# CNN_model1

May 9, 2021

Convolutional Neural Network (CNN)

Import libraries

[1]:

```
import numpy as np
import pandas as pd

import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Load the Images and split the train and test

[2]:

```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.
 ↪load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

[3]:

```
test_labels
```

[3]:

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

Display sample images (2x5)

[4]:

```
train_images.shape
```

[4]:

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

Making Labels and class names

[5]:

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

plt.figure(figsize=(10,4))
for i in range(10):
    plt.subplot(2,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

Model Architecture

- Create Convolution base simple and common pattern: CONV2D and MaxPooling2D layers.
- AS a input, CNN takes tensors of shape (image_height, image_width, and color channels)

[6]:

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

[7]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
_________________________________________________________________
```

The above output of CONV2D and MaxPooling2D layer is a 3D tensor of shape(height,width, and channels.

The height and width dimensions tend to shrink as you go in the deeper in the network.

Typically, as the width and height shrink, you can afford (computationally) to add more output channels in each Conv2D layer.

ADD Dense layer on Top

To complete our model, you will feed the last output tensor from the convolutional base (of shape (4, 4, 64)) into one or more Dense layers to perform classification.

[8]:

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
print("Complete architecture of our model")
```

```
Complete architecture of our model
```

[9]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

As you can see, our (4, 4, 64) outputs were flattened into vectors of shape (1024) before going through two Dense layers.

Compile and train the model

[10]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 19s 12ms/step - loss: 1.7692 -
accuracy: 0.3460 - val_loss: 1.3117 - val_accuracy: 0.5303
Epoch 2/10
1563/1563 [==============================] - 19s 12ms/step - loss: 1.1897 -
accuracy: 0.5773 - val_loss: 1.0436 - val_accuracy: 0.6269
Epoch 3/10
1563/1563 [==============================] - 20s 13ms/step - loss: 1.0064 -
accuracy: 0.6428 - val_loss: 1.0070 - val_accuracy: 0.6443
Epoch 4/10
1563/1563 [==============================] - 19s 12ms/step - loss: 0.9130 -
accuracy: 0.6792 - val_loss: 0.8990 - val_accuracy: 0.6859
Epoch 5/10
1563/1563 [==============================] - 19s 12ms/step - loss: 0.8279 -
accuracy: 0.7089 - val_loss: 0.8816 - val_accuracy: 0.6940
Epoch 6/10
1563/1563 [==============================] - 20s 13ms/step - loss: 0.7656 -
accuracy: 0.7315 - val_loss: 0.8858 - val_accuracy: 0.7023
Epoch 7/10
1563/1563 [==============================] - 19s 12ms/step - loss: 0.7280 -
accuracy: 0.7437 - val_loss: 0.8394 - val_accuracy: 0.7118
Epoch 8/10
```

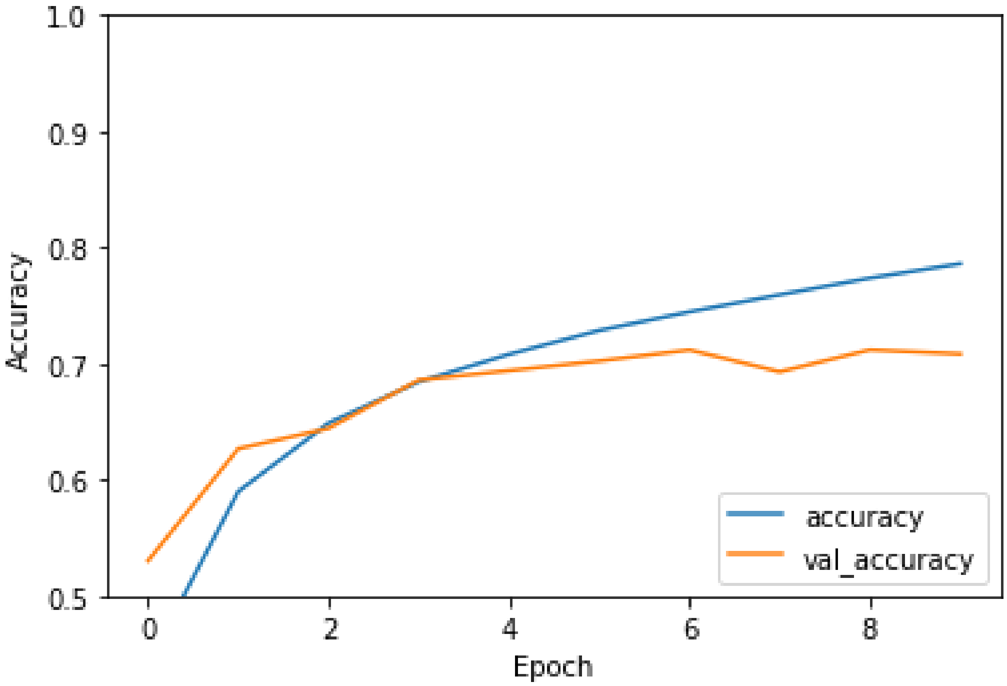
```
1563/1563 [==============================] - 20s 13ms/step - loss: 0.6768 -
accuracy: 0.7613 - val_loss: 0.9132 - val_accuracy: 0.6930
Epoch 9/10
1563/1563 [==============================] - 19s 12ms/step - loss: 0.6291 -
accuracy: 0.7803 - val_loss: 0.8685 - val_accuracy: 0.7119
Epoch 10/10
1563/1563 [==============================] - 19s 12ms/step - loss: 0.5919 -
accuracy: 0.7918 - val_loss: 0.9007 - val_accuracy: 0.7084
```

Evaluate the model

[11]:

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 1s - loss: 0.9007 - accuracy: 0.7084
```



