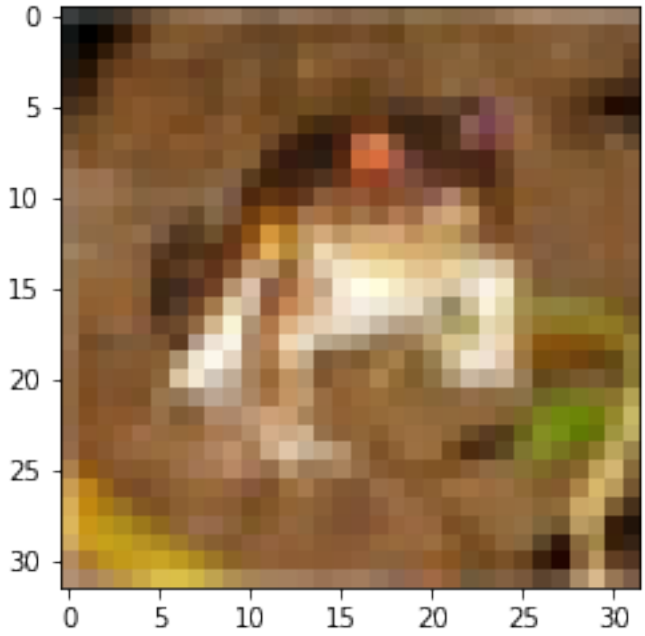
```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras.layers import Dense,Conv2D,Flatten,MaxPool2D
from tensorflow.keras import Sequential

#loading cifar10 dataset which is prebuilt in tensorflow
(xtrain,ytrain),(xtest,ytest) = tf.keras.datasets.cifar10.load_data()

#checking for data if it is loaded or not
plt.imshow(xtrain[0])
```

```
<matplotlib.image.AxesImage at 0x1f83794e7c0>
```




```
#checking for the shape of the data
xtrain.shape,xtest.shape
```

```
((50000, 32, 32, 3), (10000, 32, 32, 3))
```

```
#normalizing the data which converts data ranging from (0 to 255) ->
(0 to 1)
xtrain = xtrain/255.0
xtest = xtest/255.0
xtrain[0]
```

```
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]],
```

