Homework 2 - Implementing a CNN for CIFAR-10 dataset

Part1 - implemention of a basic convolutional neural network

Importing needed libraries

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

loading dataset

```
(X_trainData, Y_trainData), (X_testData, Y_testData) = tf.keras.datasets.cifar10.load_data
```

Reshape dataset

convert labels to one-hot encoding

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

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

Normalization

between 0 , 1 and float32

```
X_trainData = X_trainData.astype(np.float32) / 255.0
X_testData = X_testData.astype(np.float32) / 255.0
```

Creat our basic CNN model with stacking convelution and pooling layer

importing layers and models

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Dropout, MaxPool2D, Conv2D, Flatten, Glo
```

creating the model

```
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)
```

print summary

```
our_CNN_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0         
                                                                 
 conv2d (Conv2D)             (None, 32, 32, 32)        896       
                                                                 
 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248      
                                                                 
 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0         
 )                                                               
                                                                 
 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496     
```

```
 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten (Flatten)           (None, 2048)              0

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 128)               262272

 re_lu (ReLU)                (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

 activation (Activation)     (None, 10)                0

=================================================================
Total params: 550,570
Trainable params: 550,570
Non-trainable params: 0
_________________________________________________________________
```

Compile the model with optimizer and loss function

```
our_CNN_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
```

Train the model

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

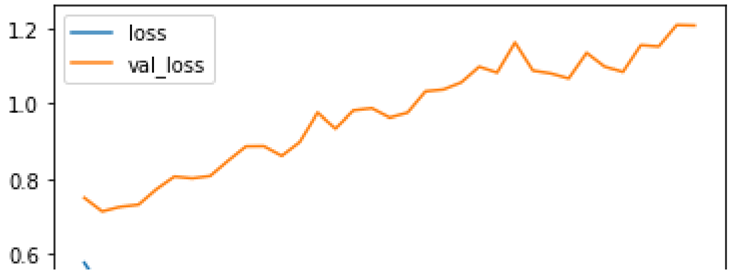
```
Epoch 8/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2925 - accurac
Epoch 9/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2715 - accurac
Epoch 10/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2578 - accuracy
Epoch 11/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2487 - accuracy
Epoch 12/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2357 - accuracy
Epoch 13/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2270 - accuracy
Epoch 14/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2138 - accurac
```

```
Epoch 15/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2141 - accuracy
Epoch 16/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2057 - accuracy
Epoch 17/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2045 - accuracy
Epoch 18/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1984 - accuracy
Epoch 19/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1869 - accurac
Epoch 20/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1946 - accurac
Epoch 21/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1823 - accuracy
Epoch 22/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1845 - accuracy
Epoch 23/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1778 - accurac
Epoch 24/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1749 - accuracy
Epoch 25/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1734 - accurac
Epoch 26/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1721 - accurac
Epoch 27/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1679 - accurac
Epoch 28/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1664 - accurac
Epoch 29/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1682 - accurac
Epoch 30/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1640 - accuracy
Epoch 31/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1655 - accuracy
Epoch 32/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1633 - accurac
Epoch 33/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1571 - accuracy
Epoch 34/35
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1617 - accuracy
Epoch 35/35
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1545 - accurac
<keras.callbacks.History at 0x7f478e337b50>
```

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

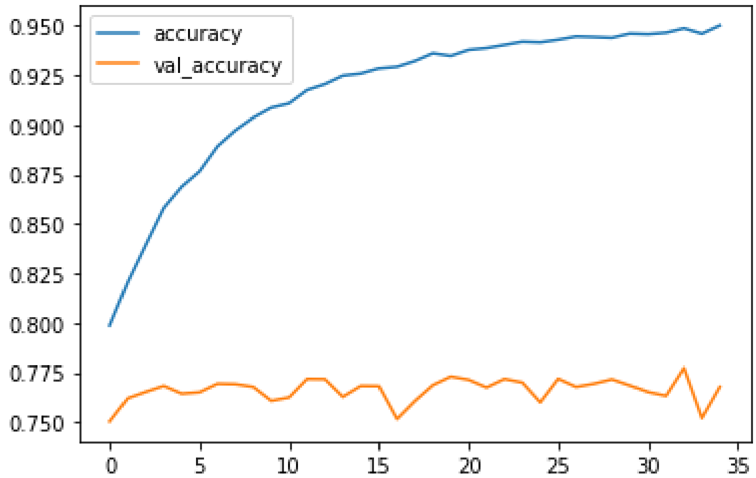
```
np.min(our_CNN_model.history.history['loss'])
```

