Madhulika Dayal
700743206

Assignment – 7

Image classification with CNN :

Github Link : https://github.com/Madhulika014/Assignment-7

```
In [8]: import keras
        from keras.models import Sequential
        from keras.preprocessing import image
        from keras.layers import Activation,Dense,Dropout,Conv2D,Flatten,MaxPooling2D,BatchNormalization
        from keras.datasets import cifar10
        from keras import optimizers
        from matplotlib import pyplot as plt
```

```
In [10]: #generate cifar10 data
         (x_train,y_train),(x_test,y_test) = cifar10.load_data()
```

```
In [9]: #config parameters
        num_classes = 10
        input_shape = x_train.shape[1:4]
        optimizer = optimizers.Adam(lr=0.001)
```

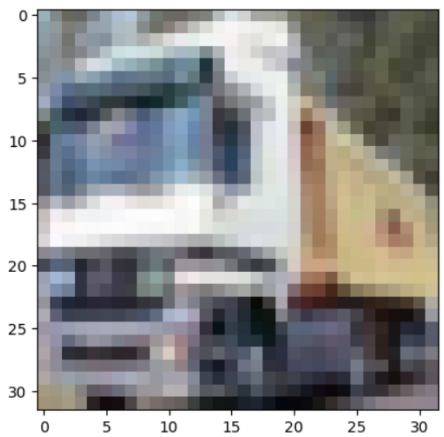
```
In [11]: #convert label to one-hot
         one_hot_y_train = keras.utils.to_categorical(y_train,num_classes=num_classes)
         one_hot_y_test = keras.utils.to_categorical(y_test,num_classes=num_classes)
```

Importing packages and generate cifar10 data, configuring parameters with shape [1:4], number class 10 and optimizers with argument lr = 0.001.

Now labels are converting to one-hot variables.

```
In [12]: # check data
         plt.imshow(x_train[1])
         print(x_train[1].shape)
```

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



Now checking the data and print it with size 32*32.

In [13]:

```
# build model(similar to VGG16, only change the input and output shape)
model = Sequential()
model.add(Conv2D(64,(3,3),activation='relu',input_shape=input_shape,padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))
```

Building the model similar to VGG16 but with only change with input and output shape.

In [14]:

```
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

In [15]:

```
model.summary()
```

```
 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 4096)              2101248

 dense_1 (Dense)             (None, 2048)              8390656

 dense_2 (Dense)             (None, 1024)              2098176

 dropout_5 (Dropout)         (None, 1024)              0

 dense_3 (Dense)             (None, 10)                10250

 activation (Activation)     (None, 10)                0

=================================================================
Total params: 27,331,914
Trainable params: 27,323,466
Non-trainable params: 8,448
_________________________________________________________________
```

Compiling the model with arguments optimizer, loss and metrics.

After that printing the summary using summary function.

```
In [ ]: history = model.fit(x=x_train, y=one_hot_y_train, batch_size=128, epochs=30, validation_split=0.1)
```

```
Epoch 1/30
352/352 [==============================] - 29s 81ms/step - loss: 1.7083 - accuracy: 0.3346 - val_loss: 2.6400 - val_accuracy:
0.3686
Epoch 2/30
352/352 [==============================] - 27s 76ms/step - loss: 1.3159 - accuracy: 0.5254 - val_loss: 1.6935 - val_accuracy:
0.5590
Epoch 3/30
352/352 [==============================] - 27s 77ms/step - loss: 1.0110 - accuracy: 0.6538 - val_loss: 1.0628 - val_accuracy:
0.6556
Epoch 4/30
352/352 [==============================] - 27s 78ms/step - loss: 0.8493 - accuracy: 0.7144 - val_loss: 0.9995 - val_accuracy:
0.6856
Epoch 5/30
352/352 [==============================] - 28s 79ms/step - loss: 0.7343 - accuracy: 0.7566 - val_loss: 0.8316 - val_accuracy:
0.7326
Epoch 6/30
352/352 [==============================] - 28s 79ms/step - loss: 0.6515 - accuracy: 0.7907 - val_loss: 0.8145 - val_accuracy:
0.7648
Epoch 7/30
352/352 [==============================] - 28s 80ms/step - loss: 0.5708 - accuracy: 0.8168 - val_loss: 0.7658 - val_accuracy:
0.7716
Epoch 8/30
352/352 [==============================] - 28s 79ms/step - loss: 0.5135 - accuracy: 0.8339 - val_loss: 0.6754 - val_accuracy:
0.7944
Epoch 9/30
352/352 [==============================] - 28s 79ms/step - loss: 0.4665 - accuracy: 0.8506 - val_loss: 0.7615 - val_accuracy:
0.7686
Epoch 10/30
352/352 [==============================] - 28s 80ms/step - loss: 0.4213 - accuracy: 0.8662 - val_loss: 0.7079 - val_accuracy:
0.8050
```

We are using fit function to model training on the data set for 30 epochs and batch size of 128.