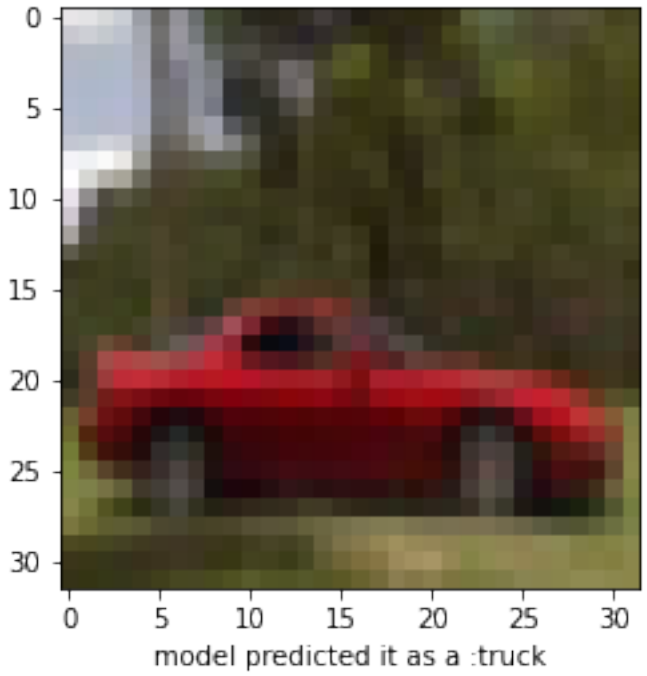
```
        [[0.0627451 , 0.07843137, 0.07843137],
         [0.        , 0.        , 0.        ],
         [0.07058824, 0.03137255, 0.        ],
         ...,
         [0.48235294, 0.34509804, 0.21568627],
         [0.46666667, 0.3254902 , 0.19607843],
         [0.47843137, 0.34117647, 0.22352941]],

        [[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]]])
```

```
#building the model
model = Sequential([Conv2D(filters=32,kernel_size=3, padding="same",
activation="relu", input_shape=[32,32,3]),
                    MaxPool2D(pool_size=2,strides=2, padding='valid'),
                    Flatten(),
                    Dense(100,activation="relu"),
                    Dense(10,activation="softmax")])
```

```
 #model needs to be compiled before it is trained on the dataset

model.compile(optimizer="sgd",loss="sparse_categorical_crossentropy",m
etrics=['accuracy'])

#training the CNN model with 10 epochs
history =
model.fit(xtrain,ytrain,validation_data=(xtest,ytest),epochs=10)

Epoch 1/10
1563/1563 [==============================] - 28s 17ms/step - loss:
1.8908 - accuracy: 0.3311 - val_loss: 1.6859 - val_accuracy: 0.4122
Epoch 2/10
1563/1563 [==============================] - 28s 18ms/step - loss:
1.5718 - accuracy: 0.4462 - val_loss: 1.4574 - val_accuracy: 0.4819
Epoch 3/10
1563/1563 [==============================] - 29s 19ms/step - loss:
1.3801 - accuracy: 0.5104 - val_loss: 1.3634 - val_accuracy: 0.5224
Epoch 4/10
1563/1563 [==============================] - 29s 19ms/step - loss:
1.2813 - accuracy: 0.5475 - val_loss: 1.2827 - val_accuracy: 0.5474
Epoch 5/10
1563/1563 [==============================] - 29s 19ms/step - loss:
1.2140 - accuracy: 0.5734 - val_loss: 1.2132 - val_accuracy: 0.5684
Epoch 6/10
1563/1563 [==============================] - 29s 19ms/step - loss:
1.1608 - accuracy: 0.5929 - val_loss: 1.2579 - val_accuracy: 0.5528
Epoch 7/10
1563/1563 [==============================] - 29s 19ms/step - loss:
1.1123 - accuracy: 0.6106 - val_loss: 1.2140 - val_accuracy: 0.5712
Epoch 8/10
1563/1563 [==============================] - 30s 19ms/step - loss:
1.0694 - accuracy: 0.6252 - val_loss: 1.1363 - val_accuracy: 0.5993
Epoch 9/10
1563/1563 [==============================] - 30s 19ms/step - loss:
1.0318 - accuracy: 0.6388 - val_loss: 1.1340 - val_accuracy: 0.6056
Epoch 10/10
1563/1563 [==============================] - 30s 19ms/step - loss:
0.9933 - accuracy: 0.6530 - val_loss: 1.1843 - val_accuracy: 0.5831

#predicting the test values
predicted = model.predict(xtest)

313/313 [==============================] - 2s 6ms/step

predicted[0]

array([0.00428185, 0.00359572, 0.02500639, 0.7028657 , 0.03333816,
       0.18724094, 0.0350663 , 0.00116917, 0.00328384, 0.00415196],
      dtype=float32)
```

```
import random
n = random.randint(0,999)

n = random.randint(0,999)
#creating a list of classes to give particular output
classes =
["airplane","automobile","bird","cat","deer","dog","frog","horse","shi
p","truck"]

#getting the index with max value
value=predicted[n][0]
index=0
for i in range(len(predicted[n])):
    if value<predicted[n][i]:
        index = i



#showing output by plotting the image and the corresponding output
plt.imshow(xtest[n])
plt.xlabel("model predicted it as a :"+classes[index])
```

```
Text(0.5, 0, 'model predicted it as a :truck')
```