```
0.15448826551437378
```

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
```

Evalute the model

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

```
313/313 [==============================] - 1s 4ms/step - loss: 1.2077 - accuracy: 0.7
```

```
[1.2077003717422485, 0.767799973487854]
```

Part2 - check the effect of layer's depth on the resalut

We use more hidden layers and GlobalArrangePooling

```
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)

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


our_CNN_model2 = Model(input, predictions)
```

Getting a summary

```
our_CNN_model2.summary()
```

```
 input_5 (InputLayer)        [(None, 32, 32, 3)]       0

 conv2d_8 (Conv2D)           (None, 32, 32, 32)        896

 re_lu_1 (ReLU)              (None, 32, 32, 32)        0

 max_pooling2d_3 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_9 (Conv2D)           (None, 16, 16, 64)        18496

 re_lu_2 (ReLU)              (None, 16, 16, 64)        0

 max_pooling2d_4 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_10 (Conv2D)          (None, 8, 8, 128)         73856

 re_lu_3 (ReLU)              (None, 8, 8, 128)         0

 max_pooling2d_5 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 conv2d_11 (Conv2D)          (None, 4, 4, 256)         295168

 re_lu_4 (ReLU)              (None, 4, 4, 256)         0

 max_pooling2d_6 (MaxPooling  (None, 2, 2, 256)        0
 2D)

 conv2d_12 (Conv2D)          (None, 2, 2, 512)         1180160

 re_lu_5 (ReLU)              (None, 2, 2, 512)         0

 max_pooling2d_7 (MaxPooling  (None, 1, 1, 512)        0
 2D)

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dense_2 (Dense)             (None, 128)               65664

 re_lu_6 (ReLU)              (None, 128)               0

 dense_3 (Dense)             (None, 32)                4128

 re_lu_7 (ReLU)              (None, 32)                0

 dense_4 (Dense)             (None, 10)                330

 activation_1 (Activation)   (None, 10)                0

=================================================================
Total params: 1,638,698
Trainable params: 1,638,698

Non-trainable params: 0
_________________________________________________________________
```

compile and training the model

```
our_CNN_model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
our_CNN_model2.fit(x=X_trainData, y=cat_y_trainData, epochs=50, batch_size=32, validation_
```