[12]:

```
print(test_acc)
```

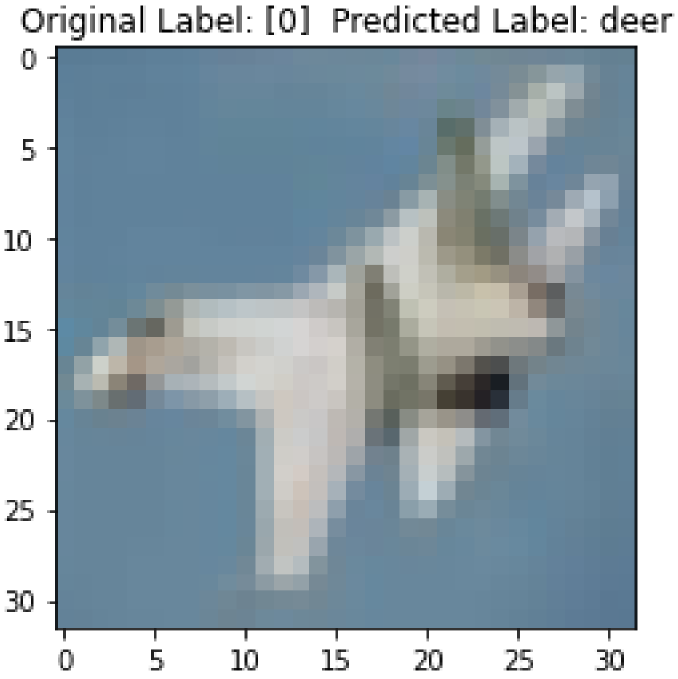
```
0.7084000110626221
```

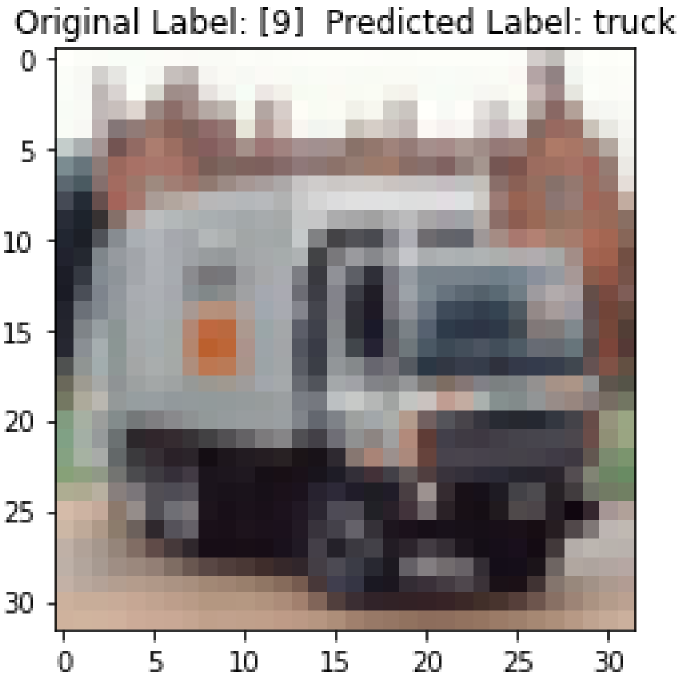
```
[13]: from sklearn.metrics import accuracy_score,confusion_matrix
      test_labels_pred = model.predict(test_images)
      test_labels_pred.shape
```

