Project- Machine Learning

July 27, 2023

1 Machine Learning Module

1.1 Implementing convolutional neural network for image classification(CIFAR 10 dataset)

[ ]:
```
#Import the necessary libraries
from tensorflow.keras import Sequential, datasets
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
import numpy as np
from matplotlib import pyplot as plt
from tensorflow.keras.utils import to_categorical
```

[ ]:
```
#Load the cifar10 dataset and assign it as a tuple as follows
(X_train_all, y_train_all),(X_test,y_test)=datasets.cifar10.load_data()
```

[3]:
```
#Dimensions of the X_train array(number of images,length of the image,width of
 ↪the image,number of channels-rgb channel)
X_train_all.shape
```

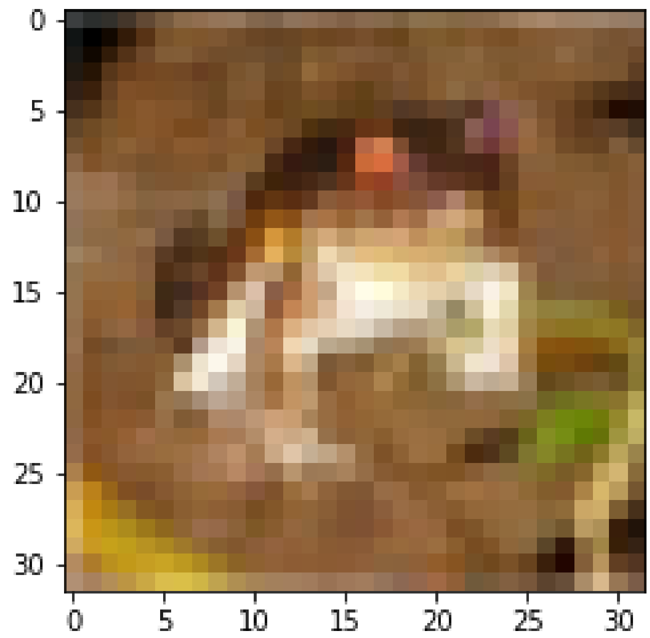
[3]: (50000, 32, 32, 3)

[4]:
```
#Dimenisons of the X_test array
X_test.shape
```

[4]: (10000, 32, 32, 3)

[5]:
```
#Use matplotlib to view the image
plt.imshow(X_train_all[0])
```

[5]: <matplotlib.image.AxesImage at 0x1833dbe7430>



[6]:
```
y_train_all
```

[6]:
```
array([[6],
       [9],
       [9],
       …,
       [9],
       [1],
       [1]], dtype=uint8)
```

[7]:
```
y_train_all.shape
```

[7]:
```
(50000, 1)
```

[8]:
```
#We have a 2 dimension numpy array
y_train_all[0][0]
```

[8]:
```
6
```

[9]:
```
LABEL_NAMES = ['airplane', 'automobile','bird','cat', 'deer', 'dog', 'frog',␣
↪'horse', 'ship', 'truck']
```

[10]:
```
#Using the lable names to get the actual names of classes
LABEL_NAMES[y_train_all[0][0]]
```

[10]:
```
'frog'
```

[11]:
```python
#The shape of the image
#32, 32 is the width and the height
#3 is the number of channels (These are the number of colors): Red, Green &
↪Blue (RGB)
X_train_all.shape
```

[11]:
```
(50000, 32, 32, 3)
```

[12]:
```python
#For the X_train data sample:
number_of_images, width, height, number_of_channels=X_train_all.shape
print(f'Number of images = {number_of_images} \t| width = {width} \t| height =
↪{height} \t| channels = {number_of_channels}')
```

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

[13]:
```python
X_test.shape
```

[13]:
```
(10000, 32, 32, 3)
```

[14]:
```python
#Normalise the data so that to make our machine learning model, be able to
↪learn easier from the data
#Each pixel value of the photo(for every channel) takes a value within the
↪range of 0-255 so we divide each pixel vbalue with 255 and transform the
↪range between 0-1
X_train_all =X_train_all / 255.0
X_test = X_test / 255.0
```

[15]:
```python
y_train_all
```

[15]:
```
array([[6],
       [9],
       [9],
       …,
       [9],
       [1],
       [1]], dtype=uint8)
```

[16]:
```python
y_test
```

[16]:
```
array([[3],
       [8],
       [8],
       …,
       [5],
       [1],
       [7]], dtype=uint8)
```

[17]:
```
#Creating the categorical encoding for the 'y_train' and 'y_test' data
y_categorical_train_all = to_categorical(y_train_all,10)
y_categorical_test = to_categorical(y_test,10)
```

