CNN Model Activity (Convolutional Neural Networks) – Unit 9

Read the Wall (2019) article and record your thoughts on the ethical and social implications of this CNN technology.

Run this CNN model - ***Convolutional Neural Networks (CNN) - Object Recognition.ipynb*** - and **review different sections of the algorithm**. Change the input image for prediction by changing the value of this variable - *plt.imshow(x_test[16]* - from 16 to value of your choice (1-15) and see whether the model predicts correctly.

Importing the required tools

```
from numpy.random import seed
seed(888)

#from tensorflow import set_random_seed
#set_random_seed(4112)
import tensorflow
tensorflow.random.set_seed(112)
```

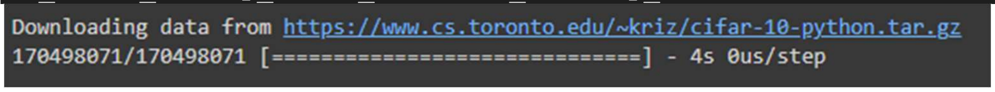
```
import os
import numpy as np
import itertools

import tensorflow as tf
import keras
from keras.datasets import cifar10 # importing the dataset

from keras.models import Sequential        #to define model/ layers
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten

from sklearn.metrics import confusion_matrix

# To Explore the images
from IPython.display import display
from keras.preprocessing.image import array_to_img

from tensorflow.keras.utils import to_categorical

import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
```

We are using Tensorflow to power Keras

CIFAR-10 is an established computer-vision dataset used for object recognition. It is a subset of the 80 million tiny images dataset and consists of 60,000 32x32 color images containing one of 10 object classes, with 6000 images per class. It was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The dataset is popularly used to train image classification models – **The same dataset that is being used for the assignment presentation**

Downloading the dataset

```
# Getting the dataset as a Tuple

(x_train_all, y_train_all), (x_test, y_test) = cifar10.load_data()
```



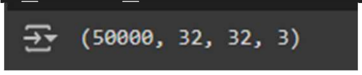
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
```

Constants

```
LABEL_NAMES = ['airplane', 'automobile','bird','cat', 'deer', 'dog',
'frog', 'horse', 'ship', 'truck']
```

Gathering an idea of the shape of the dataset

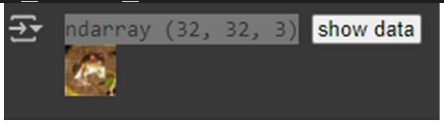
```
x_train_all.shape
```



```
(50000, 32, 32, 3)
```

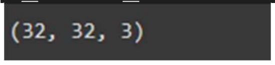
The first image of the dataset will be [0] and the dimensions are 32x32 and 3 is the number of colour channels

```
x_train_all[0]
```



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

```
x_train_all[0].shape
```



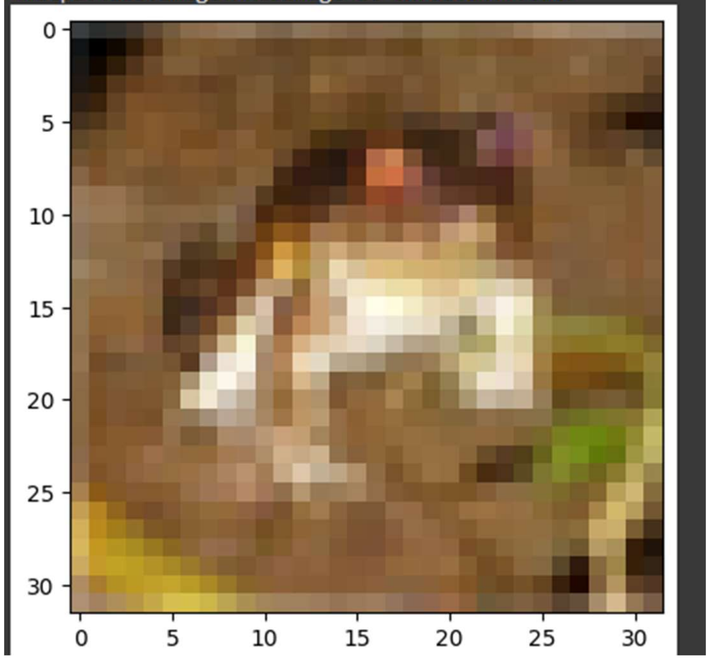
```
(32, 32, 3)
```

