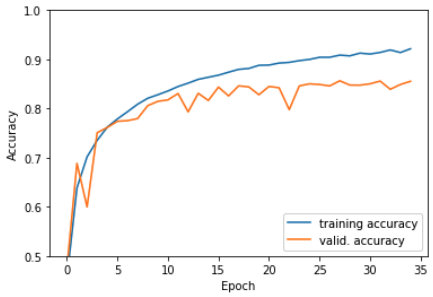
Csc578-hw7
Serena Yang
Kaggle Name: **SerenaMing Yang**
Ranking: **25**
Score: **1.63203**

A. Description of the best model

For this assignment, after I built the basic CNN model and compiled the model, I tried to change a few different hyperparameters tuning for the best prediction on the CIFAR-10 image dataset. After trying different hyperparameter tuning, the best model I produced has shown below. Based on the original model, I added convolutional layers to increase the complexity, applied batch normalization to get every layer of the network to do learning more independently, and added four dropout layers with different parameters to reduce the overfitting problem. Finally, I used the 'Adam' optimizer to compile the model and used epoch-35 to get more times experiments that find a better. The accuracy of the training dataset is 92%; the accuracy of the validation dataset is 86%; the loss is 1.63203 on Kaggle.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', padding = 'same', input_shape=(32, 32, 3)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(32, (3, 3), activation='relu', padding = 'same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2), strides=(2,2)))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), activation='relu', padding = 'same'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding = 'same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2), strides=(2,2)))
model.add(layers.Dropout(0.3))

model.add(layers.Conv2D(128, (3, 3), activation='relu', padding = 'same'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(128, (3, 3), activation='relu', padding = 'same'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.4))
model.add(layers.MaxPooling2D((2, 2), strides=(2,2)))

model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10, activation='softmax'))

model.compile(optimizer='Adam',
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), # For TF2
                  #loss='sparse_categorical_crossentropy', # For TF1
                  metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=35,
                    validation_data=(valid_images, valid_labels))
```

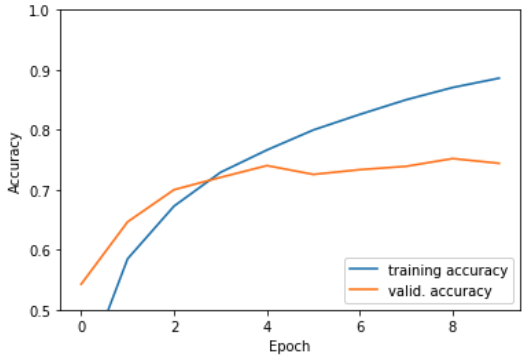
```
313/313 - 14s - loss: 0.4661 - accuracy: 0.8554 - 14s/epoch - 45ms/step
valid_accuracy=0.855400025844574, valid_loss=0.4661211371421814
```



B. Description of the journey of hyperparameter tuning

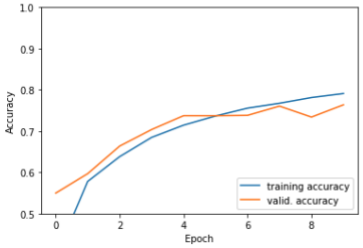
Following the list that the professor provided, I first tried Grid Search to find the best optimizer for the model within the list ['Adam', 'Adagrad', 'Adamax', 'Adadelta', 'SGD', 'RMSprop']. After comparing these optimizers' accuracy, I chose 'Adam' as the best optimizer for further models with 70% accuracy. Then, I tried a few different sizes of filters like 5*5 with 66% accuracy, 3*3 with 68% accuracy, and 7*7 with 63% accuracy. After comparing these results, I chose 3*3 size filters for further models. After I tried the above optimizer and the size of filters with no improvement on the model, I tried to add more convolution layers. I add more convolution layers because those layers can help the model increase the complexity of trying to fetch all the training data to improve the accuracy. The plot for the training accuracy and validation dataset's accuracy on this model has shown below. The model on the training dataset is 80%, but the accuracy is about 74% on the validation dataset. From this plot, it appears there is an overfitting problem; therefore, I will add dropout layers to reduce the overfitting problem.