[18]:
```
y_categorical_train_all
```

[18]:
```
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)
```

[19]:
```
y_categorical_test
```

[19]:
```
array([[0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 1., 0.],
       [0., 0., 0., …, 0., 1., 0.],
       …,
       [0., 0., 0., …, 0., 0., 0.],
       [0., 1., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 1., 0., 0.]], dtype=float32)
```

[20]:
```
#Creating the validation dataset
#Utilisng a validation dataset we ensure that only the our best version of our
 ↪model reaches our testing dataset
```

[21]:
```
VALIDATION_SIZE = 10000
```

[22]:
```
x_validation=X_train_all[:VALIDATION_SIZE]
y_validation_categorical = y_categorical_train_all[:VALIDATION_SIZE]
```

[23]:
```
x_validation.shape
```

[23]:
```
(10000, 32, 32, 3)
```

[24]:
```
y_validation_categorical.shape
```

[24]:
```
(10000, 10)
```

[25]:
```
x_validation
```

[25]:
```
array([[[[0.23137255, 0.24313725, 0.24705882],
         [0.16862745, 0.18039216, 0.17647059],
         [0.19607843, 0.18823529, 0.16862745],
         …,
         [0.61960784, 0.51764706, 0.42352941],
```

```
  [0.59607843, 0.49019608, 0.4       ],
  [0.58039216, 0.48627451, 0.40392157]],

 [[0.0627451 , 0.07843137, 0.07843137],
  [0.        , 0.        , 0.        ],
  [0.07058824, 0.03137255, 0.        ],
  …,
  [0.48235294, 0.34509804, 0.21568627],
  [0.46666667, 0.3254902 , 0.19607843],
  [0.47843137, 0.34117647, 0.22352941]],

 [[0.09803922, 0.09411765, 0.08235294],
  [0.0627451 , 0.02745098, 0.        ],
  [0.19215686, 0.10588235, 0.03137255],
  …,
  [0.4627451 , 0.32941176, 0.19607843],
  [0.47058824, 0.32941176, 0.19607843],
  [0.42745098, 0.28627451, 0.16470588]],

 …,

 [[0.81568627, 0.66666667, 0.37647059],
  [0.78823529, 0.6       , 0.13333333],
  [0.77647059, 0.63137255, 0.10196078],
  …,
  [0.62745098, 0.52156863, 0.2745098 ],
  [0.21960784, 0.12156863, 0.02745098],
  [0.20784314, 0.13333333, 0.07843137]],

 [[0.70588235, 0.54509804, 0.37647059],
  [0.67843137, 0.48235294, 0.16470588],
  [0.72941176, 0.56470588, 0.11764706],
  …,
  [0.72156863, 0.58039216, 0.36862745],
  [0.38039216, 0.24313725, 0.13333333],
  [0.3254902 , 0.20784314, 0.13333333]],

 [[0.69411765, 0.56470588, 0.45490196],
  [0.65882353, 0.50588235, 0.36862745],
  [0.70196078, 0.55686275, 0.34117647],
  …,
  [0.84705882, 0.72156863, 0.54901961],
  [0.59215686, 0.4627451 , 0.32941176],
  [0.48235294, 0.36078431, 0.28235294]]],


[[[0.60392157, 0.69411765, 0.73333333],
```

```
 [0.49411765, 0.5372549 , 0.53333333],
 [0.41176471, 0.40784314, 0.37254902],
 …,
 [0.35686275, 0.37254902, 0.27843137],
 [0.34117647, 0.35294118, 0.27843137],
 [0.30980392, 0.31764706, 0.2745098 ]],

[[0.54901961, 0.62745098, 0.6627451 ],
 [0.56862745, 0.6       , 0.60392157],
 [0.49019608, 0.49019608, 0.4627451 ],
 …,
 [0.37647059, 0.38823529, 0.30588235],
 [0.30196078, 0.31372549, 0.24313725],
 [0.27843137, 0.28627451, 0.23921569]],

[[0.54901961, 0.60784314, 0.64313725],
 [0.54509804, 0.57254902, 0.58431373],
 [0.45098039, 0.45098039, 0.43921569],
 …,
 [0.30980392, 0.32156863, 0.25098039],
 [0.26666667, 0.2745098 , 0.21568627],
 [0.2627451 , 0.27058824, 0.21568627]],

…,

[[0.68627451, 0.65490196, 0.65098039],
 [0.61176471, 0.60392157, 0.62745098],
 [0.60392157, 0.62745098, 0.66666667],