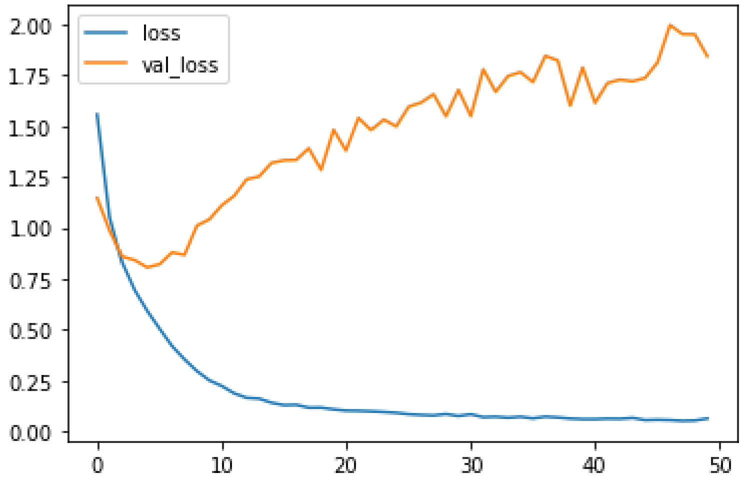
```
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0999 - accurac
Epoch 24/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0966 - accurac
Epoch 25/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0924 - accurac
Epoch 26/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0856 - accurac
Epoch 27/50

1563/1563 [==============================] - 10s 6ms/step - loss: 0.0820 - accurac
Epoch 28/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0797 - accurac
Epoch 29/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0866 - accurac
Epoch 30/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0768 - accurac
Epoch 31/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0855 - accurac
Epoch 32/50
1563/1563 [==============================] - 10s 7ms/step - loss: 0.0711 - accurac
Epoch 33/50
1563/1563 [==============================] - 10s 7ms/step - loss: 0.0731 - accurac
Epoch 34/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0684 - accurac
Epoch 35/50
1563/1563 [==============================] - 10s 7ms/step - loss: 0.0730 - accurac
Epoch 36/50
1563/1563 [==============================] - 11s 7ms/step - loss: 0.0652 - accurac
Epoch 37/50
1563/1563 [==============================] - 10s 7ms/step - loss: 0.0735 - accurac
Epoch 38/50
1563/1563 [==============================] - 10s 7ms/step - loss: 0.0696 - accurac
Epoch 39/50
1563/1563 [==============================] - 10s 7ms/step - loss: 0.0641 - accurac
Epoch 40/50
1563/1563 [==============================] - 10s 7ms/step - loss: 0.0615 - accurac
Epoch 41/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0614 - accurac
Epoch 42/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0636 - accurac
Epoch 43/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0630 - accurac
Epoch 44/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0675 - accurac
Epoch 45/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0565 - accurac
Epoch 46/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0584 - accurac
Epoch 47/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0566 - accurac
Epoch 48/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0530 - accurac
Epoch 49/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0546 - accurac
```

```
Epoch 50/50
1563/1563 [==============================] - 10s 6ms/step - loss: 0.0643 - accurac
<keras.callbacks.History at 0x7f478e1b1510>
```

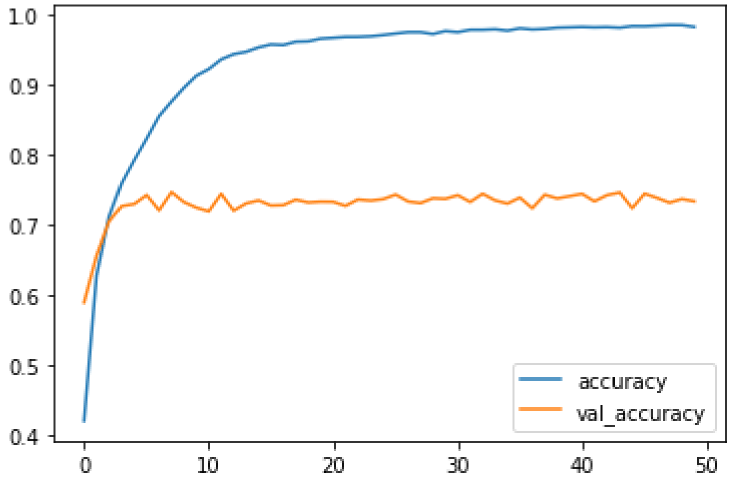
```
plot_history = pd.DataFrame(our_CNN_model2.history.history)
plot_history[['loss', 'val_loss']].plot()
```

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




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

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



Evalute the model

```
our_CNN_model2.evaluate(X_testData, cat_y_testData)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 1.8434 - accuracy: 0.7
[1.8434138298034668, 0.7324000000953674]
```

