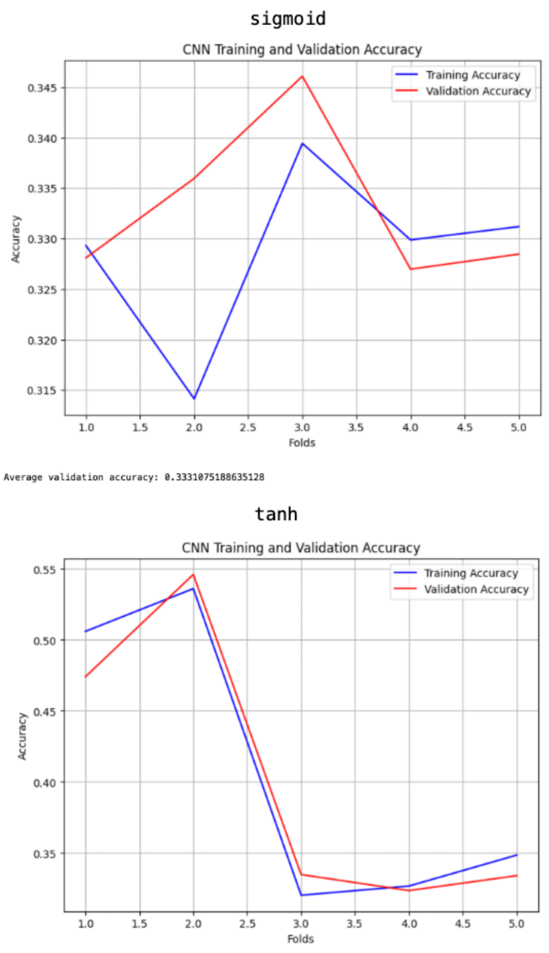
F. *Optimize 4: The Activation Function*

Activation Function (AF) is a nonlinear function that is mainly used to introduce nonlinear properties [3].

In the Base CNN model, we used ReLU as the activation function. Based on all the previous optimizations, we tried two activation functions, sigmoid and tanh. The results are shown in Fig. 10.

The average accuracy of sigmoid's model is 33.31%, while the average accuracy of tanh's model is 40.25%, both of which are far lower than ReLU (78.26%). One possible reason is that tanh and Sigmoid are S-shaped curves. When the input is close to extreme values, the derivative is close to 0, which means that the nonlinear features will become smaller and will cause the vanishing gradient problem.

Finally, ReLU is chosen as the activation function of the model.



```
Average validation accuracy: 0.3331075188635128
```

```
Average validation accuracy: 0.40254698499766184
```

Fig. 10. Average accuracy for different Activation Function

G. *Optimize 5: The Optimisation Method(s)*

In neural network models, the model can be made more generalized by finding model parameters that minimize the loss function. In Basic CNN model, the optimization method we used is Adam with a learning rate of 0.001.

Based on the previous optimized model, we set different learning rates for Adam to test. At the same time, we also set different parameters for two new optimization methods, Stochastic Gradient Descent (SGD) and Root Mean Square Propagation (RMSProp). The experimental results are summarized in Table 1.

It can be seen that after different optimization methods are configured with different parameters, the average accuracy of the model's validation set shows different changes. Finally, the optimization method with the highest accuracy was selected: Adam (lr=0.001).

Optimization	Learning rate	momentum	Average validation accuracy
Adam	0.1	/	33.09%
Adam	0.01	/	33.09%
Adam	0.001	/	78.26%
RMSprop	0.01	0.8	32.84%
RMSprop	0.01	0.9	32.82%
RMSprop	0.001	0.8	41.02%
RMSprop	0.001	0.9	33.13%
SGD	0.01	0.8	54.71%
SGD	0.01	0.9	59.14%
SGD	0.001	0.8	35.33%
SGD	0.001	0.9	36.37%