 …,
 [0.16470588, 0.13333333, 0.14117647],
 [0.23921569, 0.20784314, 0.22352941],
 [0.36470588, 0.3254902 , 0.35686275]],

[[0.64705882, 0.60392157, 0.50196078],
 [0.61176471, 0.59607843, 0.50980392],
 [0.62352941, 0.63137255, 0.55686275],
 …,
 [0.40392157, 0.36470588, 0.37647059],
 [0.48235294, 0.44705882, 0.47058824],
 [0.51372549, 0.4745098 , 0.51372549]],

[[0.63921569, 0.58039216, 0.47058824],
 [0.61960784, 0.58039216, 0.47843137],
 [0.63921569, 0.61176471, 0.52156863],
 …,
 [0.56078431, 0.52156863, 0.54509804],
 [0.56078431, 0.5254902 , 0.55686275],
```

```
 [0.56078431, 0.52156863, 0.56470588]]],


[[[1.        , 1.        , 1.        ],
  [0.99215686, 0.99215686, 0.99215686],
  [0.99215686, 0.99215686, 0.99215686],
  …,
  [0.99215686, 0.99215686, 0.99215686],
  [0.99215686, 0.99215686, 0.99215686],
  [0.99215686, 0.99215686, 0.99215686]],

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  …,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 [[1.        , 1.        , 1.        ],
  [0.99607843, 0.99607843, 0.99607843],
  [0.99607843, 0.99607843, 0.99607843],
  …,
  [0.99607843, 0.99607843, 0.99607843],
  [0.99607843, 0.99607843, 0.99607843],
  [0.99607843, 0.99607843, 0.99607843]],

 …,

 [[0.44313725, 0.47058824, 0.43921569],
  [0.43529412, 0.4627451 , 0.43529412],
  [0.41176471, 0.43921569, 0.41568627],
  …,
  [0.28235294, 0.31764706, 0.31372549],
  [0.28235294, 0.31372549, 0.30980392],
  [0.28235294, 0.31372549, 0.30980392]],

 [[0.43529412, 0.4627451 , 0.43137255],
  [0.40784314, 0.43529412, 0.40784314],
  [0.38823529, 0.41568627, 0.38431373],
  …,
  [0.26666667, 0.29411765, 0.28627451],
  [0.2745098 , 0.29803922, 0.29411765],
  [0.30588235, 0.32941176, 0.32156863]],

 [[0.41568627, 0.44313725, 0.41176471],
  [0.38823529, 0.41568627, 0.38431373],
```

```
  [0.37254902, 0.4       , 0.36862745],
  …,
  [0.30588235, 0.33333333, 0.3254902 ],
  [0.30980392, 0.33333333, 0.3254902 ],
  [0.31372549, 0.3372549 , 0.32941176]]],


…,


[[[0.27843137, 0.30196078, 0.17254902],
  [0.23529412, 0.27058824, 0.1372549 ],
  [0.29019608, 0.3254902 , 0.21960784],
  …,
  [0.98431373, 0.98431373, 0.99215686],
  [1.        , 1.        , 1.        ],
  [0.74901961, 0.75294118, 0.72941176]],

 [[0.34901961, 0.37647059, 0.24313725],
  [0.31372549, 0.34901961, 0.22745098],
  [0.2627451 , 0.30196078, 0.2       ],
  …,
  [0.98039216, 0.97254902, 0.97647059],
  [1.        , 0.99607843, 0.99607843],
  [0.70980392, 0.70980392, 0.66666667]],

 [[0.29019608, 0.31372549, 0.18039216],
  [0.27058824, 0.30196078, 0.19607843],
  [0.26666667, 0.30196078, 0.20784314],
  …,
  [0.96470588, 0.96078431, 0.97254902],
  [0.99607843, 0.99607843, 0.99607843],
  [0.75686275, 0.76470588, 0.71764706]],

 …,

 [[0.32941176, 0.33333333, 0.32941176],
  [0.31372549, 0.31764706, 0.31764706],
  [0.31372549, 0.31372549, 0.31372549],
  …,
  [0.16078431, 0.16862745, 0.18823529],
  [0.21568627, 0.21568627, 0.23137255],
  [0.23921569, 0.24313725, 0.24705882]],

 [[0.2627451 , 0.25882353, 0.2627451 ],
  [0.25490196, 0.25098039, 0.25098039],
  [0.24313725, 0.23921569, 0.23921569],
```

```
 …,
 [0.24705882, 0.24705882, 0.25490196],
 [0.26666667, 0.25882353, 0.2627451 ],
 [0.27843137, 0.2745098 , 0.2627451 ]],

[[0.25882353, 0.16862745, 0.10980392],
 [0.25882353, 0.17254902, 0.10980392],
