# Report of question 6 - CNN for CIFAR10

In this part of the exercise, we aim to train on 50,000 images from 10 classes. To implement the neural network, we use convolutional and pooling layers to reduce the number of parameters and make the model more manageable. Additionally, we will examine the effects of the number of blocks and hidden layers, and use dropout and early stopping methods to achieve higher accuracy. We will also display these changes on a graph.

First, we need to preprocess the data. One of the tasks we perform is one-hot encoding. Instead of representing each class with a single number, a one-hot array of length equal to the number of classes is created for the labels. In this array, the presence of each class is indicated: all elements are zero except for the index corresponding to the class of the image, which is one. Alternatively, all elements can be numerical values between 0 and 1, showing the probability of each class for the image. The sum of all numbers in the array must equal 1.

### Reshape dataset

convert labels to one-hot encoding

```
from tensorflow.keras.utils import to_categorical
```

```
[5] cat_y_trainData = to_categorical(Y_trainData, num_classes=10)
    cat_y_testData = to_categorical(Y_testData, num_classes=10)
```

## Part One: Building a Simple CNN Model

In this part, we implement a model consisting of three blocks, each consisting of just two convolutional layers and one max pooling layer. Finally, all these sections are flattened and produce output through a single fully connected layer.

Homework2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
input = Input(shape=(32, 32, 3))

x = Conv2D(32, (3, 3), activation='relu', padding='same')(input)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = MaxPool2D((2, 2))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = MaxPool2D((2, 2))(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = MaxPool2D((2, 2))(x)

x = Flatten()(x)
x = Dropout(0.2)(x)

x = Dense(units=128)(x)
x = ReLU()(x)

x = Dense(units=10)(x)
predictions = Activation(activation='softmax')(x)


our_CNN_model = Model(input, predictions)
```

✓ 2s completed at 9:37 AM

Homework2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
our_CNN_model.fit(x=X_trainData, y=cat_y_trainData, epochs=35, batch_size=32,
                  validation_data=(X_testData, cat_y_testData))
```

```
1563/1563 [==============================] - 9s 6ms/step - loss: 0.3132 - accuracy: 0.8892 - val_loss: 0.8022 - val_accuracy: 0.7694
Epoch 8/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2925 - accuracy: 0.8973 - val_loss: 0.8084 - val_accuracy: 0.7692
Epoch 9/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2715 - accuracy: 0.9037 - val_loss: 0.8489 - val_accuracy: 0.7678
Epoch 10/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2578 - accuracy: 0.9087 - val_loss: 0.8871 - val_accuracy: 0.7608
Epoch 11/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2487 - accuracy: 0.9108 - val_loss: 0.8877 - val_accuracy: 0.7625
Epoch 12/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2357 - accuracy: 0.9175 - val_loss: 0.8619 - val_accuracy: 0.7718
Epoch 13/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2270 - accuracy: 0.9205 - val_loss: 0.8984 - val_accuracy: 0.7717
Epoch 14/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2138 - accuracy: 0.9248 - val_loss: 0.9761 - val_accuracy: 0.7628
Epoch 15/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2141 - accuracy: 0.9258 - val_loss: 0.9343 - val_accuracy: 0.7683
Epoch 16/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2057 - accuracy: 0.9284 - val_loss: 0.9818 - val_accuracy: 0.7682
Epoch 17/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2045 - accuracy: 0.9293 - val_loss: 0.9876 - val_accuracy: 0.7516
Epoch 18/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1984 - accuracy: 0.9322 - val_loss: 0.9630 - val_accuracy: 0.7607
Epoch 19/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1869 - accuracy: 0.9361 - val_loss: 0.9758 - val_accuracy: 0.7687
Epoch 20/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1946 - accuracy: 0.9348 - val_loss: 1.0325 - val_accuracy: 0.7730
Epoch 21/35
```