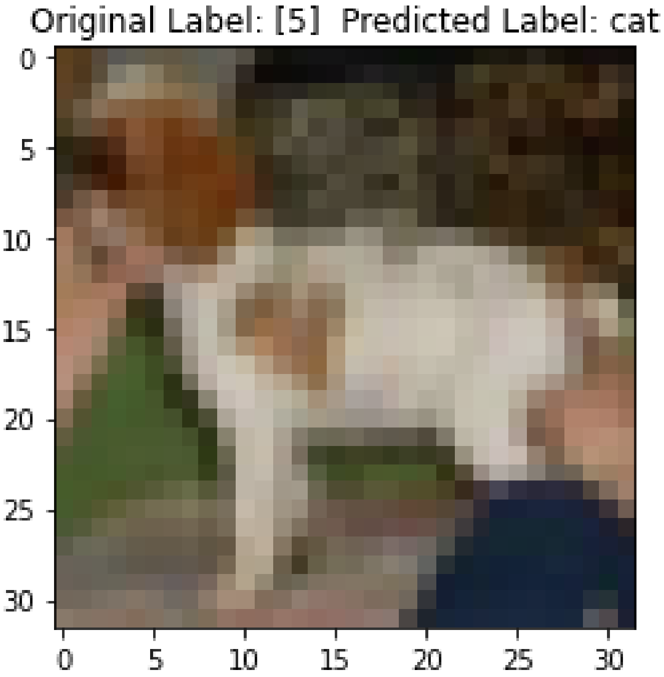
```
[13]: (10000, 10)
```

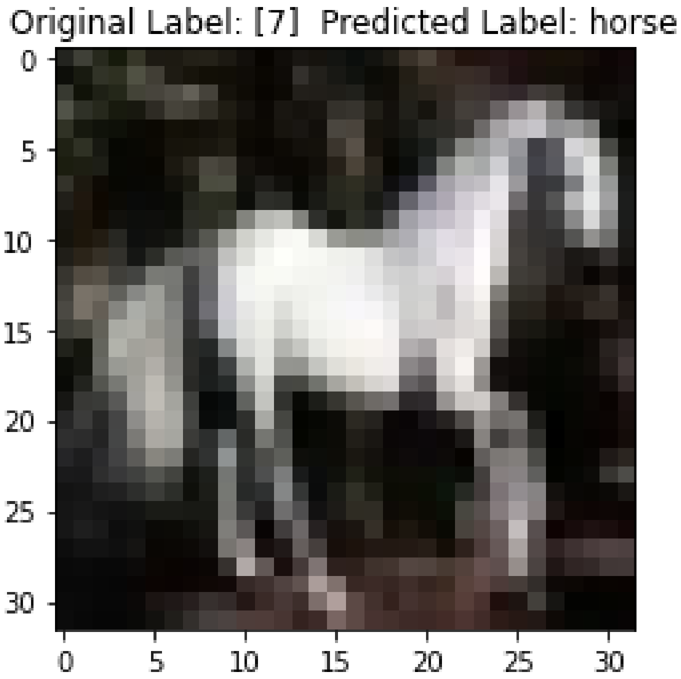
Prediction using test images

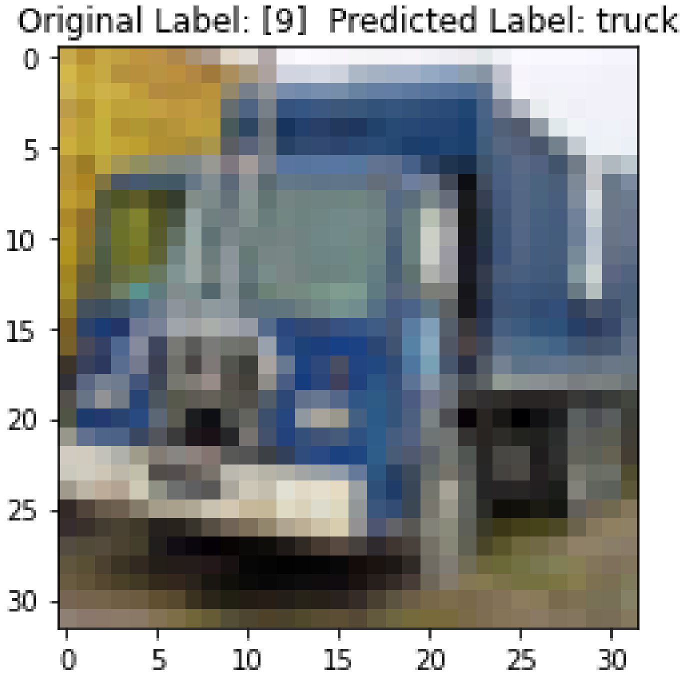
```
[14]: for i in range(10,20):
          plt.imshow(test_images[i], cmap=plt.cm.binary)

          #Image name prediction
          image = test_images[i]
          image1 = np.array(image).reshape((1,32,32,3))
          #print(image.shape,image1.shape)
          prediction = model.predict(image1)
          #print(prediction.shape)

          score = tf.nn.softmax(prediction[0])
      #     print(class_names[np.argmax(score)])
          plt.title('Original Label: {0}  Predicted Label: {1}'.
       ↪format(test_labels[i],class_names[np.argmax(score)]))
          plt.show()
```