Table 1. Different Optimisation Methods and Accuracy Comparison

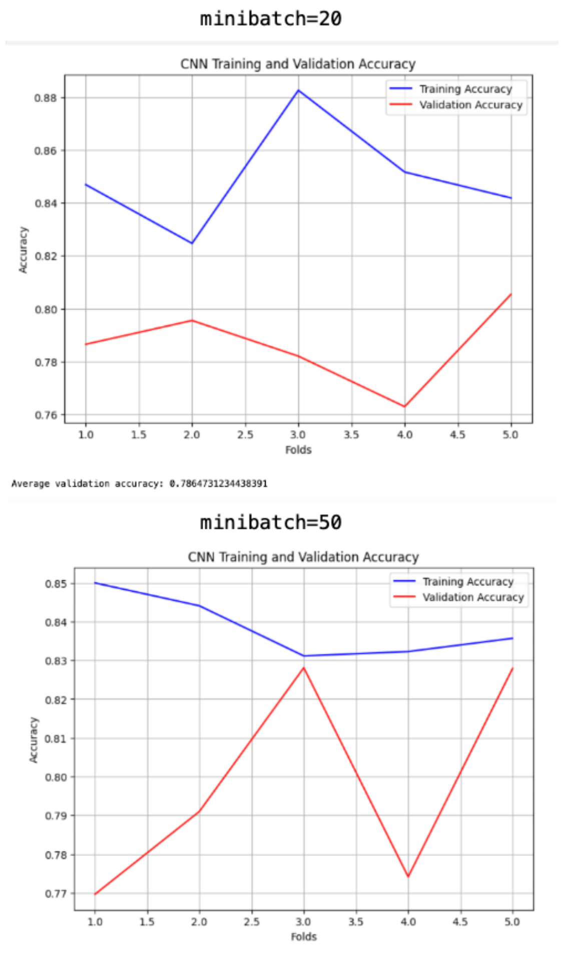
H. *Optimize 6: Minibatch Sizes*

When training the model, passing samples in "mini-batches" allows the data to be reorganized at each epoch to reduce model overfitting [4].

In the previous optimization model, the Minibatch Sizes of the model were set to 100. Based on the limitation of computational power, we tried to set the Minibatch Sizes to 20 and 50 for testing respectively.

As shown in Fig. 11, the average accuracy of the validation set is 78.65% when the mini-batch size is equal to 20. And when the mini-batch size is equal to 50, the average accuracy is 79.82%. Compared to the previously optimized model (minibatch=100, accuracy=78.26%), it can be seen that the accuracy is higher when the mini-batch is equal to 50.

There is also an interesting phenomenon. When minibatch continues to decrease from 50 to 20, the accuracy actually drops. The possible reason is that a small minibatch size is updated more frequently and can converge to the local optimal solution faster, but it may not be the global optimal solution.



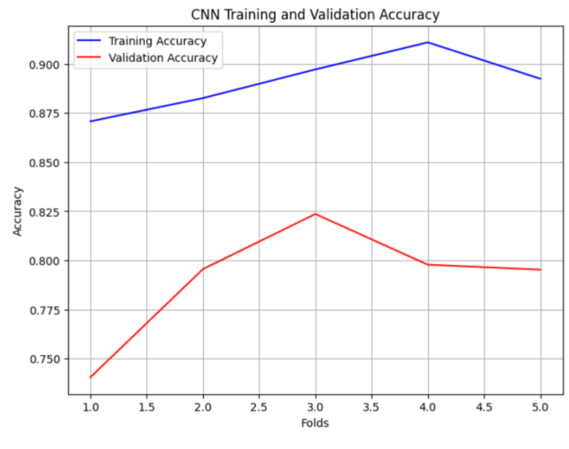
```
Average validation accuracy: 0.7864731234438391
```

```
Average validation accuracy: 0.7981635722500979
```

Fig. 11. Average accuracy for different minibatch sizes

I. *Optimize 7: Other Optimizations*

In order to achieve a better generalization model, we have made two additional attempts: one is to increase the number of fully connected layers, and the other is to perform dropout in the fully connected layer instead of the convolutional layer.