✓ 2s completed at 9:37 AM

Homework2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
Epoch 24/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1749 - accuracy: 0.9420 - val_loss: 1.0822 - val_accuracy: 0.7700
Epoch 25/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1734 - accuracy: 0.9416 - val_loss: 1.1625 - val_accuracy: 0.7600
Epoch 26/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1721 - accuracy: 0.9430 - val_loss: 1.0877 - val_accuracy: 0.7719
Epoch 27/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1679 - accuracy: 0.9446 - val_loss: 1.0801 - val_accuracy: 0.7678
Epoch 28/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1664 - accuracy: 0.9443 - val_loss: 1.0667 - val_accuracy: 0.7694
Epoch 29/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1682 - accuracy: 0.9439 - val_loss: 1.1349 - val_accuracy: 0.7716
Epoch 30/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1640 - accuracy: 0.9461 - val_loss: 1.0984 - val_accuracy: 0.7685
Epoch 31/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1655 - accuracy: 0.9458 - val_loss: 1.0841 - val_accuracy: 0.7652
Epoch 32/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1633 - accuracy: 0.9465 - val_loss: 1.1554 - val_accuracy: 0.7633
Epoch 33/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1571 - accuracy: 0.9487 - val_loss: 1.1518 - val_accuracy: 0.7773
Epoch 34/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1617 - accuracy: 0.9460 - val_loss: 1.2089 - val_accuracy: 0.7521
Epoch 35/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1545 - accuracy: 0.9502 - val_loss: 1.2077 - val_accuracy: 0.7678
<keras.callbacks.History at 0x7f478e337b50>
```

plot model loss history

✓ 2s completed at 9:37 AM

As we can see, the model initially trains well but eventually slows down and suffers from overfitting. This indicates that the architecture can be optimized further. To clarify the status of the simple model, we will plot its performance graphs.

Homework2.ipynb

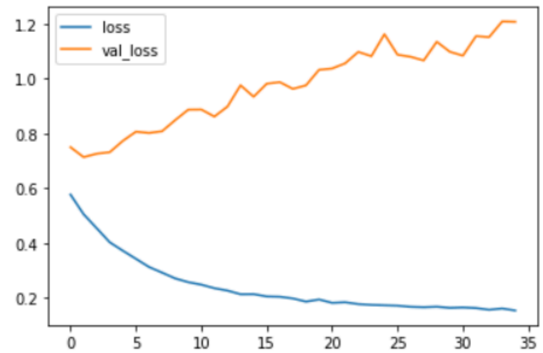
File Edit View Insert Runtime Tools Help All changes saved

\+ Code + Text

## plot model loss history

```
plot_history = pd.DataFrame(our_CNN_model.history.history)
plot_history[['loss', 'val_loss']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f478e331c50>
```




## min of loss history

```
np.min(our_CNN_model.history.history['loss'])
```

```
0.15448826551437378
```

2s completed at 9:37 AM

We observe that initially, the gap between training and validation data was decreasing, indicating that the model was performing well. However, due to overfitting, the gap between validation data and training data increased, which is undesirable. The same issue is evident with accuracy, where the performance on training data improves while the performance on validation data worsens.

Homework2.ipynb

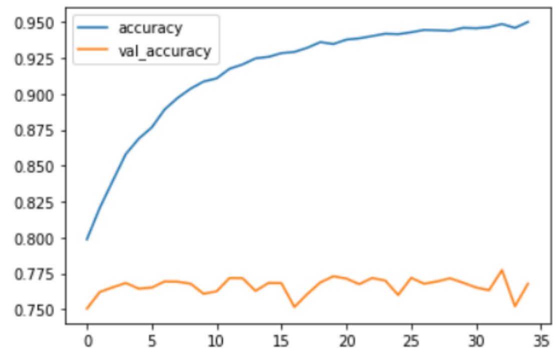
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

## plot model accuracy

```
plot_history = pd.DataFrame(our_CNN_model.history.history)
plot_history[['accuracy', 'val_accuracy']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f478e1cd8d0>
```




## max of validation accuracy

```
np.max(our_CNN_model.history.history['val_accuracy'])
```

```
0.7773000001907349
```

✓ 2s completed at 9:37 AM

Finally, we check to see how much the accuracy values of the data differ from the maximum accuracy achieved on the validation data.

## Evalute the model

```
our_CNN_model.evaluate(X_testData, cat_y_testData)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 1.2077 - accuracy: 0.7678
[1.2077003717422485, 0.767799973487854]
```

# Part Two: Implementing Hidden Layers with Greater Depth

Homework2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
input = Input(shape=(32, 32, 3))

x = Conv2D(filters=32, kernel_size=(3, 3), strides=1, padding='same')(input)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

x = Conv2D(filters=64, kernel_size=(3, 3), strides=1, padding='same')(x)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)


x = Conv2D(filters=128, kernel_size=(3, 3), strides=1, padding='same')(x)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

x = Conv2D(filters=256, kernel_size=(3, 3), strides=1, padding='same')(x)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

x = Conv2D(filters=512, kernel_size=(3, 3), strides=1, padding='same')(x)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

#globalaveragepooling
x = GlobalAveragePooling2D()(x)

x = Dense(units=128)(x)
x = ReLU()(x)
```

✓ 2s completed at 9:37 AM

```
x = Dense(units=32)(x)
x = ReLU()(x)
x = Dense(units=10)(x)
predictions = Activation(activation='softmax')(x)


our_CNN_model2 = Model(input, predictions)
```

In this layer, we increased the number of hidden layer blocks. Additionally, we also increased the number of fully connected layers to make the model deeper. We then retrained the model.