Part3 - check the early-stoping technic to raech an optimum model

use early-stoping in our model

to reach a fewer loss and a better accuracy

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
early_stoping = EarlyStopping(monitor='val_loss', mode='min', patience=10, restore_best_we
```

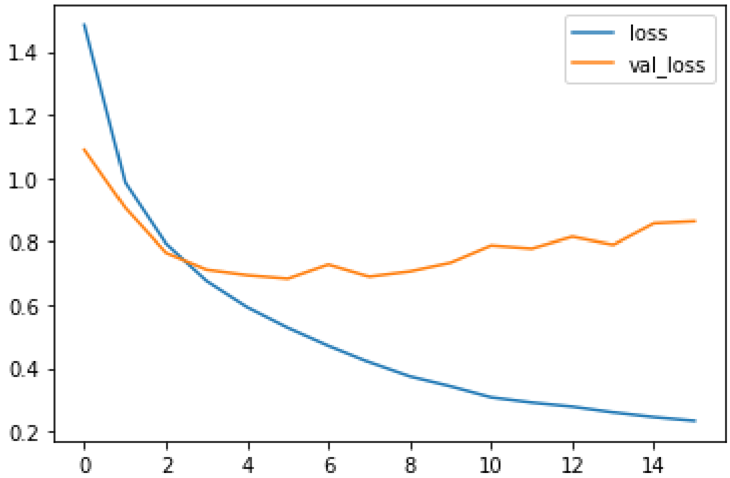
```
our_CNN_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac
our_CNN_model.fit(x=X_trainData, y=cat_y_trainData, epochs=40, batch_size=32,
             validation_data=(X_testData, cat_y_testData), callbacks=[early_stoping])
```

```
Epoch 1/40
1563/1563 [==============================] - 13s 7ms/step - loss: 1.4843 - accuracy:
Epoch 2/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.9856 - accuracy: 0
Epoch 3/40
1563/1563 [==============================] - 10s 6ms/step - loss: 0.7936 - accuracy:
Epoch 4/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.6757 - accuracy: 0
Epoch 5/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.5926 - accuracy: 0
Epoch 6/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.5276 - accuracy: 0
Epoch 7/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4705 - accuracy: 0
Epoch 8/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4186 - accuracy: 0
Epoch 9/40
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3735 - accuracy:
Epoch 10/40
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3422 - accuracy:
Epoch 11/40
1563/1563 [==============================] - 10s 7ms/step - loss: 0.3072 - accuracy:
Epoch 12/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2908 - accuracy: 0
Epoch 13/40
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2778 - accuracy:
Epoch 14/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2598 - accuracy: 0
Epoch 15/40
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2447 - accuracy: 0
Epoch 16/40
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2329 - accuracy:
<keras.callbacks.History at 0x7faf9001df50>
```

plot model loss history

```
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

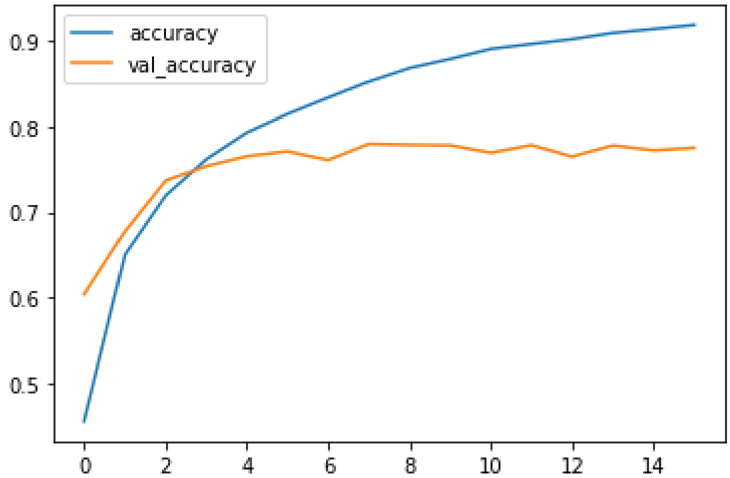
```
np.min(our_CNN_model.history.history['loss'])
```

```
0.23287376761436462
```

plot model accuracy

```
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

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

```
0.779699981212616
```

Evalute the model

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

```
313/313 [==============================] - 1s 4ms/step - loss: 0.6829 - accuracy: 0.7
[0.682878851890564, 0.7710999846458435]
```

confusion matrix and classification report

should be sparse

```
predictions = our_CNN_model.predict(X_testData)
predictions_sparse = np.argmax(predictions, axis=1)
predictions[0], predictions_sparse[0]
```