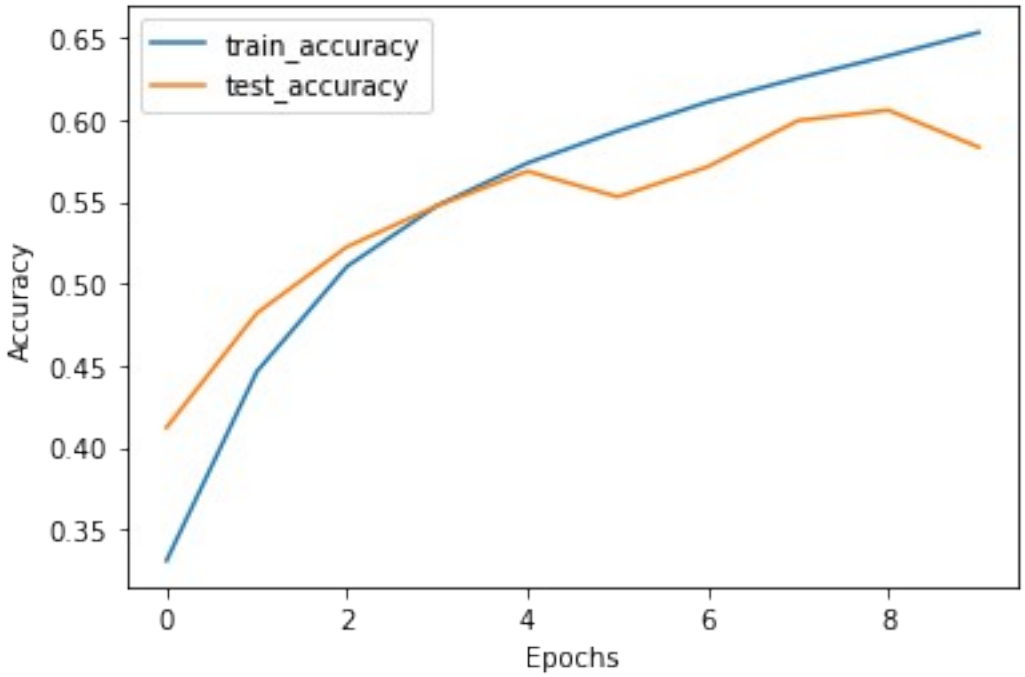



```
history.history.keys()
```

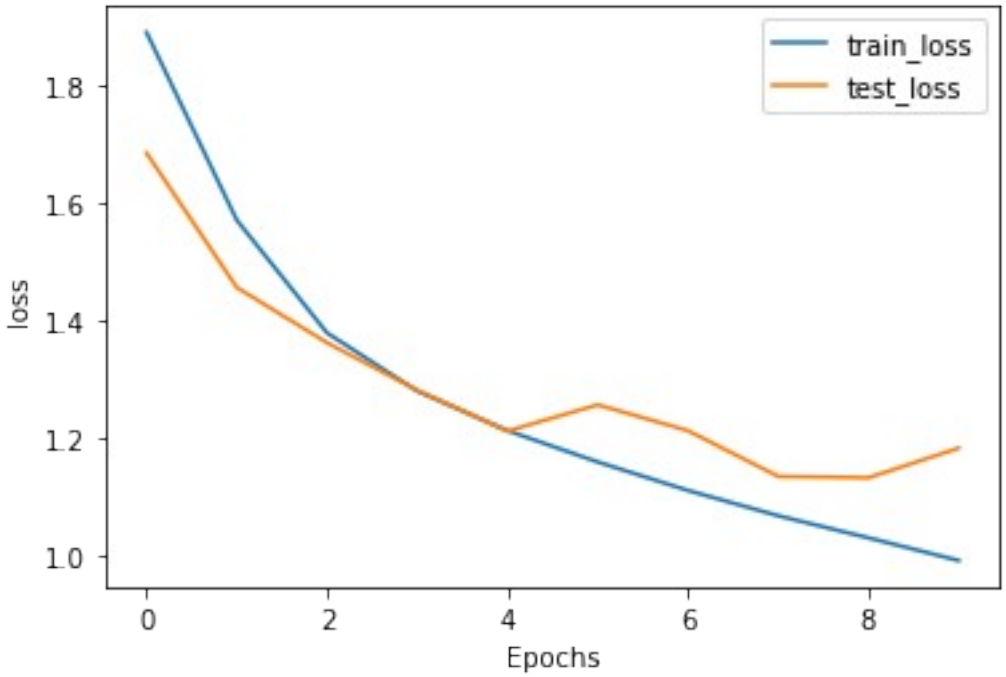
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
#plotting accuracy parameters
plt.plot(history.history['accuracy'],label="train_accuracy")
```

```
plt.plot(history.history['val_accuracy'],label="test_accuracy")
plt.legend(loc="upper left")

plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.show()
```




```
#plotting Loss parameters
plt.plot(history.history['loss'],label="train_loss")
plt.plot(history.history['val_loss'],label="test_loss")
plt.legend(loc="upper right")
plt.xlabel("Epochs")
plt.ylabel("loss")
plt.show()
```


train_loss
test_loss
1.8
1.6
1.4
1.2
1.0
loss
0
2
4
6
8
Epochs