Homework2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
cur_CNN_model2.fit(x=x_traindata, y=cat_y_traindata, epochs=50, batch_size=32, validation_data=(x_testdata, cat_y_testdata))

Epoch 1/50
1563/1563 [==============================] - 12s 7ms/step - loss: 1.5562 - accuracy: 0.4194 - val_loss: 1.1465 - val_accuracy: 0.5885
Epoch 2/50
1563/1563 [==============================] - 10s 7ms/step - loss: 1.0516 - accuracy: 0.6266 - val_loss: 0.9858 - val_accuracy: 0.6554
Epoch 3/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.8297 - accuracy: 0.7104 - val_loss: 0.8603 - val_accuracy: 0.7035
Epoch 4/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.6944 - accuracy: 0.7584 - val_loss: 0.8427 - val_accuracy: 0.7255
Epoch 5/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.5932 - accuracy: 0.7913 - val_loss: 0.8069 - val_accuracy: 0.7285
Epoch 6/50
1563/1563 [==============================] - 10s 7ms/step - loss: 0.5053 - accuracy: 0.8221 - val_loss: 0.8228 - val_accuracy: 0.7413
Epoch 7/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.4203 - accuracy: 0.8542 - val_loss: 0.8807 - val_accuracy: 0.7194
Epoch 8/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3551 - accuracy: 0.8751 - val_loss: 0.8685 - val_accuracy: 0.7455
Epoch 9/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2976 - accuracy: 0.8946 - val_loss: 1.0104 - val_accuracy: 0.7312
Epoch 10/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2518 - accuracy: 0.9119 - val_loss: 1.0410 - val_accuracy: 0.7233
Epoch 11/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2243 - accuracy: 0.9213 - val_loss: 1.1103 - val_accuracy: 0.7180
Epoch 12/50
1563/1563 [==============================] - 10s 7ms/step - loss: 0.1882 - accuracy: 0.9352 - val_loss: 1.1567 - val_accuracy: 0.7431
Epoch 13/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1662 - accuracy: 0.9426 - val_loss: 1.2379 - val_accuracy: 0.7190
Epoch 14/50
1563/1563 [==============================] - 10s 7ms/step - loss: 0.1623 - accuracy: 0.9460 - val_loss: 1.2518 - val_accuracy: 0.7293
Epoch 15/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1421 - accuracy: 0.9523 - val_loss: 1.3186 - val_accuracy: 0.7336
```

✓ 2s completed at 9:37 AM

Homework2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

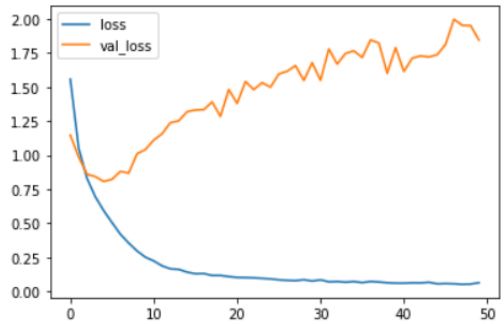
```
Epoch 37/50
1563/1563 [==============================] - 10s 7ms/step - loss: 0.0735 - accuracy: 0.9780 - val_loss: 1.8445 - val_accuracy: 0.7224
Epoch 38/50
1563/1563 [==============================] - 10s 7ms/step - loss: 0.0696 - accuracy: 0.9788 - val_loss: 1.8223 - val_accuracy: 0.7418
Epoch 39/50
1563/1563 [==============================] - 10s 7ms/step - loss: 0.0641 - accuracy: 0.9805 - val_loss: 1.6001 - val_accuracy: 0.7365
Epoch 40/50
1563/1563 [==============================] - 10s 7ms/step - loss: 0.0615 - accuracy: 0.9810 - val_loss: 1.7876 - val_accuracy: 0.7397
Epoch 41/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0614 - accuracy: 0.9816 - val_loss: 1.6135 - val_accuracy: 0.7431
Epoch 42/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0636 - accuracy: 0.9810 - val_loss: 1.7113 - val_accuracy: 0.7323
Epoch 43/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0630 - accuracy: 0.9814 - val_loss: 1.7279 - val_accuracy: 0.7411
Epoch 44/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0675 - accuracy: 0.9805 - val_loss: 1.7200 - val_accuracy: 0.7450
Epoch 45/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0565 - accuracy: 0.9827 - val_loss: 1.7364 - val_accuracy: 0.7225
Epoch 46/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0584 - accuracy: 0.9826 - val_loss: 1.8127 - val_accuracy: 0.7433
Epoch 47/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0566 - accuracy: 0.9835 - val_loss: 1.9963 - val_accuracy: 0.7372
Epoch 48/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0530 - accuracy: 0.9844 - val_loss: 1.9519 - val_accuracy: 0.7304
Epoch 49/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0546 - accuracy: 0.9842 - val_loss: 1.9505 - val_accuracy: 0.7356
Epoch 50/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0643 - accuracy: 0.9816 - val_loss: 1.8434 - val_accuracy: 0.7324
<keras.callbacks.History at 0x7f478e1b1510>
```

✓ 2s completed at 9:37 AM

As observed, although the model included more neurons and used global average pooling, there was no improvement in validation accuracy. In fact, the model even worsened.