```
313/313 [==============================] - 1s 2ms/step
(array([1.4575123e-03, 4.9356726e-04, 4.6526126e-04, 8.7052953e-01,
        2.1654585e-05, 1.1634006e-01, 3.9997026e-03, 1.8319597e-03,
        4.5076455e-03, 3.5320688e-04], dtype=float32), 3)
```

printing classification report

```
from sklearn.metrics import confusion_matrix, classification_report
```

```
print(classification_report(Y_testData, predictions_sparse))
```

```
              precision    recall  f1-score   support

           0       0.75      0.84      0.79      1000
           1       0.86      0.91      0.88      1000
           2       0.72      0.62      0.67      1000
           3       0.60      0.65      0.62      1000
           4       0.68      0.80      0.74      1000
           5       0.78      0.58      0.67      1000
           6       0.86      0.76      0.81      1000
           7       0.76      0.85      0.80      1000
           8       0.90      0.83      0.87      1000
           9       0.84      0.87      0.86      1000
```

```
    accuracy                           0.77     10000
   macro avg       0.78      0.77      0.77     10000
weighted avg       0.78      0.77      0.77     10000
```

use cinfusion matrix

```
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]])
```

Part4 - using dropout layer and batch normalization to report its effect

Import dropout layer

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Dropout, MaxPool2D
from tensorflow.keras.layers import Conv2D, Flatten, GlobalAveragePooling2D, ReLU, Activat
```

Create the model using dropout layer

```
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)
```

summary of the model

```
our_CNN_model3.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 32, 32, 3)]       0         
                                                                 
 conv2d_6 (Conv2D)           (None, 32, 32, 32)        2432      
                                                                 
 batch_normalization (BatchN  (None, 32, 32, 32)       128       
 ormalization)                                                   
                                                                 
 re_lu_1 (ReLU)              (None, 32, 32, 32)        0         
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 16, 16, 32)       0         
 2D)                                                             
                                                                 
 conv2d_7 (Conv2D)           (None, 16, 16, 64)        18496     
                                                                 
 batch_normalization_1 (Batc  (None, 16, 16, 64)       256       
 hNormalization)                                                 
                                                                 
 re_lu_2 (ReLU)              (None, 16, 16, 64)        0         
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 8, 8, 64)         0         
 2D)                                                             
                                                                 
 conv2d_8 (Conv2D)           (None, 8, 8, 128)         73856     
                                                                 
 batch_normalization_2 (Batc  (None, 8, 8, 128)        512       
 hNormalization)                                                 
                                                                 
 re_lu_3 (ReLU)              (None, 8, 8, 128)         0         
                                                                 
 max_pooling2d_5 (MaxPooling  (None, 4, 4, 128)        0         
 2D)                                                             
                                                                 
 flatten_1 (Flatten)         (None, 2048)              0         
                                                                 
 dense_2 (Dense)             (None, 128)               262272    
```

```
 re_lu_4 (ReLU)              (None, 128)               0

 dropout_1 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 10)                1290

 activation_1 (Activation)   (None, 10)                0

=================================================================
Total params: 359,242
Trainable params: 358,794
Non-trainable params: 448
_________________________________________________________________
```

compile the model with optimizer

```
our_CNN_model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
```

Train the model using early stoping

```
our_CNN_model3.fit(x=X_trainData, y=cat_y_trainData, epochs=40, batch_size=32,
              validation_data=(X_testData, cat_y_testData), callbacks=[early_stoping])
```