 [0.25490196, 0.16470588, 0.10588235],
 …,
 [0.31372549, 0.28627451, 0.26666667],
 [0.3254902 , 0.29411765, 0.27058824],
 [0.32941176, 0.30196078, 0.26666667]]],


[[[0.98039216, 1.        , 1.        ],
 [0.99607843, 0.99215686, 0.99607843],
 [0.82745098, 0.87843137, 0.8627451 ],
 …,
 [0.7372549 , 0.76470588, 0.75686275],
 [1.        , 1.        , 1.        ],
 [1.        , 0.99607843, 1.        ]],

[[0.98039216, 1.        , 1.        ],
 [1.        , 0.99607843, 1.        ],
 [0.83529412, 0.88235294, 0.87058824],
 …,
 [0.74117647, 0.77647059, 0.76470588],
 [1.        , 1.        , 1.        ],
 [1.        , 0.99607843, 1.        ]],

[[0.98039216, 1.        , 1.        ],
 [1.        , 0.99607843, 1.        ],
 [0.83529412, 0.88627451, 0.87058824],
 …,
 [0.72941176, 0.77647059, 0.76078431],
 [1.        , 1.        , 1.        ],
 [1.        , 0.99607843, 1.        ]],

…,

[[1.        , 0.99607843, 1.        ],
 [0.99607843, 0.99607843, 1.        ],
 [0.83529412, 0.87843137, 0.86666667],
 …,
 [0.82745098, 0.82745098, 0.84313725],
 [1.        , 1.        , 1.        ],
 [1.        , 0.99607843, 0.99607843]],
```

```
 [[1.        , 0.99607843, 1.        ],
  [0.98823529, 0.99215686, 0.99215686],
  [0.85098039, 0.88235294, 0.87058824],
  …,
  [0.83921569, 0.83529412, 0.84705882],
  [1.        , 1.        , 1.        ],
  [1.        , 0.99607843, 0.99607843]],

 [[1.        , 0.99607843, 1.        ],
  [0.99215686, 0.99215686, 0.99215686],
  [0.85490196, 0.86666667, 0.8627451 ],
  …,
  [0.83921569, 0.83137255, 0.84313725],
  [1.        , 1.        , 1.        ],
  [1.        , 0.99607843, 0.99607843]]],


[[[0.24313725, 0.21568627, 0.02745098],
  [0.23921569, 0.21568627, 0.02745098],
  [0.23529412, 0.21568627, 0.02352941],
  …,
  [0.25098039, 0.22745098, 0.03921569],
  [0.32156863, 0.29411765, 0.10980392],
  [0.24313725, 0.21568627, 0.03529412]],

 [[0.24313725, 0.21960784, 0.02745098],
  [0.24705882, 0.20392157, 0.03137255],
  [0.23921569, 0.18823529, 0.02352941],
  …,
  [0.30196078, 0.27058824, 0.10980392],
  [0.44705882, 0.41960784, 0.25490196],
  [0.25098039, 0.22352941, 0.04705882]],

 [[0.2627451 , 0.23137255, 0.05882353],
  [0.30588235, 0.24313725, 0.10980392],
  [0.45098039, 0.36470588, 0.25882353],
  …,
  [0.39215686, 0.36078431, 0.22745098],
  [0.46666667, 0.43529412, 0.29019608],
  [0.24705882, 0.21960784, 0.04705882]],

 …,

 [[0.63137255, 0.75294118, 0.49019608],
  [0.62352941, 0.74509804, 0.48235294],
  [0.62352941, 0.74509804, 0.48235294],
```

```
          …,
          [0.59607843, 0.65490196, 0.48235294],
          [0.61568627, 0.74509804, 0.50196078],
          [0.61176471, 0.75294118, 0.48235294]],

         [[0.63921569, 0.76470588, 0.49803922],
          [0.63137255, 0.75294118, 0.49019608],
          [0.63529412, 0.75686275, 0.49411765],
          …,
          [0.63529412, 0.74901961, 0.50196078],
          [0.63137255, 0.75686275, 0.49803922],
          [0.63137255, 0.75686275, 0.49411765]],

         [[0.6627451 , 0.78823529, 0.51764706],
          [0.65490196, 0.77647059, 0.50980392],
          [0.65490196, 0.77647059, 0.51372549],
          …,
          [0.65490196, 0.77647059, 0.50980392],
          [0.65490196, 0.77647059, 0.50980392],
          [0.65490196, 0.77647059, 0.51372549]]]])
```