```
[ ] plot_history = pd.DataFrame(our_CNN_model2.history.history)
    plot_history[['loss', 'val_loss']].plot()
```

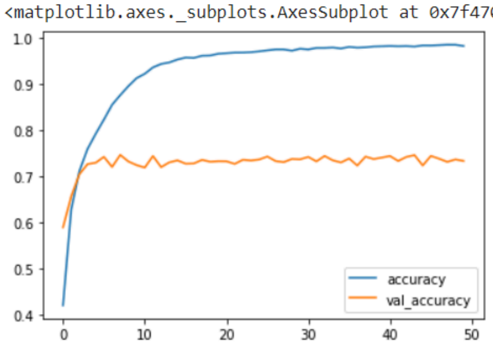
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47368e88d0>
```



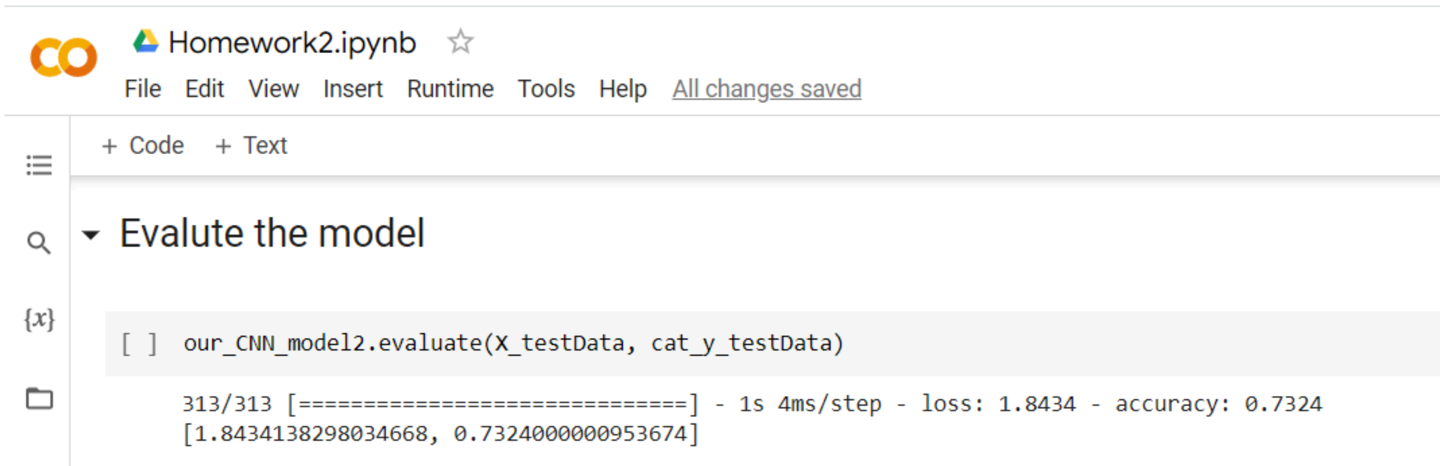

✓ 2s completed at 9:37 AM

```
[ ] plot_history[['accuracy', 'val_accuracy']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47021e8ed0>
```




In this problem, since our number of samples is not very large, increasing the number of layers can lead to issues. Each convolutional layer extracts a portion of features, and as the layers increase, the features become more detailed. This can result in overfitting during training or even a decrease in accuracy if the model becomes too complex. On the other hand, since the data is RGB, if the number of layers is too few, we may not extract meaningful features. Additionally, when flattening the model and adding more fully connected layers, the features obtained from the last convolutional layer might be insufficient, leading to inadequate updates during each epoch.



## Part Three: Using Alternative Architectures to Optimize the Model

In this part, we use early stopping to prevent overfitting. We do this by monitoring either the maximum accuracy or the minimum loss. For example, we set a threshold for accuracy and stop training if the accuracy does not improve for a certain number of epochs (e.g., ten epochs). Alternatively, we can set a threshold for the minimum loss and halt training if the loss does not decrease beyond this point.

Homework2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[13] our_CNN_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
     our_CNN_model.fit(x=X_trainData, y=cat_y_trainData, epochs=40, batch_size=32,
                       validation_data=(X_testData, cat_y_testData), callbacks=[early_stoping])
```

```
Epoch 1/40
1563/1563 [==============================] - 13s 7ms/step - loss: 1.4843 - accuracy: 0.4553 - val_loss: 1.0891 - val_accuracy: 0.6037
Epoch 2/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.9856 - accuracy: 0.6503 - val_loss: 0.9061 - val_accuracy: 0.6775
Epoch 3/40
1563/1563 [==============================] - 10s 6ms/step - loss: 0.7936 - accuracy: 0.7194 - val_loss: 0.7644 - val_accuracy: 0.7370
Epoch 4/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.6757 - accuracy: 0.7618 - val_loss: 0.7111 - val_accuracy: 0.7538
Epoch 5/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.5926 - accuracy: 0.7930 - val_loss: 0.6938 - val_accuracy: 0.7654
Epoch 6/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.5276 - accuracy: 0.8147 - val_loss: 0.6829 - val_accuracy: 0.7711
Epoch 7/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4705 - accuracy: 0.8338 - val_loss: 0.7276 - val_accuracy: 0.7609
Epoch 8/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4186 - accuracy: 0.8525 - val_loss: 0.6894 - val_accuracy: 0.7797
Epoch 9/40
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3735 - accuracy: 0.8683 - val_loss: 0.7058 - val_accuracy: 0.7787
Epoch 10/40
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3422 - accuracy: 0.8789 - val_loss: 0.7332 - val_accuracy: 0.7782
Epoch 11/40
1563/1563 [==============================] - 10s 7ms/step - loss: 0.3072 - accuracy: 0.8906 - val_loss: 0.7875 - val_accuracy: 0.7695
Epoch 12/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2908 - accuracy: 0.8964 - val_loss: 0.7776 - val_accuracy: 0.7784
Epoch 13/40
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2778 - accuracy: 0.9020 - val_loss: 0.8148 - val_accuracy: 0.7650
```

✓ 2s completed at 9:37 AM

In the plots, we observe that training stopped just before overfitting occurred, and the data did not suffer from overfitting.

Homework2.ipynb

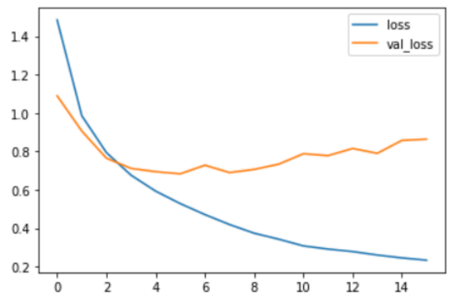
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

plot model loss history