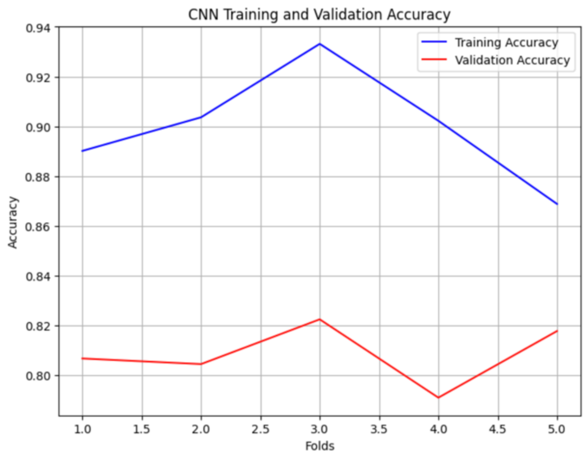


```
Average validation accuracy: 0.7905157922675395
```

Fig. 12. Average accuracy of 4 fully connected layers

In the previous optimization model and the Basic CNN model, there are three fully connected layers (1 input layer, 1 hidden layer with 128 neurons, and 1 output layer with 3 neurons). We tuned the fully connected layer to 4 layers (1 input layer, 2 hidden layers with 256 and 64 neurons respectively, and 1 output layer with 3 neurons) to try to improve the generalization of the model.

As can be seen from Fig. 12, compared with the model with three fully connected layers, the accuracy of the four fully connected layers has dropped (79.82%->79.05%). Therefore, the model still selects 3 fully connected layers.



```
Average validation accuracy: 0.8084983759052589
```

Fig. 13. Average accuracy of dropout on fully connected layer

For changing the dropout location, we no longer dropout on the convolutional layer and dropout on the fully connected hidden layer. As can be seen in Fig. 13, the accuracy is improved to 80.84%. Therefore, this optimization is kept in the final training model.

IV. SUMMARY AND DISCUSSION

A. *Structure and overall settings of baseline MLP and best CNN*

The Baseline MLP model consists of one input layer (number of neurons 300*300*3), one output layer (number of neurons 3) and two hidden layers (number of neurons 512 and 256). The activation function is ReLU, the loss function is cross-entropy loss, the optimizer is Adam (lr=0.001), and the mini-batch size is 100. In addition, 30 epochs are executed.

The best CNN model consists of five convolutional layers and three fully connected layers. The five convolutional layers have 32, 64, 128, 256 and 512 nodes, respectively, with a convolutional kernel size of 3*3 and padding of 1. Each convolutional layer is pooled with a pooling window of 4, 4, 9, 4, 4. And three fully connected layers consist of an input layer (number of neurons 512 * 6 * 6), an output layer (number of neurons 3), and a hidden layer (number of neurons 128). The activation functions for the convolutional and fully connected layers are both ReLU, the loss functions is cross-entropy loss, the optimizer is Adam (lr = 0.001), the mini-batch size is 50, and 15 epochs are executed.

B. *Compare the results of baseline MLP and CNN*

The highest validation set accuracy for Baseline MLP is 49.89%, while the 5-fold average validation set accuracy for the Best CNN model is 80.84%. For the training time, running on the Colab T4 GPU, the Baseline MLP has a runtime of 39 minutes, while the Best CNN has a runtime of 19 minutes and 4 seconds. The Best CNN Model can achieve higher validation set accuracy in shorter run times.

The loss and accuracy plot for Baseline MLP is shown in Fig. 5, and the loss and accuracy plot for Best CNN is shown in Fig. 14.

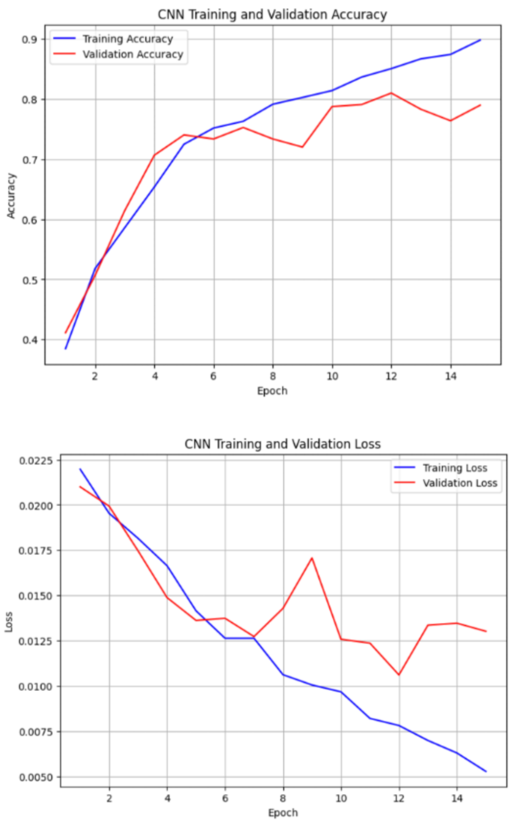


Fig. 14. Accuracy and loss of Best CNN on training and validation sets

Comparing Baseline MLP and Best CNN, first of all, the validation set accuracy trend is different. The accuracy of MLP on the validation set no longer rises significantly at 3 epochs, while the accuracy of CNN on the validation set has been trending up. It can be seen that the MLP model is quickly overfitting. The possible reason is that MLP lacks spatial utilization of images and cannot utilize the space features of images, so it cannot extract high-level features of images like CNN. As a result, the features extracted by MLP are not well used for image classification and lead to overfitting very quickly. In addition, the difference between the training set accuracy and the validation set accuracy of MLP is larger than that of CNN, which is also due to the overfitting problem.

The comparison of the Loss plots shows that within the same 15 epochs, the training set curve of the MLP model converges quickly, while the validation set curve is rises. This is also a problem of overfitting. More hyperparameters need to be tuned to minimize the overfitting problem.

V. Conclusions and Future Work

In general, for the problem of image classification, we can see that CNN has more obvious advantages than MLP. Especially for unseen images, the CNN model can achieve high accuracy.

The advantage of the method shown in this project is that the model is generalized better and better through step-by-step tuning, eventually reaching a relatively high average validation set accuracy to find better models and hyperparameters. At the same time, K folder cross-validation can also be used for better performance evaluation.

The limitation of my approach is that it takes a lot of time for parameter tuning, as cross-validation takes longer than performing 15 epochs alone. As a result, more hyperparametric possibilities are not considered. At the same time, there is a lack of samples for the training set data, because the classification of images is more complex, and more samples will lead to a better model.

For future work, training data can be increased to enable the model to learn more features. Operations such as flipping and rotating images can also be performed to increase the diversity of the training data.

In addition, more optimization methods can be tried to find more suitable hyperparameters. Additional techniques such as the use of integrated learning can also be used to further improve performance.

References

[1] Bhatt, Dulari, Chirag Patel, Hardik Talsania, Jigar Patel, Rasmika Vaghela, Sharnil Pandya, Kirit Modi, and Hemant Ghayvat. 2021. "CNN Variants for Computer Vision: History, Architecture, Application, Challenges and Future Scope" *Electronics* 10, no. 20: 2470. https://doi.org/10.3390/electronics10202470.

[2] Park, S., Kwak, N. (2017). Analysis on the Dropout Effect in Convolutional Neural Networks. In: Lai, SH., Lepetit, V., Nishino, K., Sato, Y. (eds) Computer Vision – ACCV 2016. ACCV 2016. Lecture Notes in Computer Science(), vol 10112. Springer, Cham. https://doi.org/10.1007/978-3-319-54184-6_12.

[3] Lecture slides and tutorials.

[4] https://pytorch.org/tutorials/beginner/basics/data_tutorial.html