```
# To use the ipython display to view an image

pic = array_to_img(x_train_all[0])
display(pic)
```

Using Matplotlib to view the image

```
#Using Matplotlib to view the image
plt.imshow(x_train_all[0])
```

```
<matplotlib.image.AxesImage at 0x7ca0a4fff0a0>
```



```
# To check the label
y_train_all.shape
```

```
(50000, 1)
```

```
# Note that in the image above the index 1 corresponds to "Automobile"
# we have a 2 dimension numpy array; that is why we also include " [0] "

y_train_all[0][0]
```

```
6
```

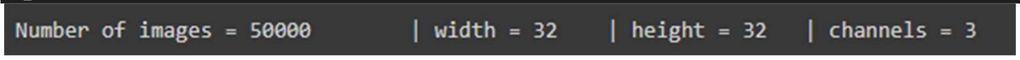
```
# Using the label names to get the actual names of classes

LABEL_NAMES[y_train_all[0][0]]
```

```
'frog'
```

```
x_train_all.shape
```

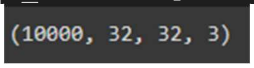
```
(50000, 32, 32, 3)
```

```
number_of_images, x, y, c = x_train_all.shape
print(f'Number of images = {number_of_images} \t| width = {x} \t| height =
{y} \t| channels = {c}')
```



```
Number of images = 50000        | width = 32     | height = 32     | channels = 3
```

```
x_test.shape
```



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

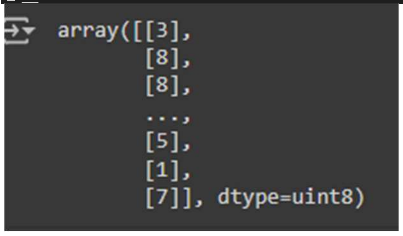
Preprocessing data

We need to do this so that it’s easier to find within the neural network

```
x_train_all =x_train_all / 255.0
```

```
x_test =  x_test / 255.0
```

```
y_test
```



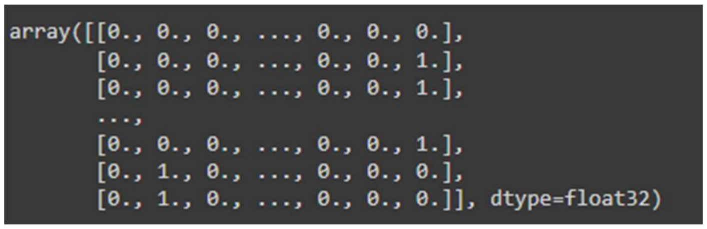
```
array([[3],
       [8],
       [8],
       ...,
       [5],
       [1],
       [7]], dtype=uint8)
```

Creating categorial encoding for the y data

```
# 10 >>> simply means we have 10 classes like we already know (creating
the encoding for 10 classes)
y_cat_train_all = to_categorical(y_train_all,10)
```

```
# 10 >>> simply means we have 10 classes like we already know (creating
the encoding for 10 classes)
y_cat_test = to_categorical(y_test,10)
```

```
y_cat_train_all
```


```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]], dtype=float32)
```

Creating the validation dataset

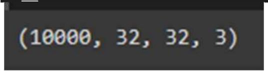
For small data * 60% for Training * 20% Validation * 20% Testing is normally used

Only the final selected model gets to see the testing data. This helps us to ensure that we have close to real data in real-world when the model is deployed. Only our best model gets to see our testing dataset. Because it will give us a realistic impression of how our model will do in the real world

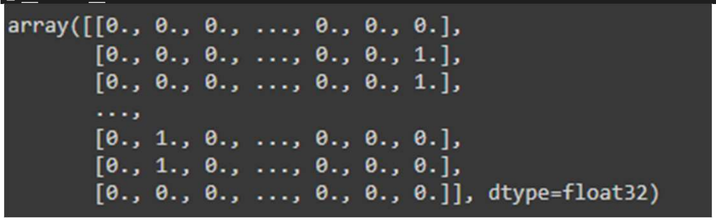
However, if the dataset is enormous.: * 1% for is used for validation * 1% for is used for testing