```
[14] plot_history = pd.DataFrame(our_CNN_model.history.history)
     plot_history[['loss', 'val_loss']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faf420eb050>
```




min of loss history

```
[15] np.min(our_CNN_model.history.history['loss'])
```

```
0.23287376761436462
```

✓ 2s completed at 9:37 AM

Homework2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

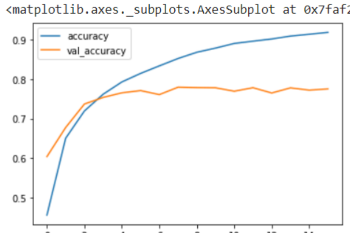
+ Code + Text

RAM Disk Editing

plot model accuracy

```
[16] plot_history = pd.DataFrame(our_CNN_model.history.history)
     plot_history[['accuracy', 'val_accuracy']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faf2b701090>
```



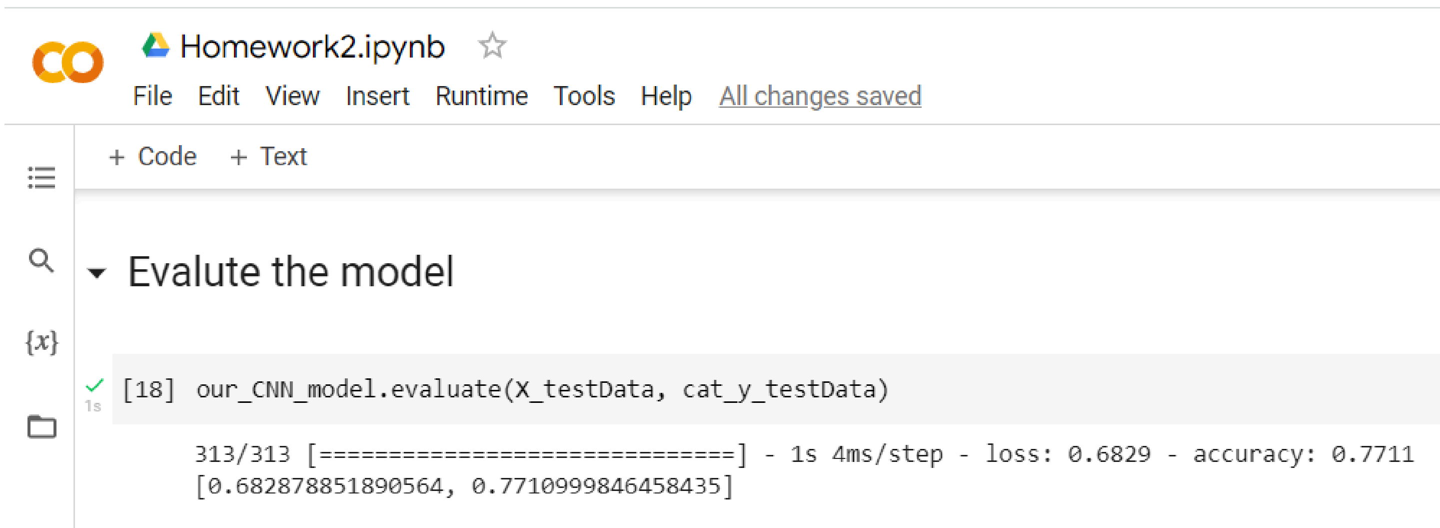

max of validation accuracy