```
In [ ]: # evaluate
print(model.metrics_names)
model.evaluate(x=x_test,y=one_hot_y_test,batch_size=512)
```

```
['loss', 'accuracy']
20/20 [==============================] - 5s 132ms/step - loss: 0.6627 - accuracy: 0.8592
```

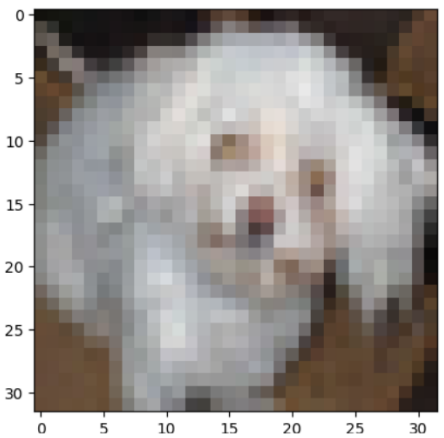
```
Out[21]: [0.6626988053321838, 0.8592000007629395]
```

Now, we are printing metrics_name and evaluating with argument batch_size of 512. Now the loss is 0.66269 and accuracy is 0.85920.

```
In [ ]: model.save("keras-VGG16-cifar10.h5")
plt.imshow(x_test[1000])

result = model.predict(x_test[1000:1001]).tolist()
predict = 0
expect = y_test[1000][0]
for i,_ in enumerate(result[0]):
   if result[0][i] > result[0][predict]:
    predict = i
print("predict class:",predict)
print("expected class:",expect)
```

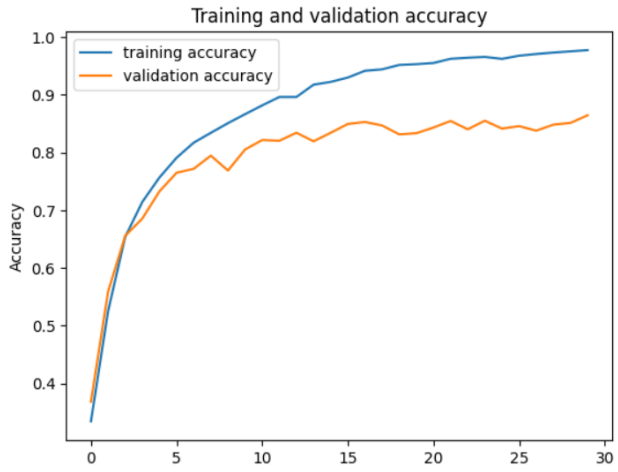
```
1/1 [==============================] - 1s 740ms/step
predict class: 5
expected class: 5
```



Now we are saving the model using save function. And predicting the model into datatype list. And, printing predict class and expected class.

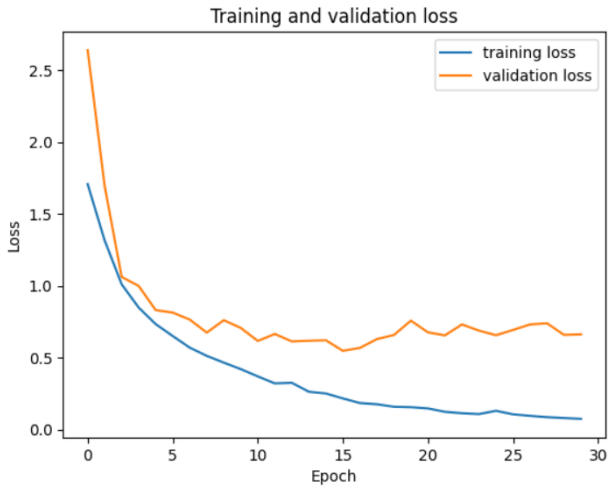
In [16]:
```
# save model
model.save("keras-VGG16-cifar10.h5")
```

In [ ]:
```
#plot the training and validation accuracy
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