```
VALIDATION_SIZE = 10000
```

```
# VALIDATION_SIZE = 10,000 as defined above

x_val = x_train_all[:VALIDATION_SIZE]
y_val_cat = y_cat_train_all[:VALIDATION_SIZE]
x_val.shape
```


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

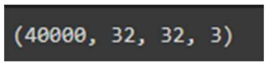
```
y_val_cat
```


```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       ...,
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

- We Create two NumPy arrays x_train and y_train that have the shape(40000, 3072) and (40000,1) respectively.
- They will contain the last 40000 values from x_train_all and y_train_all respectively

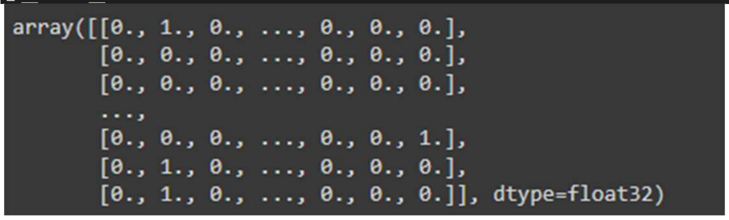
```
x_train = x_train_all[VALIDATION_SIZE:]
y_cat_train= y_cat_train_all[VALIDATION_SIZE:]
```

```
x_train.shape
```



```
(40000, 32, 32, 3)
```

```
y_cat_train
```



```
array([[0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]], dtype=float32)
```

* *FILTERS:* Typical values for the number of filters can be determined by the data set's complexity. So essentially the larger the images, the more variety and the more classes you're trying to classify then the more filters you should have.

* Most times people typically pick filter based on powers of 2, for example, 32. However, if you have more complex data like road signs etc. you should be starting with a higher filter value