```
[17] np.max(our_CNN_model.history.history['val_accuracy'])
```

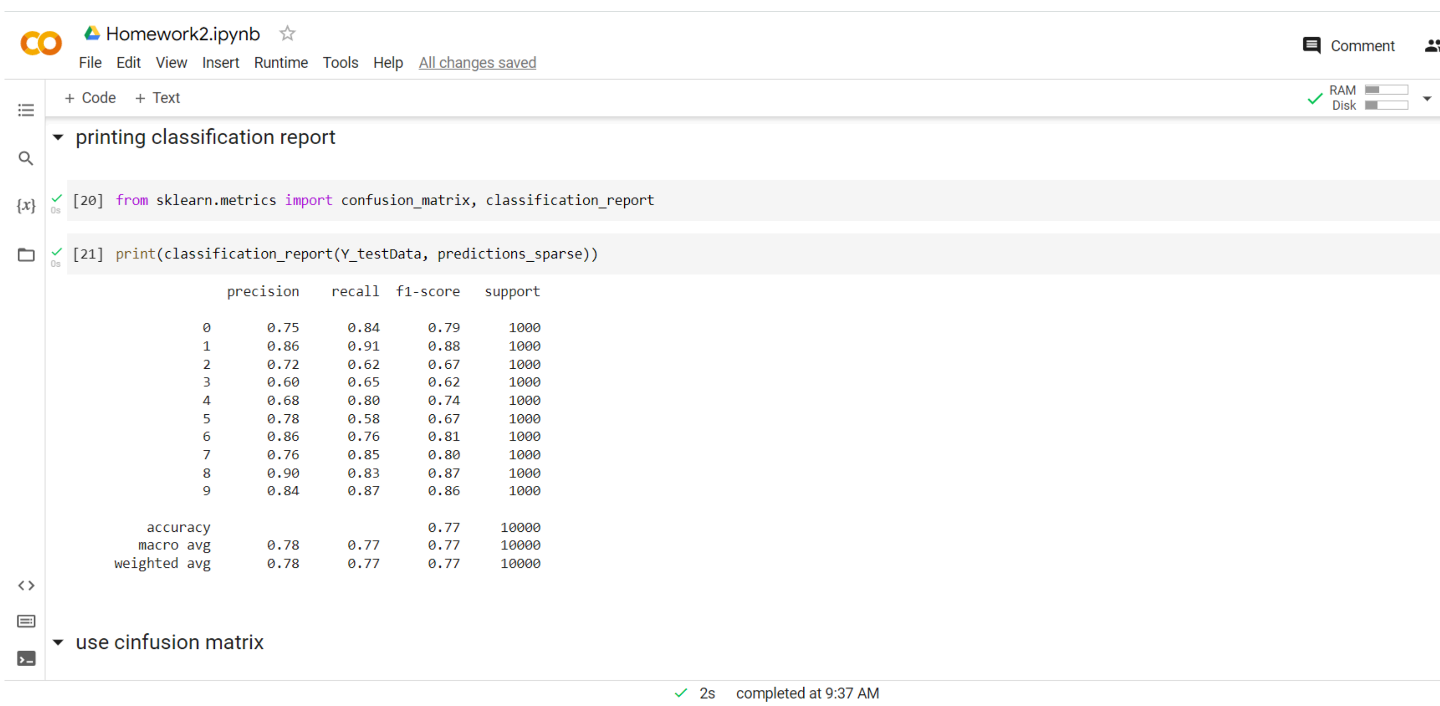
```
0.779699981212616
```

✓ 2s completed at 9:37 AM

And ultimately, we will achieve one of the best validation accuracies.




We can make a confusion matrix for it:



Homework2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

use cinfusion matrix

```
[22] confusion_matrix(Y_testData, predictions_sparse)

array([[843,  20,  27,  14,  19,   0,   0,  13,  38,  26],
       [ 10, 914,   1,   1,   2,   1,   5,   1,  15,  50],
       [ 97,   5, 620,  50,  93,  37,  48,  31,   7,  12],
       [ 25,   9,  49, 650,  77,  76,  34,  56,   7,  17],
       [ 18,   1,  44,  44, 799,  11,  20,  54,   3,   6],
       [ 14,   4,  46, 199,  54, 579,  10,  89,   3,   2],
       [ 11,   4,  38,  83,  74,  16, 756,   8,   5,   5],
       [ 15,   4,  22,  28,  46,  16,   1, 849,   2,  17],
       [ 68,  40,  11,  11,   4,   1,   1,   5, 835,  24],
       [ 27,  66,   4,   9,   5,   4,   1,   8,  10, 866]])
```

This matrix is such that, except for the diagonal where the values are high, other numbers have significant deviations, indicating that the model did not perform well in those areas. For example, in this model, it struggled to distinguish between the dog and cat classes, represented as class 4 and class 6 (with values 199 and 83, respectively), leading to higher confusion in these categories.

## Part Four: Using Batch Normalization and Dropout Techniques

In batch normalization, the process works by normalizing the data to a specific range after each convolutional layer and then passing it through the ReLU activation function. This helps to prevent the production of excessive outlier data and ensures that the data remains well-distributed, preventing the loss of important information.

Homework2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[24] input = Input(shape=(32, 32, 3))

     x = Conv2D(filters=32, kernel_size=(5, 5), strides=1, padding='same')(input)
     x = BatchNormalization()(x)
     x = ReLU()(x)
     x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

     x = Conv2D(filters=64, kernel_size=(3, 3), strides=1, padding='same')(x)
     x = BatchNormalization()(x)
     x = ReLU()(x)
     x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

     x = Conv2D(filters=128, kernel_size=(3, 3), strides=1, padding='same')(x)
     x = BatchNormalization()(x)
     x = ReLU()(x)
     x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

     x = Flatten()(x)

     x = Dense(units=128)(x)
     x = ReLU()(x)
     x = Dropout(0.3)(x)
     x = Dense(units=10)(x)
     predictions = Activation(activation='softmax')(x)

     our_CNN_model3 = Model(input, predictions)
```

✓ 2s completed at 9:37 AM



As we can see, early stopping prevents overfitting, and on the other hand, we achieve good accuracy with minimal discrepancy between training and validation accuracy. This indicates that our model has become quite effective with these modifications.