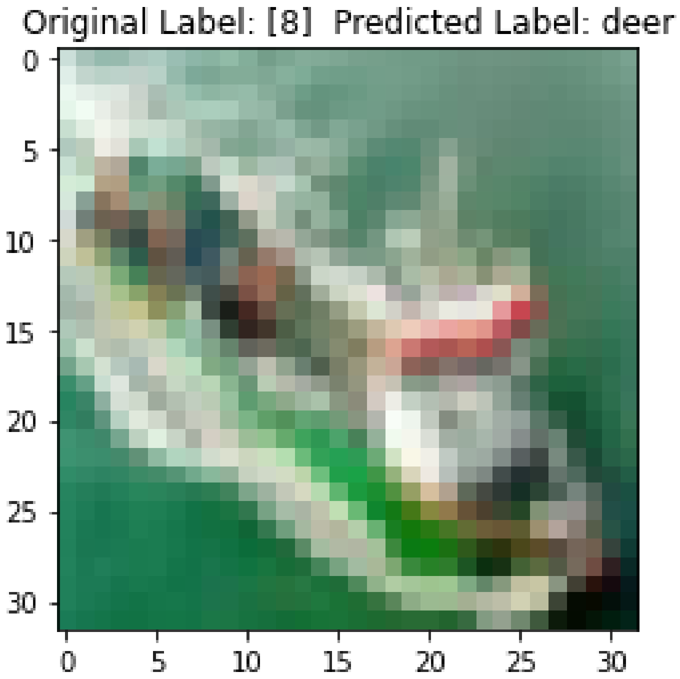


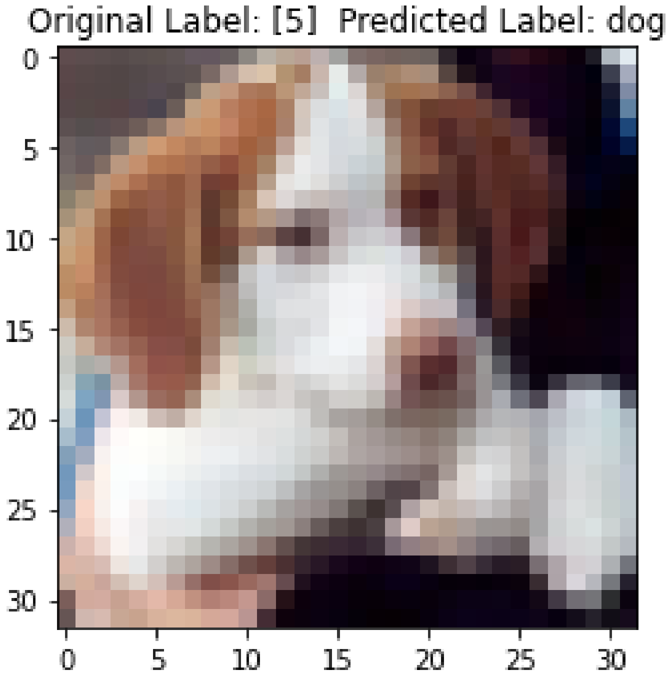

Original Label: [9]  Predicted Label: truck

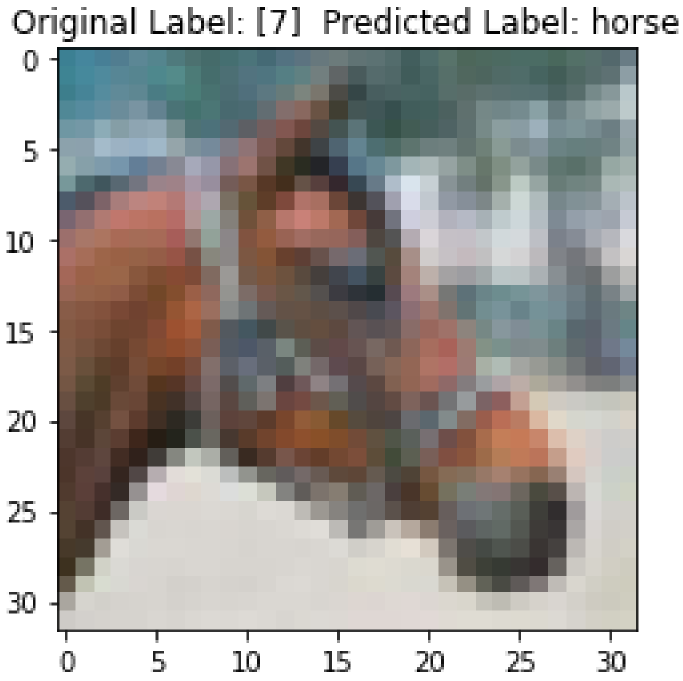

Original Label: [5]  Predicted Label: cat

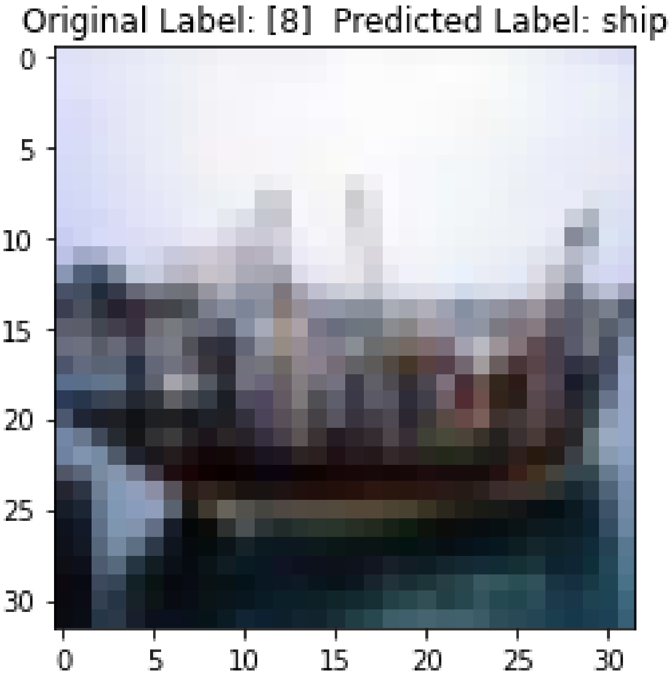

Original Label: [7]  Predicted Label: horse
0
5
10
15
20
25
30
0
5
10
15
20
25
30

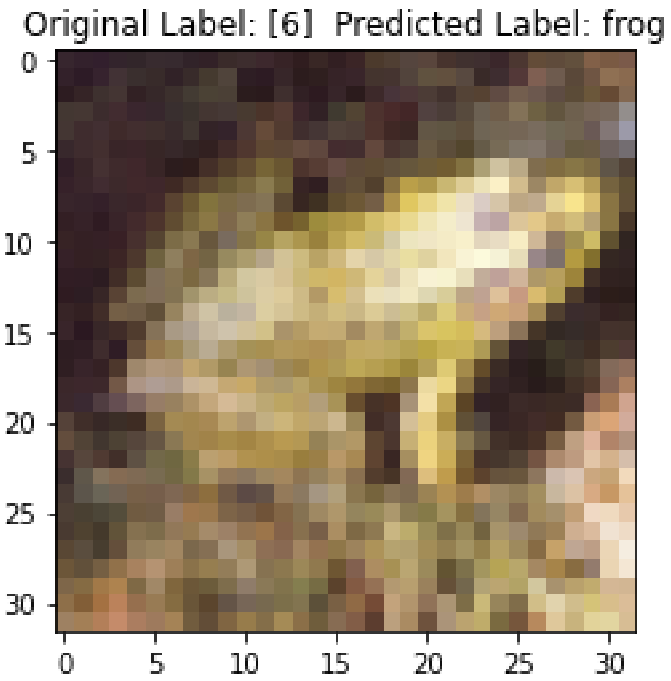

Original Label: [9]  Predicted Label: truck
0
5
10
15
20
25
30
0
5
10
15
20
25
30


Original Label: [8]  Predicted Label: deer
0
5
10
15
20
25
30
0
5
10
15
20
25
30


Original Label: [5]  Predicted Label: dog
0
5
10
15
20
25
30
0
5
10
15
20
25
30


Original Label: [7]  Predicted Label: horse
0
5
10
15
20
25
30
0
5
10
15
20
25
30

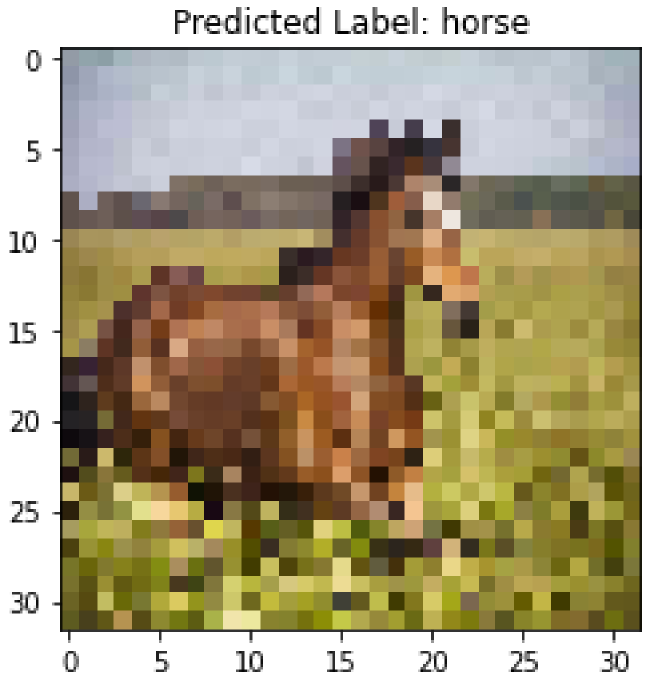

Original Label: [8]  Predicted Label: ship
0
5
10
15
20
25
30
0
5
10
15
20
25
30



Prediction: export the image and prediction

[15]:

```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

timg = "/home/jayanthikishore/Downloads/ML_classwork/Week5/horse.jpg"
# timg = "/Users/preethamvignesh/Downloads/deer.jpg"
# timg = "/Users/preethamvignesh/Downloads/dog.jpg"
img = keras.preprocessing.image.load_img(timg, target_size=(32, 32))

plt.imshow(img, cmap=plt.cm.binary)
#change the dimension to train and test data (32x32)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch
#Prediction
prediction = model.predict(img_array)
score = tf.nn.softmax(prediction[0])
# print(class_names[np.argmax(score)])
plt.title('Predicted Label: {0}'.format(class_names[np.argmax(score)]))
plt.show()
img_array.shape
```



```
[15]: TensorShape([1, 32, 32, 3])
```

```
[ ]:
```

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

```
[ ]:
```