[26]:
```
y_validation_categorical
```

[26]:
```
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)
```

[27]:
```
#The last 40000 values will consist the training dataset
X_train = X_train_all[VALIDATION_SIZE:]
y_categorical_train= y_categorical_train_all[VALIDATION_SIZE:]
```

[28]:
```
X_train.shape
```

[28]:
```
(40000, 32, 32, 3)
```

[29]:
```
y_categorical_train.shape
```

[29]:
```
(40000, 10)
```

[30]:
```
#BUILD THE MODEL
```

[33]:
```
# Develop the model
model=Sequential()
```

```python
# FIRST SET OF LAYERS
# Add a convolutional layer
model.
 ↪add(Conv2D(filters=64,strides=1,kernel_size=(3,3),activation='relu',input_shape=(32,32,3)))
# Add a pooling layer
model.add(MaxPooling2D(pool_size=(2,2)))

#SECOND SET OF LAYERS
#Add another convolutional layer
model.add(Conv2D(filters=64,strides=1,kernel_size=(3,3),activation='relu'))
# Add another pulling layer
model.add(MaxPooling2D(pool_size=(2,2)))

#Flatten our output for the subsequent dense layers(final layers)
model.add(Flatten())

#256 neurons in dense hidden layer-this number is changable depending on the␣
 ↪complexity
model.add(Dense(units=256,activation='relu'))
#Last dense layer is the classifier, thus we select 10 neurons(10 possible␣
 ↪classes)
model.add(Dense(units=10,activation='softmax'))
```

[34]:
```python
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

[35]:
```python
model.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 30, 30, 64)        1792

 max_pooling2d_2 (MaxPoolin  (None, 15, 15, 64)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 13, 13, 64)        36928

 max_pooling2d_3 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 flatten_1 (Flatten)         (None, 2304)              0

 dense_2 (Dense)             (None, 256)               590080

 dense_3 (Dense)             (None, 10)                2570
```

```
=================================================================
Total params: 631370 (2.41 MB)
Trainable params: 631370 (2.41 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[36]:
```
#Adding early stopping
from tensorflow.keras.callbacks import EarlyStopping
```

[37]:
```
#Early stopping is a technique which stops model training when parameter␣
↪updates remain the same on a validation test
early_stop = EarlyStopping(monitor='val_loss',patience=2)
```

[38]:
```
history = model.
↪fit(X_train,y_categorical_train,epochs=25,validation_data=(x_validation,y_validation_catego
```

```
Epoch 1/25
1250/1250 [==============================] - 144s 113ms/step - loss: 1.4288 -
accuracy: 0.4866 - val_loss: 1.0968 - val_accuracy: 0.6082
Epoch 2/25
1250/1250 [==============================] - 124s 99ms/step - loss: 1.0548 -
accuracy: 0.6306 - val_loss: 1.0031 - val_accuracy: 0.6469
Epoch 3/25
1250/1250 [==============================] - 129s 103ms/step - loss: 0.8960 -
accuracy: 0.6877 - val_loss: 0.9103 - val_accuracy: 0.6873
Epoch 4/25
1250/1250 [==============================] - 128s 102ms/step - loss: 0.7768 -
accuracy: 0.7283 - val_loss: 0.9569 - val_accuracy: 0.6671
Epoch 5/25
1250/1250 [==============================] - 129s 104ms/step - loss: 0.6643 -
accuracy: 0.7689 - val_loss: 0.8732 - val_accuracy: 0.7064
Epoch 6/25
1250/1250 [==============================] - 122s 97ms/step - loss: 0.5617 -
accuracy: 0.8032 - val_loss: 0.8912 - val_accuracy: 0.7012
Epoch 7/25
1250/1250 [==============================] - 110s 88ms/step - loss: 0.4683 -
accuracy: 0.8370 - val_loss: 1.0096 - val_accuracy: 0.6908
```

[39]:
```
model.history.history.keys()
```

[39]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[40]:
```
#Include the relevant metrics into a dataframe
import pandas as pd
metrics = pd.DataFrame(model.history.history)
```