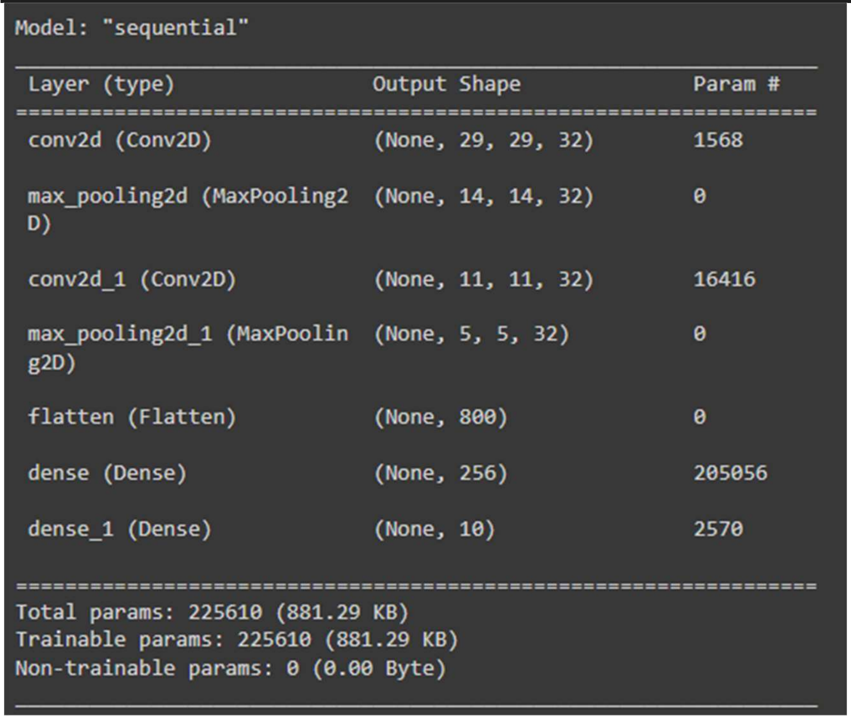
The default STRIDE value is 1 x 1 pixel

Building the model

```
model = Sequential()

## ************* FIRST SET OF LAYERS *************************

# CONVOLUTIONAL LAYER
model.add(Conv2D(filters=32, kernel_size=(4,4),input_shape=(32, 32, 3),
activation='relu',))
# POOLING LAYER
model.add(MaxPool2D(pool_size=(2, 2)))

## *************** SECOND SET OF LAYERS ***********************
#Since the shape of the data is 32 x 32 x 3 =3072 ...
#We need to deal with this more complex structure by adding yet another
convolutional layer

# ************CONVOLUTIONAL LAYER
model.add(Conv2D(filters=32, kernel_size=(4,4),input_shape=(32, 32, 3),
activation='relu',))
# POOLING LAYER
model.add(MaxPool2D(pool_size=(2, 2)))

# FLATTEN IMAGES FROM 32 x 32 x 3 =3072 BEFORE FINAL LAYER
model.add(Flatten())
```

```
# 256 NEURONS IN DENSE HIDDEN LAYER (YOU CAN CHANGE THIS NUMBER OF
NEURONS)
model.add(Dense(256, activation='relu'))

# LAST LAYER IS THE CLASSIFIER, THUS 10 POSSIBLE CLASSES
model.add(Dense(10, activation='softmax'))


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

```
model.summary()
```



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 29, 29, 32)        1568

 max_pooling2d (MaxPooling2  (None, 14, 14, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 11, 11, 32)        16416

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 32)          0
 g2D)

 flatten (Flatten)           (None, 800)               0

 dense (Dense)               (None, 256)               205056

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 225610 (881.29 KB)
Trainable params: 225610 (881.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Adding early stopping

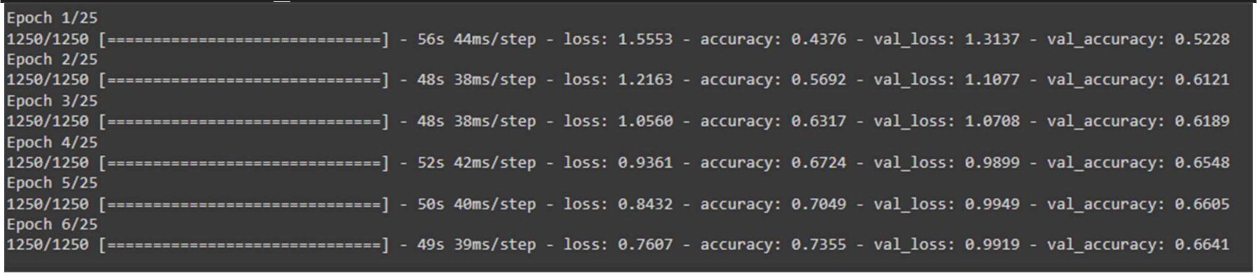
Earlystopping is a powerful utility that allows you to halt the training process of your machine learning model when a monitored metric has stopped improving. This callback can help prevent overfitting by terminating the training process early if the model's performance on a validation set ceases to improve beyond a certain threshold.

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

We want to halt the training of a TensorFlow/Keras model when the validation loss does not improve for two consecutive epochs. This setup is effective for preventing overfitting and saves computation time by stopping the training early if the model performance isn't getting any better on the validation set.

```
early_stop = EarlyStopping(monitor='val_loss',patience=2)
```

```
history =
model.fit(x_train,y_cat_train,epochs=25,validation_data=(x_val,y_val_cat),
callbacks=[early_stop])
```



```
Epoch 1/25
1250/1250 [==============================] - 56s 44ms/step - loss: 1.5553 - accuracy: 0.4376 - val_loss: 1.3137 - val_accuracy: 0.5228
Epoch 2/25
1250/1250 [==============================] - 48s 38ms/step - loss: 1.2163 - accuracy: 0.5692 - val_loss: 1.1077 - val_accuracy: 0.6121
Epoch 3/25
1250/1250 [==============================] - 48s 38ms/step - loss: 1.0560 - accuracy: 0.6317 - val_loss: 1.0708 - val_accuracy: 0.6189
Epoch 4/25
1250/1250 [==============================] - 52s 42ms/step - loss: 0.9361 - accuracy: 0.6724 - val_loss: 0.9899 - val_accuracy: 0.6548
Epoch 5/25
1250/1250 [==============================] - 50s 40ms/step - loss: 0.8432 - accuracy: 0.7049 - val_loss: 0.9949 - val_accuracy: 0.6605
Epoch 6/25
1250/1250 [==============================] - 49s 39ms/step - loss: 0.7607 - accuracy: 0.7355 - val_loss: 0.9919 - val_accuracy: 0.6641
```

Below is a break down of the above code

history = model.fit(

x_train, # Training data features

y_cat_train, # Training data labels, assumed to be one-hot encoded

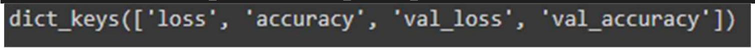
epochs=25, # Total number of epochs to train for, unless early stopping is triggered

validation_data=(x_val, y_val_cat), # Validation data used to monitor the model's performance

callbacks=[early_stop] # List of callbacks, here only EarlyStopping is used

)

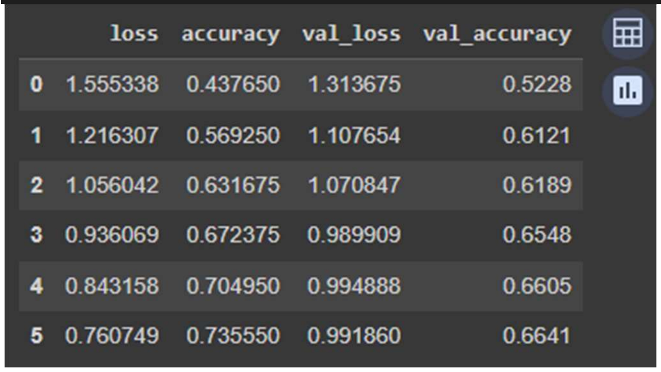
```
model.history.history.keys()
```



```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

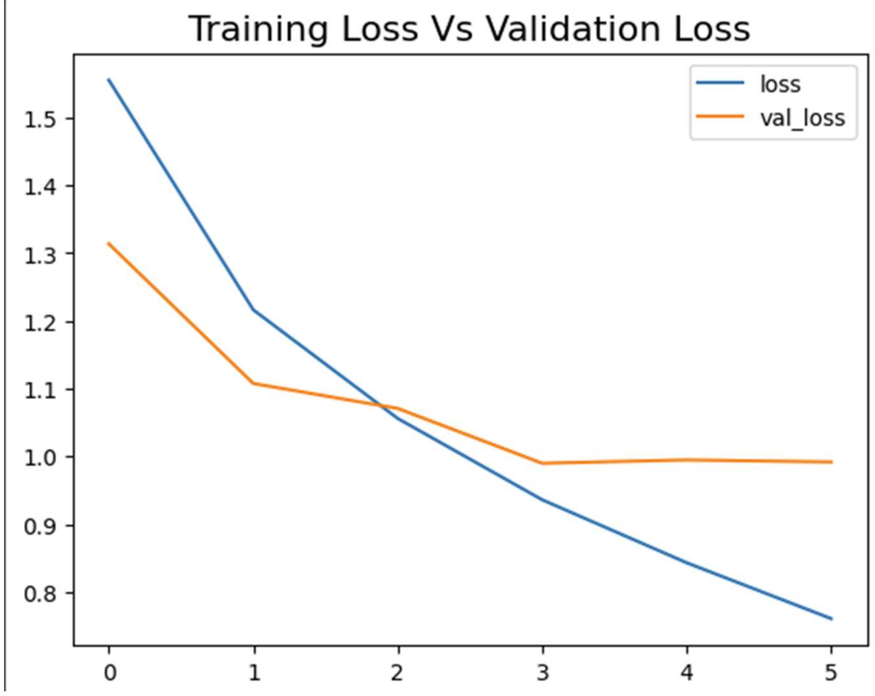
```
metrics = pd.DataFrame(model.history.history)
```

```
metrics
```

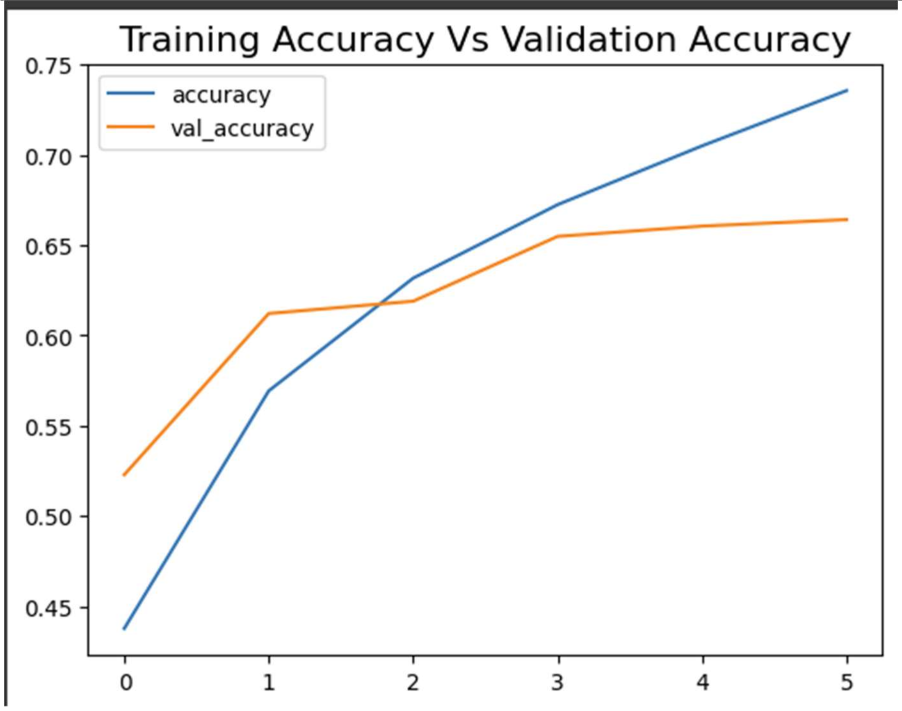


	loss	accuracy	val_loss	val_accuracy
0	1.555338	0.437650	1.313675	0.5228
1	1.216307	0.569250	1.107654	0.6121
2	1.056042	0.631675	1.070847	0.6189
3	0.936069	0.672375	0.989909	0.6548
4	0.843158	0.704950	0.994888	0.6605
5	0.760749	0.735550	0.991860	0.6641

```
metrics[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
```



```
metrics[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
```



Validating on test data

```
model.evaluate(x_test,y_cat_test)
```

```
313/313 [==============================] - 4s 13ms/step - loss: 1.0073 - accuracy: 0.6519
[1.0073405504226685, 0.6518999934196472]
```

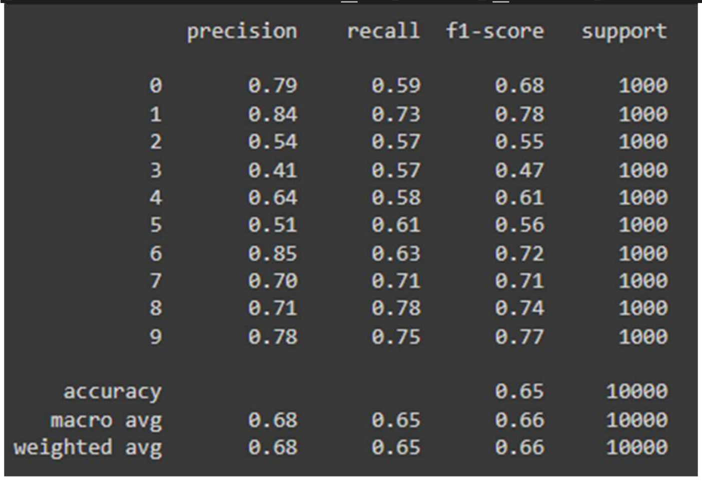
Classification report and confusion matrix

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

```
#predictions = model.predict_classes(x_test)
predictions = np.argmax(model.predict(x_test), axis=-1)
```

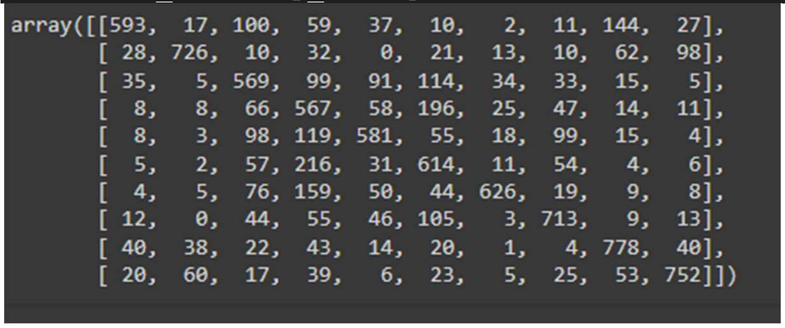
```
313/313 [==============================] - 5s 15ms/step
```

```
print(classification_report(y_test,predictions))
```



```
              precision    recall  f1-score   support

           0       0.79      0.59      0.68      1000
           1       0.84      0.73      0.78      1000
           2       0.54      0.57      0.55      1000
           3       0.41      0.57      0.47      1000
           4       0.64      0.58      0.61      1000
           5       0.51      0.61      0.56      1000
           6       0.85      0.63      0.72      1000
           7       0.70      0.71      0.71      1000
           8       0.71      0.78      0.74      1000
           9       0.78      0.75      0.77      1000

    accuracy                           0.65     10000
   macro avg       0.68      0.65      0.66     10000
weighted avg       0.68      0.65      0.66     10000
```

```
confusion_matrix(y_test,predictions)
```

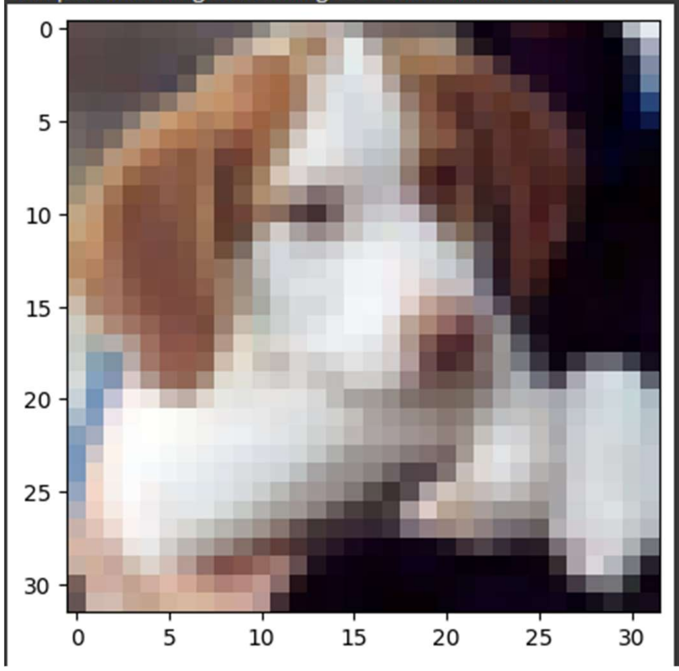


```
array([[593,  17, 100,  59,  37,  10,   2,  11, 144,  27],
       [ 28, 726,  10,  32,   0,  21,  13,  10,  62,  98],
       [ 35,   5, 569,  99,  91, 114,  34,  33,  15,   5],
       [  8,   8,  66, 567,  58, 196,  25,  47,  14,  11],
       [  8,   3,  98, 119, 581,  55,  18,  99,  15,   4],
       [  5,   2,  57, 216,  31, 614,  11,  54,   4,   6],
       [  4,   5,  76, 159,  50,  44, 626,  19,   9,   8],
       [ 12,   0,  44,  55,  46, 105,   3, 713,   9,  13],
       [ 40,  38,  22,  43,  14,  20,   1,   4, 778,  40],
       [ 20,  60,  17,  39,   6,  23,   5,  25,  53, 752]])
```

Predicting on single image

```
plt.imshow(x_test[16])
```

```
<matplotlib.image.AxesImage at 0x7ca048467010>
```



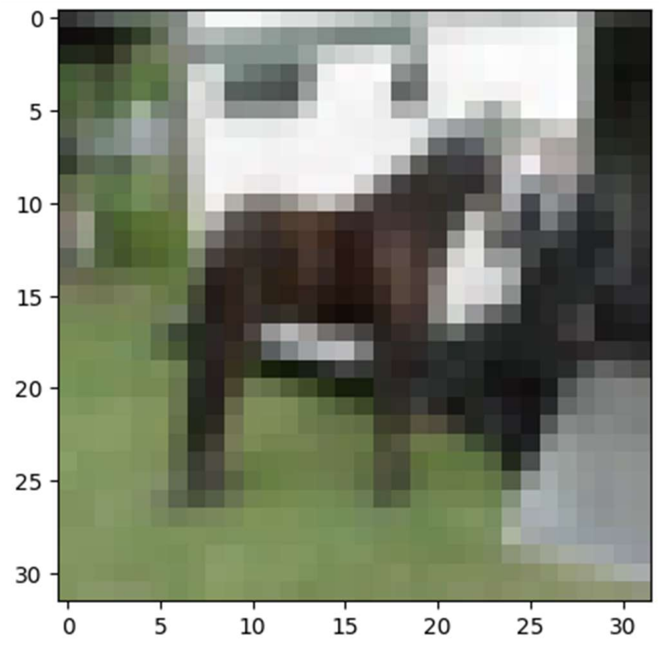
```
my_image = x_test[16]
```

```
# Assuming my_image is a numpy array with shape (32, 32, 3)
# Reshape it to (1, 32, 32, 3) to prepare it as input for the model,
matching the expected input shape
my_image_reshaped = my_image.reshape(1, 32, 32, 3)

# Predict using the model
predictions = model.predict(my_image_reshaped)

# Extract the class index with the highest probability
predicted_class_index = np.argmax(predictions, axis=1)

print("Predicted class index:", predicted_class_index)
```

```
LABEL_NAMES[y_test[16][0]]
```

```
'dog'
```

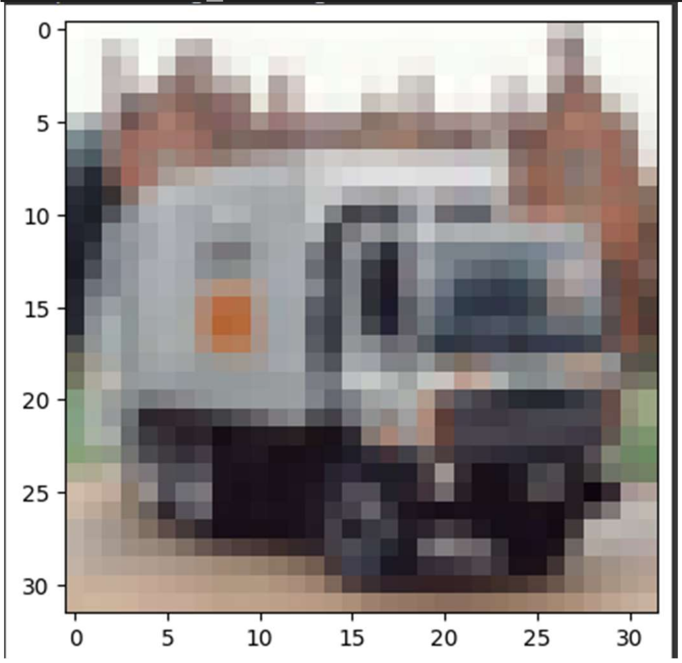
```
plt.imshow(x_test[20])
```



```
my_image = x_test[20]
```

```
'horse'
```

```
plt.imshow(x_test[11])
```



```
my_image = x_test[11]
```

```
LABEL_NAMES[y_test[11][0]]
```

```
'truck'
```