Homework2.ipynb

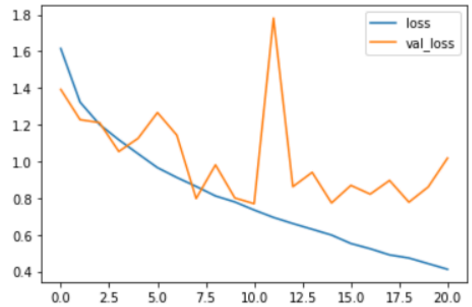
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

## plot model loss history

```
[29] plot_history3[['loss', 'val_loss']].plot()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7faea0a42d90>





Homework2.ipynb

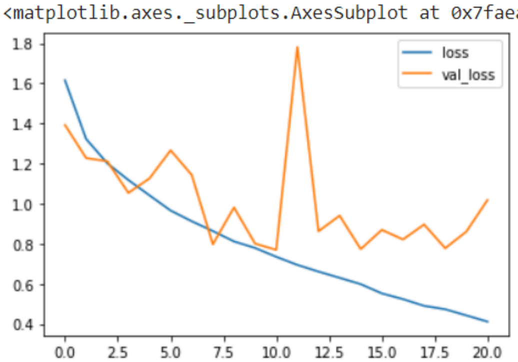
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

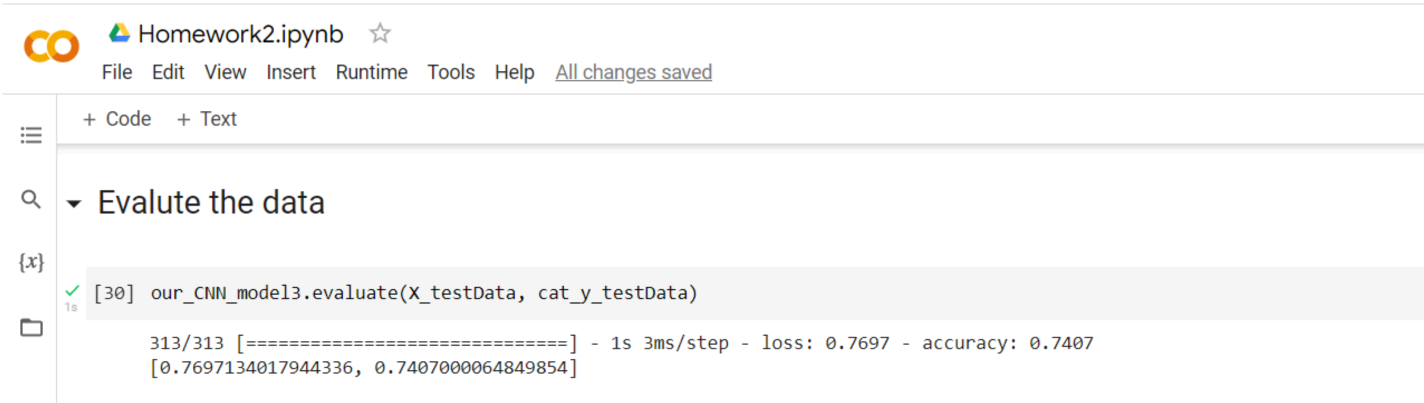
plot model loss history

[29] plot_history3[['loss', 'val_loss']].plot()

<matplotlib.axes._subplots.AxesSubplot at 0x7faea0a42d90>

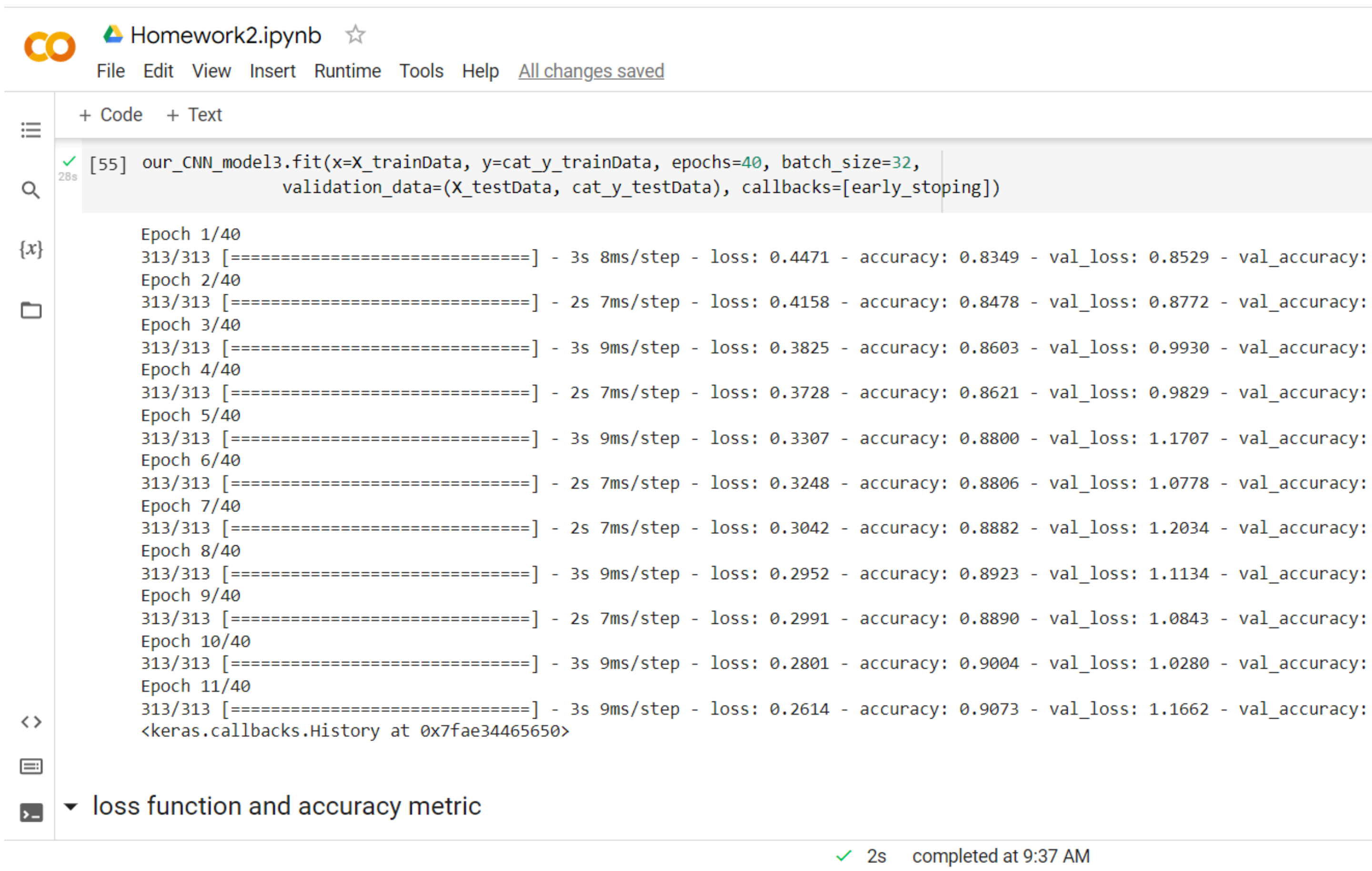



Finally, if we calculate the validation accuracy, we will obtain a good result.



## Part Five: Using a Subset of Data

In some cases, using a subset of the data can improve our accuracy, especially when the total amount of data is limited and training time is reduced. In this part of the code, we observe that even with a smaller portion of the data, we still achieve good accuracy. However, this subset cannot be too small because the initial data is grayscale, and it must be related to the type of data being used.