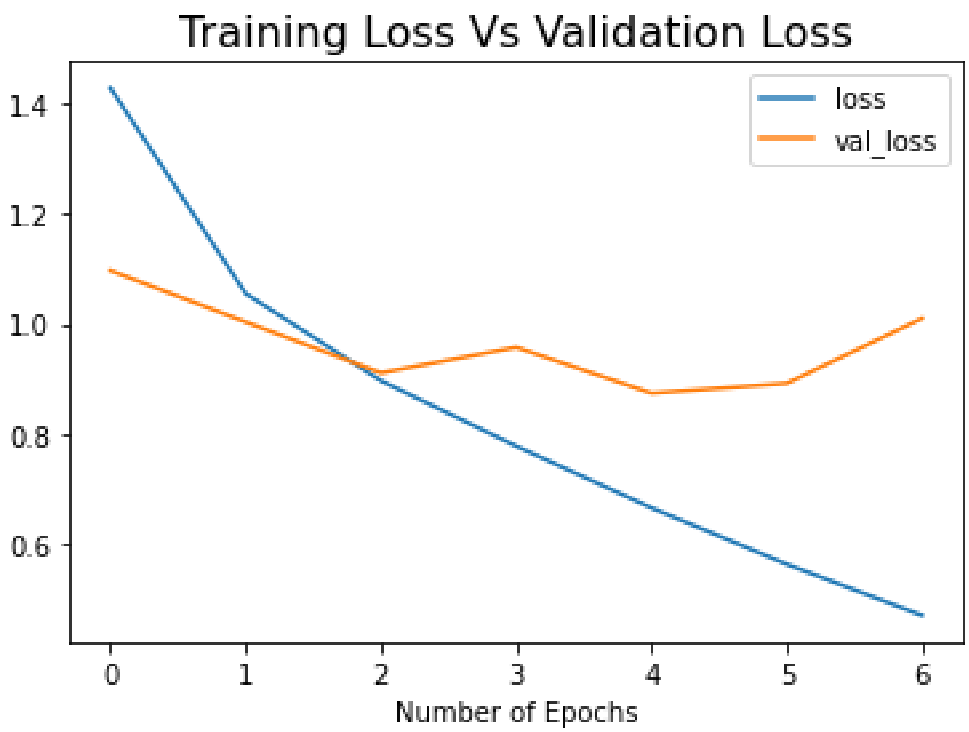
[41]:
```
metrics
```

[41]:

	loss	accuracy	val_loss	val_accuracy
0	1.428837	0.486600	1.096804	0.6082
1	1.054827	0.630550	1.003059	0.6469
2	0.896005	0.687725	0.910326	0.6873
3	0.776821	0.728325	0.956937	0.6671
4	0.664333	0.768875	0.873242	0.7064
5	0.561666	0.803200	0.891168	0.7012
6	0.468349	0.837000	1.009645	0.6908

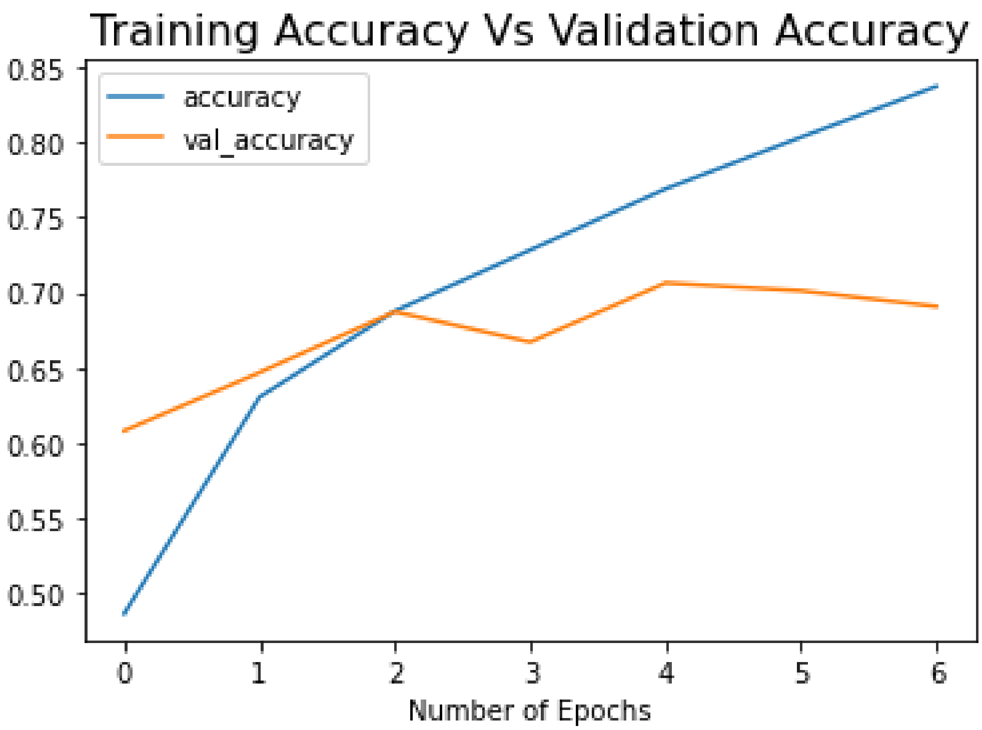
[42]:

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



[43]:

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



[44]:
```
#Evaluate the model for testing data
model.evaluate(X_test,y_categorical_test)
```

```
313/313 [==============================] - 9s 29ms/step - loss: 1.0101 -
accuracy: 0.6895
```

[44]:
```
[1.010068416595459, 0.6894999742507935]
```

[230]:
```
#Classification Report and Confusion matrix
```

[231]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

[234]:
```
predictions = model.predict(X_test)
```

```
313/313 [==============================] - 4s 11ms/step
```