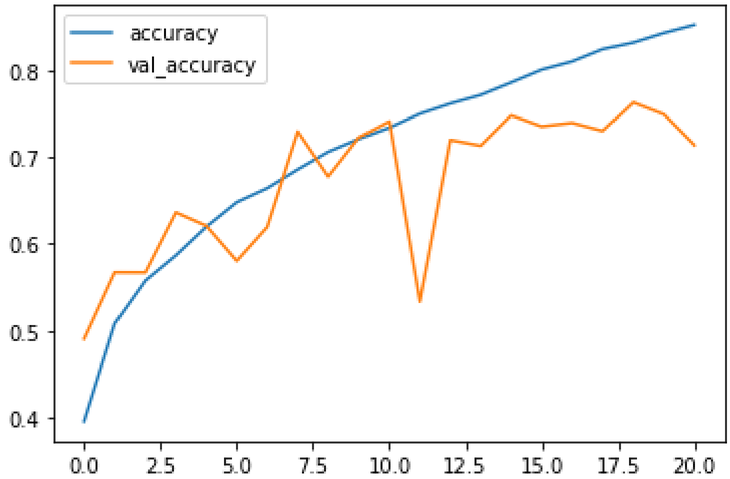
```
Epoch 1/40
1563/1563 [==============================] - 10s 6ms/step - loss: 1.6155 - accuracy:
Epoch 2/40
1563/1563 [==============================] - 8s 5ms/step - loss: 1.3231 - accuracy: 0
Epoch 3/40
1563/1563 [==============================] - 8s 5ms/step - loss: 1.2022 - accuracy: 0
Epoch 4/40
1563/1563 [==============================] - 8s 5ms/step - loss: 1.1183 - accuracy: 0
Epoch 5/40
1563/1563 [==============================] - 8s 5ms/step - loss: 1.0417 - accuracy: 0
Epoch 6/40
1563/1563 [==============================] - 8s 5ms/step - loss: 0.9661 - accuracy: 0
Epoch 7/40
1563/1563 [==============================] - 8s 5ms/step - loss: 0.9125 - accuracy: 0
Epoch 8/40
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8638 - accuracy: 0
Epoch 9/40
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8119 - accuracy: 0
Epoch 10/40
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7792 - accuracy: 0
Epoch 11/40
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7352 - accuracy: 0
Epoch 12/40
1563/1563 [==============================] - 8s 5ms/step - loss: 0.6948 - accuracy: 0
Epoch 13/40
1563/1563 [==============================] - 12s 8ms/step - loss: 0.6614 - accuracy:
Epoch 14/40
1563/1563 [==============================] - 8s 5ms/step - loss: 0.6303 - accuracy: 0
Epoch 15/40
```

```
1563/1563 [==============================] - 9s 6ms/step - loss: 0.5987 - accuracy: (
Epoch 16/40
1563/1563 [==============================] - 8s 5ms/step - loss: 0.5526 - accuracy: (
Epoch 17/40
1563/1563 [==============================] - 8s 5ms/step - loss: 0.5240 - accuracy: (
Epoch 18/40
1563/1563 [==============================] - 8s 5ms/step - loss: 0.4905 - accuracy: (
Epoch 19/40
1563/1563 [==============================] - 8s 5ms/step - loss: 0.4735 - accuracy: (
Epoch 20/40
1563/1563 [==============================] - 8s 5ms/step - loss: 0.4428 - accuracy: (
Epoch 21/40
1563/1563 [==============================] - 8s 5ms/step - loss: 0.4120 - accuracy: (
<keras.callbacks.History at 0x7fae35fe2890>
```

loss function and accuracy metric

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fae33e51f10>
```



plot model loss history

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

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

Evalute the data

```
our_CNN_model3.evaluate(X_testData, cat_y_testData)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.7697 - accuracy: 0.7
[0.7697134017944336, 0.7407000064849854]
```

Part5 - using some portion of data

```
X_trainData = X_trainData[:40000]
X_trainData.shape
```

```
(10000, 32, 32, 3)
```

```
Y_trainData = Y_trainData[:40000]
Y_trainData.shape
```

```
(10000, 1)
```

```
cat_y_trainData = cat_y_trainData[0:40000]
cat_y_trainData.shape
```

```
(10000, 10)
```

```
our_CNN_model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
```

```
our_CNN_model3.fit(x=X_trainData, y=cat_y_trainData, epochs=40, batch_size=32,
              validation_data=(X_testData, cat_y_testData), callbacks=[early_stoping])
```

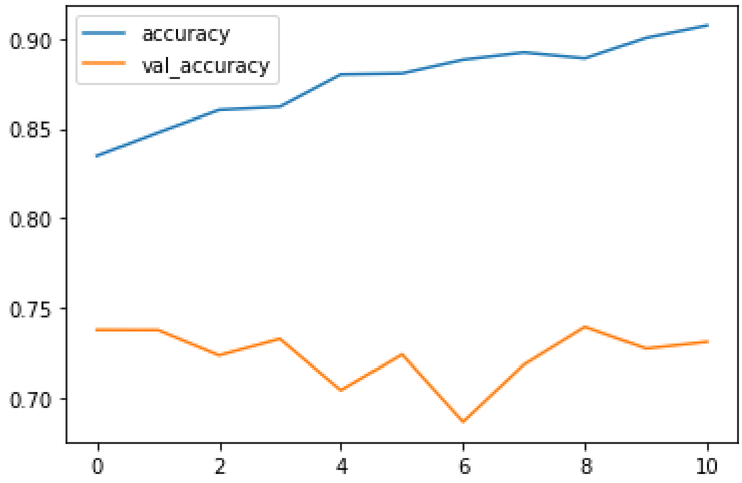
```
Epoch 1/40
313/313 [==============================] - 3s 8ms/step - loss: 0.4471 - accuracy: 0.8
Epoch 2/40
313/313 [==============================] - 2s 7ms/step - loss: 0.4158 - accuracy: 0.8
Epoch 3/40
313/313 [==============================] - 3s 9ms/step - loss: 0.3825 - accuracy: 0.8
Epoch 4/40
313/313 [==============================] - 2s 7ms/step - loss: 0.3728 - accuracy: 0.8
Epoch 5/40
313/313 [==============================] - 3s 9ms/step - loss: 0.3307 - accuracy: 0.8
Epoch 6/40
313/313 [==============================] - 2s 7ms/step - loss: 0.3248 - accuracy: 0.8
Epoch 7/40
313/313 [==============================] - 2s 7ms/step - loss: 0.3042 - accuracy: 0.8
Epoch 8/40
```

```
313/313 [==============================] - 3s 9ms/step - loss: 0.2952 - accuracy: 0.8
Epoch 9/40
313/313 [==============================] - 2s 7ms/step - loss: 0.2991 - accuracy: 0.8
Epoch 10/40
313/313 [==============================] - 3s 9ms/step - loss: 0.2801 - accuracy: 0.9
Epoch 11/40
313/313 [==============================] - 3s 9ms/step - loss: 0.2614 - accuracy: 0.9
<keras.callbacks.History at 0x7fae34465650>
```

loss function and accuracy metric

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

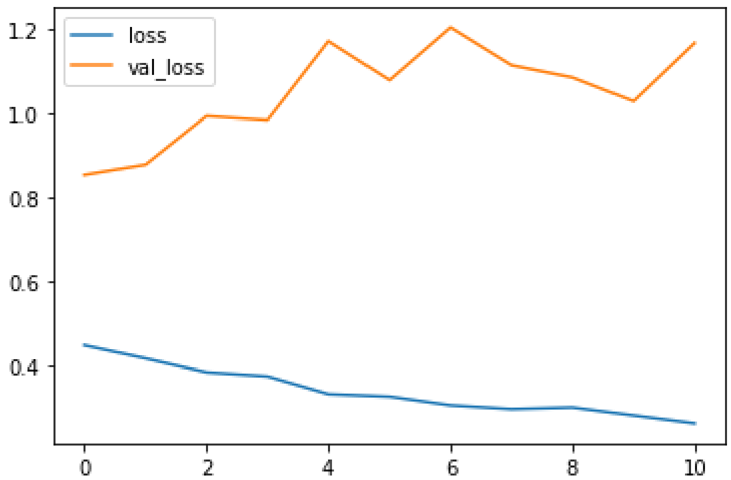
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fae34c2abd0>
```



plot model loss history

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

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



Evalute the data

```
our_CNN_model3.evaluate(X_testData, cat_y_testData)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.8529 - accuracy: 0.7
[0.8528999090194702, 0.7378000020980835]
```