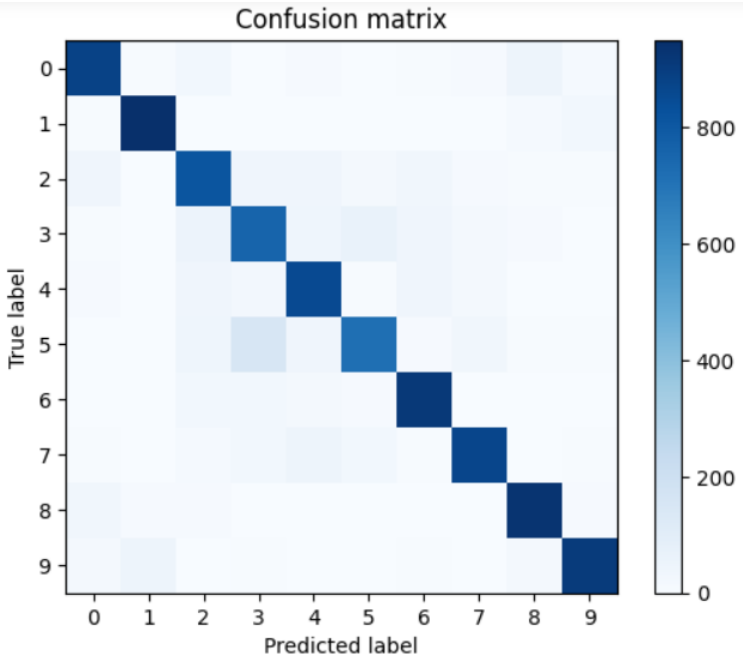


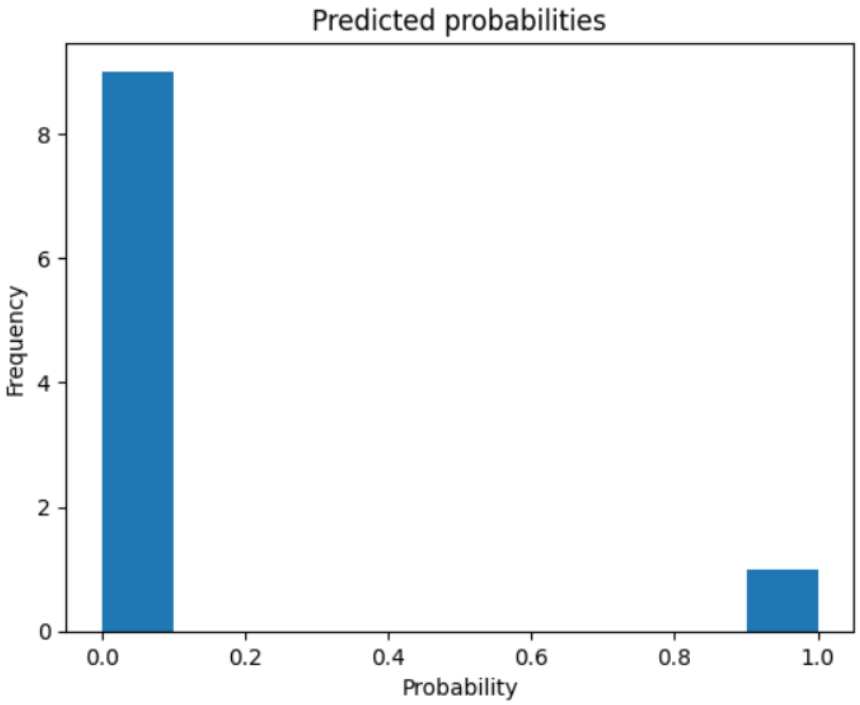
Now saving the model again and printing graph of Training and Validation accuracy.

In [ ]:
```
#plot the training and validation loss
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Plotting the training and validation loss.





Now printing the confusion matrix and Predicted Probabilities.