[235]:
```
predictions
```

[235]:
```
array([[2.14551147e-02, 1.04081619e-03, 6.26917481e-02, …,
         1.02446543e-03, 2.99954526e-02, 3.61176426e-05],
        [1.51226655e-01, 2.28984267e-01, 8.11720895e-08, …,
         6.18736067e-08, 5.64496994e-01, 5.52845784e-02],
        [3.95336486e-02, 3.31833303e-01, 1.85838027e-03, …,
         1.34381131e-04, 6.10948920e-01, 1.39947254e-02],
```

```
       ...,
       [1.02267895e-05, 1.31167369e-07, 3.97579744e-02, ...,
        2.62154583e-02, 4.74616172e-06, 1.58402543e-07],
       [5.55975974e-01, 1.15443565e-01, 1.72174361e-03, ...,
        1.70346582e-03, 4.45578713e-03, 3.56910256e-04],
       [7.66455798e-07, 4.29643535e-07, 1.30622720e-04, ...,
        9.81209099e-01, 7.84658383e-08, 3.69861255e-07]], dtype=float32)
```

[236]:
```
predictions.shape
```

[236]:
```
(10000, 10)
```

[253]:
```
index_prediction = np.argmax(predictions,axis=-1)
```

[254]:
```
index_prediction
```

[254]:
```
array([3, 8, 8, ..., 5, 0, 7], dtype=int64)
```

[255]:
```
index_prediction.shape
```

[255]:
```
(10000,)
```

[256]:
```
print(classification_report(y_test,index_prediction))
```

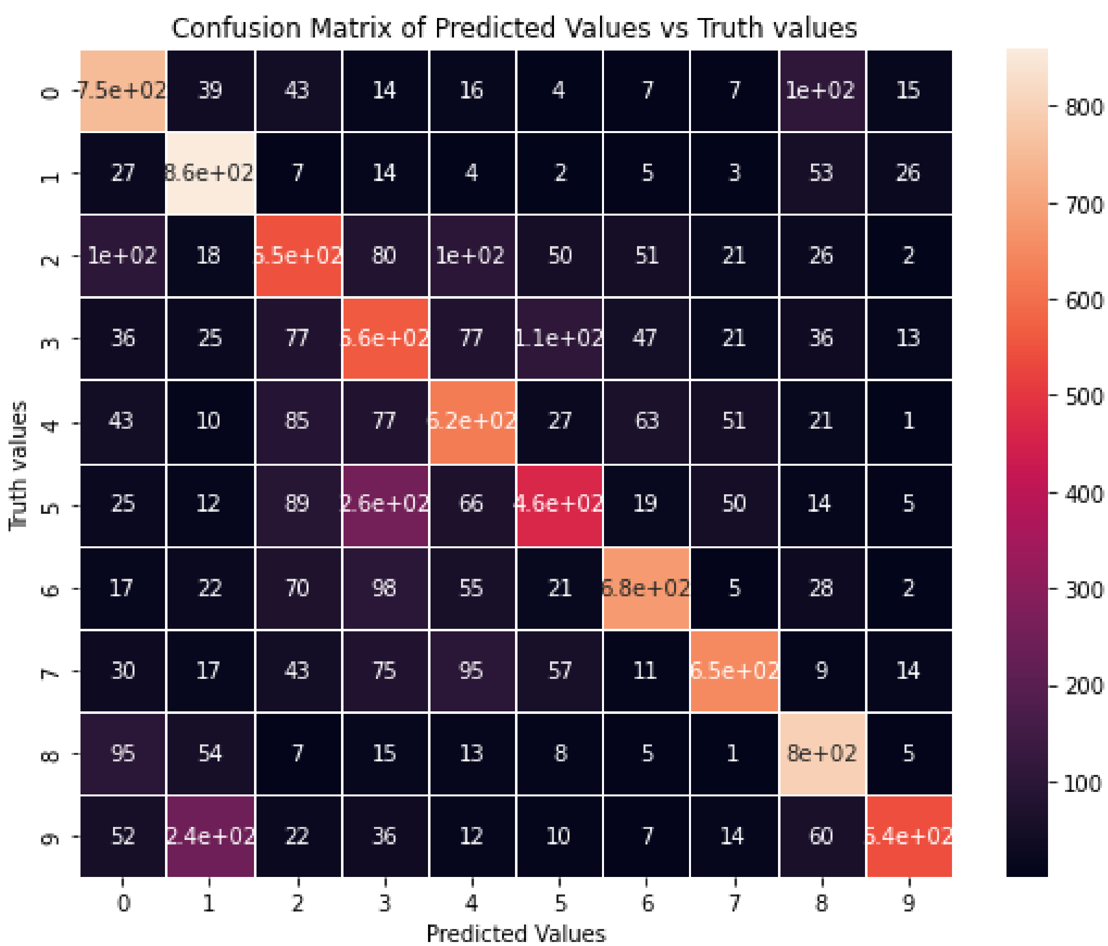
```
              precision    recall  f1-score   support

           0       0.64      0.75      0.69      1000
           1       0.66      0.86      0.75      1000
           2       0.55      0.55      0.55      1000
           3       0.46      0.56      0.50      1000
           4       0.59      0.62      0.60      1000
           5       0.62      0.47      0.53      1000
           6       0.76      0.68      0.72      1000
           7       0.79      0.65      0.71      1000
           8       0.69      0.80      0.74      1000
           9       0.87      0.54      0.67      1000

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

[267]:
```
confusion_matrix(y_test,index_prediction)
```

[267]:
```
array([[750,  39,  43,  14,  16,   4,   7,   7, 105,  15],
       [ 27, 859,   7,  14,   4,   2,   5,   3,  53,  26],
       [102,  18, 550,  80, 100,  50,  51,  21,  26,   2],
       [ 36,  25,  77, 562,  77, 106,  47,  21,  36,  13],
```