```
313/313 - 4s - loss: 0.9194 - accuracy: 0.7441 - 4s/epoch - 13ms/step
valid_accuracy=0.7440999746322632, valid_loss=0.9194271564483643
```

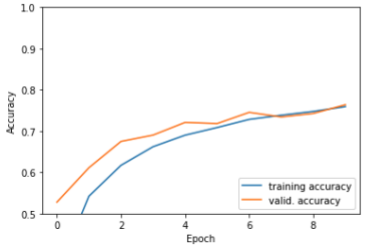


I first tried to add four dropout layers with 0.2 after every MaxPooling function and the first dense layer. The plot has shown below left. There is still a little overfitting for this dropout = 0.2 model from the plot. Therefore, I change the number for each dropout layer to 0.2, 0.3, 0.4, and 0.3. From the plot shown below right, we can see the overfitting problem has improved with 75% on the training dataset and 76% on the validation dataset.

```
313/313 - 4s - loss: 0.6882 - accuracy: 0.7636 - 4s/epoch - 12ms/step
valid_accuracy=0.7635999917984009, valid_loss=0.6882150769233704
```



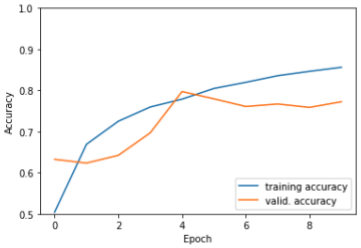

```
313/313 - 4s - loss: 0.7014 - accuracy: 0.7638 - 4s/epoch - 13ms/step
valid_accuracy=0.7638000249862671, valid_loss=0.7013581991195679
```



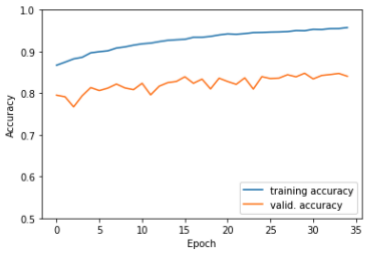
After fixing the overfitting problem, I tried to add the batch normalization function after each convolutional layer to get every layer of the network to do learning more independently to get a better prediction on this dataset. Also, I tried different epochs to get more times of experiments that find a better model to fetch the training data as

many as possible. The plot shown below left displays the training accuracy and validation accuracy with epochs = 10. The plot shown below right indicates there is still a considerable overfitting problem, even though the accuracy on the training dataset has improved to about 96%, and the accuracy on the validation dataset has improved to 84%. This model has caught my attention that the model performs differently on the validation dataset with different epoch numbers. For the epoch=10 model, there are two cross points between two lines, and after that, the overfitting problem gets clearer. As for the epoch-35 model, two lines are likely to parallel each other without any cross point. Even though the accuracy for validation loss with epoch = 35 is 0.57, I submitted to Kaggle with only 1.84 loss. After running to this stage, we still need to find the best dropout layer parameter to reduce the overfitting problem.

```
313/313 - 6s - loss: 0.8175 - accuracy: 0.7724 - 6s/epoch - 21ms/step
valid_accuracy=0.7724000215530396, valid_loss=0.8175137042999268
```



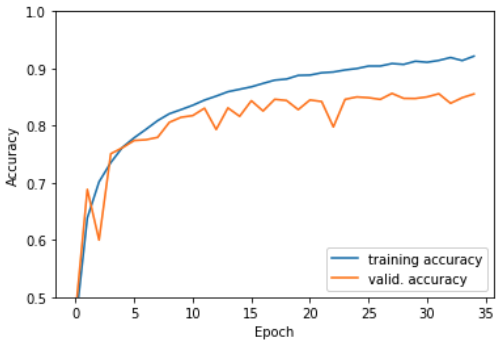

```
313/313 - 10s - loss: 0.5703 - accuracy: 0.8406 - 10s/epoch - 34ms/step
valid_accuracy=0.8406000137329102, valid_loss=0.5703471899032593
```



C. Final conclusion

Continue the model made in part b, and I tried a few different epochs numbers to find the best model. The final model with dropout layer: 0.2, 0.3, 0.4, 0.5, and epoch= 35, which improved about 2% on the validation dataset, and the loss on the validation dataset decreased by about 0.1. However, the plot is shown below still appears to be an overfitting problem. Therefore, I have also tried data augmentation. However, the running times took way too long, and my computer was crushed. If I had more time, I would try to remove layers to make the model simpler.

```
313/313 - 14s - loss: 0.4661 - accuracy: 0.8554 - 14s/epoch - 45ms/step
valid_accuracy=0.855400025844574, valid_loss=0.4661211371421814
```



D. Reaction and reflection

For this assignment, I think the difficulty level is medium to hard. The long-running time has caused a certain degree of difficulty to this assignment. Since building models is an extended exploration process, it is hard to find the best parameters for the best model.