Homework2.ipynb

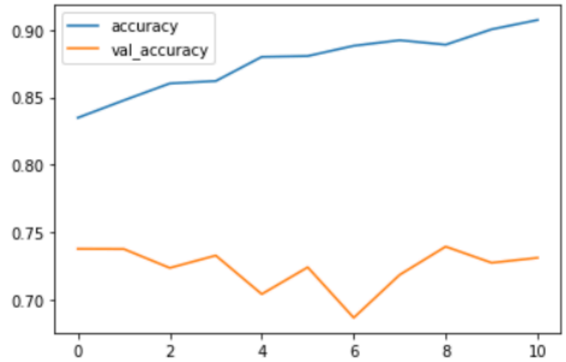
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

## loss function and accuracy metric

[56]
```
plot_history3 = pd.DataFrame(our_CNN_model3.history.history)
plot_history3[['accuracy', 'val_accuracy']].plot()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fae34c2abd0>




Homework2.ipynb

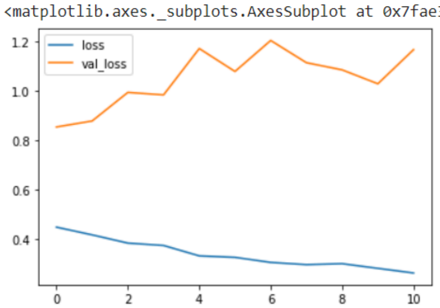
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

## plot model loss history

[57]
```
plot_history3[['loss', 'val_loss']].plot()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fae34bab390>



Homework2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

## Evalute the data

```
[58] our_CNN_model3.evaluate(X_testData, cat_y_testData)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.8529 - accuracy: 0.7378
[0.8528999090194702, 0.7378000020980835]
```

2s completed at 9:37 AM