```
       [ 43,  10,  85,  77, 622,  27,  63,  51,  21,   1],
       [ 25,  12,  89, 255,  66, 465,  19,  50,  14,   5],
       [ 17,  22,  70,  98,  55,  21, 682,   5,  28,   2],
       [ 30,  17,  43,  75,  95,  57,  11, 649,   9,  14],
       [ 95,  54,   7,  15,  13,   8,   5,   1, 797,   5],
       [ 52, 244,  22,  36,  12,  10,   7,  14,  60, 543]], dtype=int64)
```

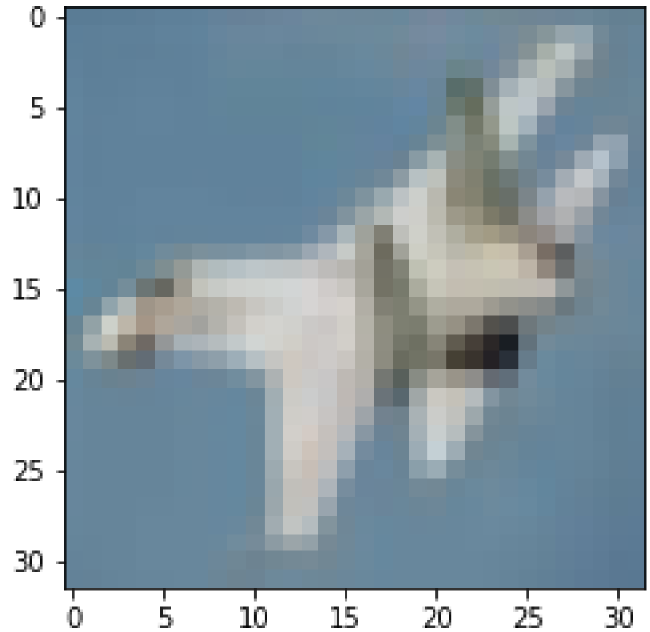
[270]:
```python
import seaborn as sns
plt.figure(figsize=(9,7))
sns.
 ↪heatmap(confusion_matrix(y_test,index_prediction),annot=True,linewidths=1,linecolor='white'
plt.ylabel('Truth values')
plt.xlabel('Predicted Values')
plt.title('Confusion Matrix of Predicted Values vs Truth values')
plt.show()
```



[ ]:
```python
#Prediction of a single image
```

[281]:
```
plt.imshow(X_test[10])
```

[281]: <matplotlib.image.AxesImage at 0x1cdcff67eb0>



[282]:
```
LABEL_NAMES[y_test[10][0]]
```

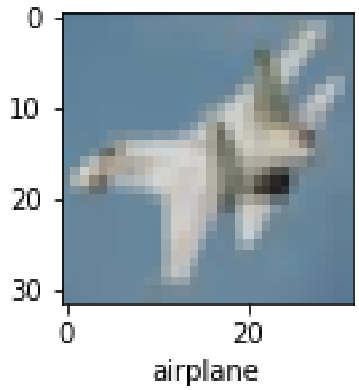
[282]: 'airplane'

[275]:
```
#The tenth image within the testing dataset is correctly labelled as an airplane
```

[275]: 6

[289]:
```
#Function for image display along with the category refference
def showImage(x,y,index):
    plt.figure(figsize=(15,2))
    plt.imshow(x[index])
    plt.xlabel(LABEL_NAMES[y[index][0]])
```

[290]:
```
showImage(X_test,y_test,10)
```


0
10
20
30
0
20
airplane

[ ]:

[ ]:

[ ]:

[ ]: