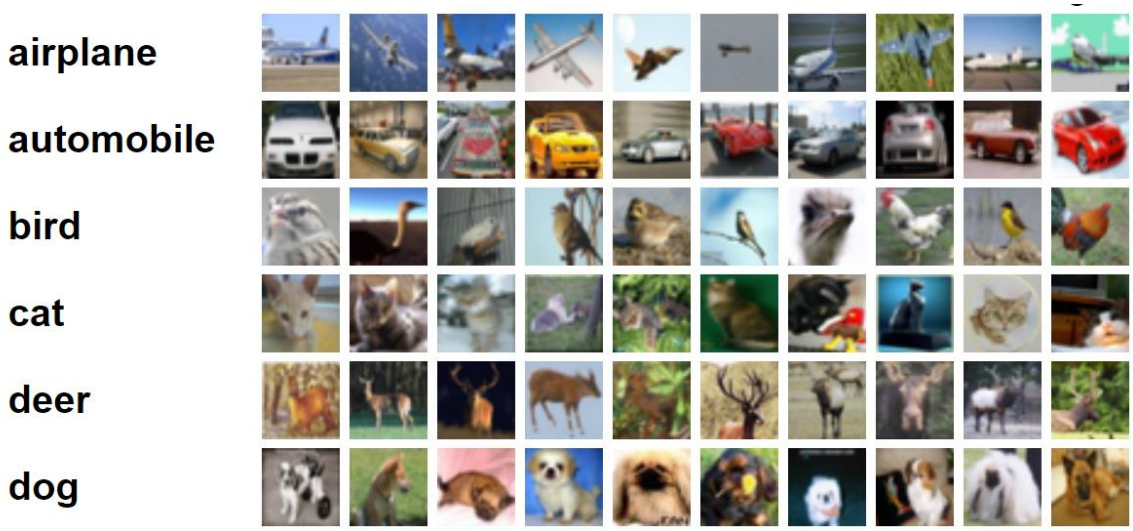
PREDICTING IMAGES OF COMMON OBJECTS USING CNN

WITH TENSORFLOW

DR. ALVIN ANG



CONTENTS

https://www.alvinang.sg/s/Predicting_Images_of_Common_Objects_using_CNN_by_Dr_Alvin_Ang.ipynb

https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

Predicting Images of Common Objects using CNN by Dr Alvin Ang

the code presented here uses colored images... which means it takes a long time for epochs processing to run... very very slow...

if you wish to convert these images to grayscale (so that you can train the model faster), u may follow the code here:

https://becominghuman.ai/cifar-10-image-classification-fd2ace47c5e8

https://becominghuman.ai/cifar-10-image-classification-fd2ace47c5e8

Ex: CNN

Create a CNN for the CIFAR-10 dataset with the following specification

- first conv layer with 16 3x3 filters, followed by 2x2 max pooling
- second conv layer with 32 3x3 filters, followed by 2x2 max pooling
- flatten the layers
- dense classification layer with 64 neurons
- softmax output layer

Time: 15 mins

CIFAR-10 Image Dataset

- The CIFAR-10 dataset (https://www.cs.toronto.edu/~kriz/cifar.html)
- 60000 32x32 colour images
- 10 classes, with 6000 images per class.
- 50000 training images and 10000 test images

- airplane : 0
- automobile : 1
- bird : 2
- cat : 3
- deer : 4
- dog : 5
- frog : 6
- horse : 7
- ship : 8
- truck : 9

I. STEP 1: LOAD DATA

A. IMPORT ALL LIBRARIES

Step 1: Load Data

1a) Import All Libraries

```
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
```

B. IMPORT THE CIFAR-10 DATA + TRAIN TEST SPLIT

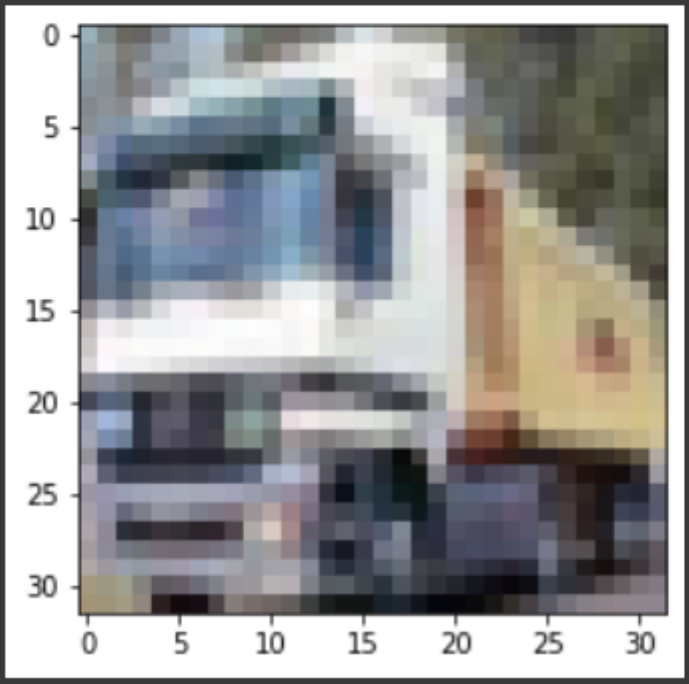
1b) Import the CIFAR-10 Data + Train Test Split

```
[ ] import tensorflow as tf
    from tensorflow import keras

    cifar10 = keras.datasets.cifar10
    (x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

1c) Visualize the Data

```
plt.imshow(x_train[1])
```

```
<matplotlib.image.AxesImage at 0x7f806db865d0>
```



```
x_train[1]
```

```
array([[[154, 177, 187],
        [126, 137, 136],
        [105, 104,  95],
        ...,
        [ 91,  95,  71],
        [ 87,  90,  71],
        [ 79,  81,  70]],

       [[140, 160, 169],
        [145, 153, 154],
        [125, 125, 118],
        ...,
        [ 96,  99,  78],
        [ 77,  80,  62],
        [ 71,  73,  61]],

       [[140, 155, 164],
        [139, 146, 149],
        [115, 115, 112],
```

```
x_train[1].shape

#we see that the image in x_train[1] is represented by 32 rows x 32 columns of data
#with 3 channels (RGB)
#each represents 0 (black) to 255 (white)
```

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

```
y_train[1]
# 9 is a truck
```

```
array([9], dtype=uint8)
```

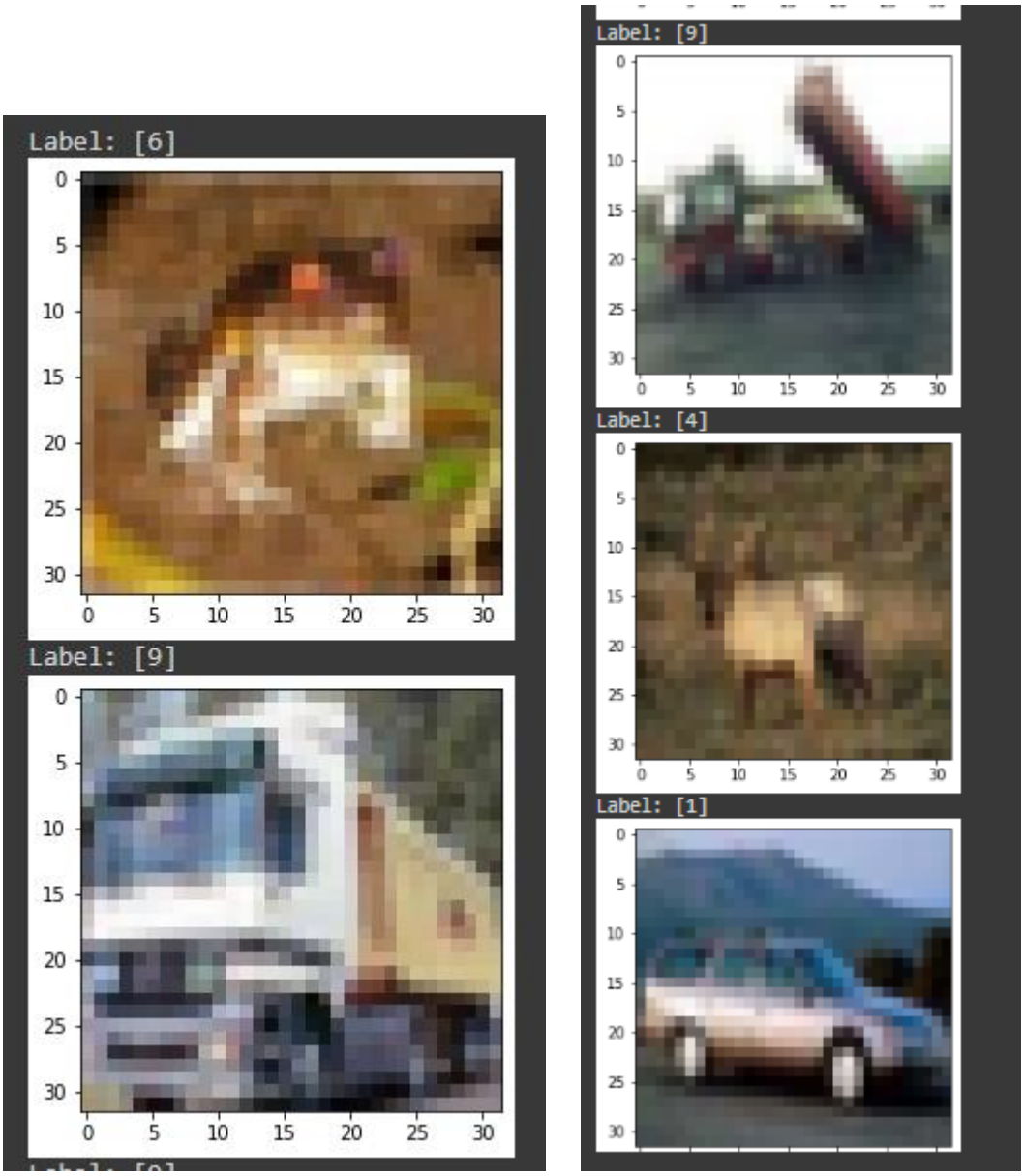
```
display (y_train)
```

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

```
print(len(np.unique(y_train)))
#to see how many classes there are ... 10 classes
```

```
10
```

```
for i in range(5):
  print(f'Label: {y_train[i]}')
  plt.imshow(x_train[i])
  plt.show()
```



D. NORMALIZE THE DATA

1d) Normalize the Data

```
[ ] x_train, x_test = x_train/255, x_test/255

    #pictures are from grayscale 0 to 255, so we divide by 255 to
    #normalize from 0 to 1 probability

    #0 is pure black
    #255 (or 1) is pure white
```

E. RESHAPE THE DATA

1e) Reshape the Data

```
[ ] print(f'Before: {x_train.shape}')
    print(f'Before: {x_test.shape}')

    #this means there are 50k images inside x_train which are 32x32 pixels
    #and there are 10k images inside x_test
```

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

```
x_train = x_train[..., tf.newaxis]
x_test = x_test[..., tf.newaxis]
print(f'After: {x_train.shape}')
print(f'After: {x_test.shape}')

#this means that we convert the x_train and x_test to include another parameter (or dimension)
#purpose is to 'fit' in the CNN layer later

#NOTE: EACH TIME YOU RUN THIS CODE IT WILL ADD ANOTHER DIMESION (so don't run it too many times!)
#e.g. 1st time (..., 32, 32, 3, 1)
#e.g. 2nd time (..., 32, 32, 3, 1, 1)... and so forth
```

```
After: (50000, 32, 32, 3, 1)
After: (10000, 32, 32, 3, 1)
```

```
#alternatively, to RESHAPE the data... you can try
#x_train = x_train.reshape((50000, 32, 32, 3))
#x_test = x_test.reshape((10000, 32, 32, 3))
```

Step 2: Build the Model

```
[ ] from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

    model = Sequential([

    #Convolution Layer
        Conv2D(filters = 16,
               kernel_size = (3, 3),
               activation = 'relu',
               input_shape = (32,32,3)),

    #Pooling Layer
        MaxPooling2D((2,2)),

    #Convolution Layer
        Conv2D(32, (3, 3), activation = 'relu'),

    #Pooling Layer
        MaxPooling2D((2, 2)),

    #ANN Layers
        Flatten(),
        Dense(64, activation = 'relu'),
        Dense(64, activation = 'relu'),
        Dense(10, activation = 'softmax')
    ])
```

A. MODEL SUMMARY

2a) Model Summary

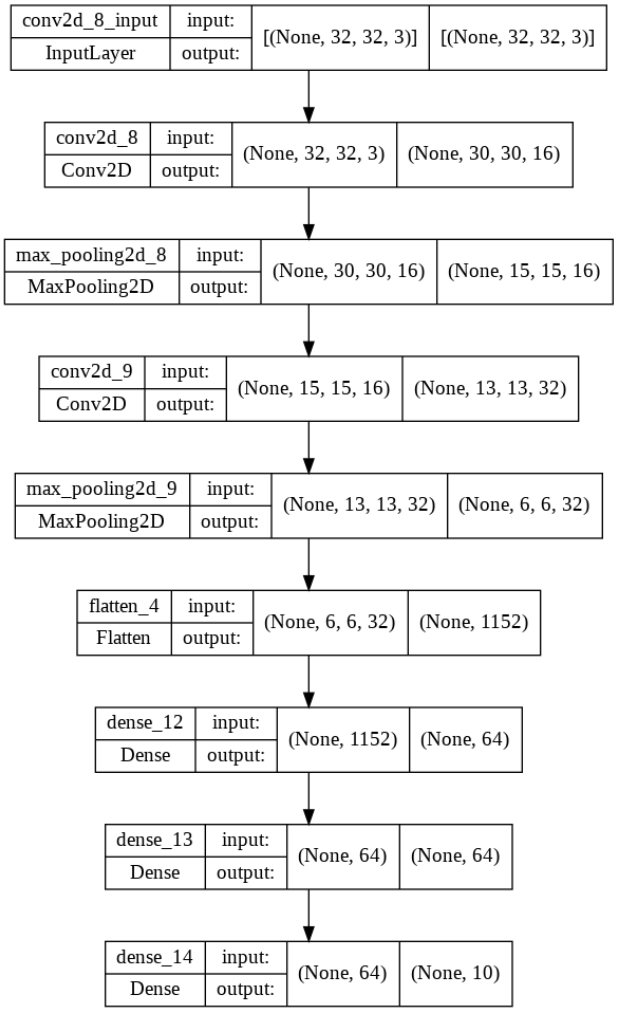
```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 30, 30, 16)        448

 max_pooling2d_8 (MaxPooling  (None, 15, 15, 16)       0
 2D)

 conv2d_9 (Conv2D)           (None, 13, 13, 32)        4640

 max_pooling2d_9 (MaxPooling  (None, 6, 6, 32)         0
 2D)

 flatten_4 (Flatten)         (None, 1152)              0

 dense_12 (Dense)            (None, 64)                73792

 dense_13 (Dense)            (None, 64)                4160

 dense_14 (Dense)            (None, 10)                650

=================================================================
Total params: 83,690
Trainable params: 83,690
Non-trainable params: 0
_________________________________________________________________
```

B. MODEL VISUALIZATION

2b) Model Visualization

```
import pydot
keras.utils.plot_model(model, 'model.png', show_shapes=True)
```



Step 3: Compile the Model

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

    #sparse_categorical_crossentropy is slower to train
    #but easier to code
    #try converting to categorical_crossentropy for faster

    #sparse_categorical_crossentropy dun need "One-Hot Encoding" for
    #both inputs and outputs

    #adam is for cross entropy --> Classification
    #RMSprop is for Regression
```

Step 4: Train the Model

```
[ ] history = model.fit(
        x_train, y_train,
        epochs = 4,
        validation_data = (x_test, y_test)
    )
```

```
Epoch 1/4
1563/1563 [==============================] - 43s 27ms/step - loss: 1.0616 - accuracy: 0.6260 - val_loss: 1.0281 - val_accuracy: 0.6404
Epoch 2/4
1563/1563 [==============================] - 39s 25ms/step - loss: 0.9753 - accuracy: 0.6577 - val_loss: 1.0110 - val_accuracy: 0.6470
Epoch 3/4
1563/1563 [==============================] - 37s 24ms/step - loss: 0.9100 - accuracy: 0.6800 - val_loss: 1.0031 - val_accuracy: 0.6543
Epoch 4/4
1563/1563 [==============================] - 37s 23ms/step - loss: 0.8565 - accuracy: 0.7000 - val_loss: 0.9757 - val_accuracy: 0.6658
```

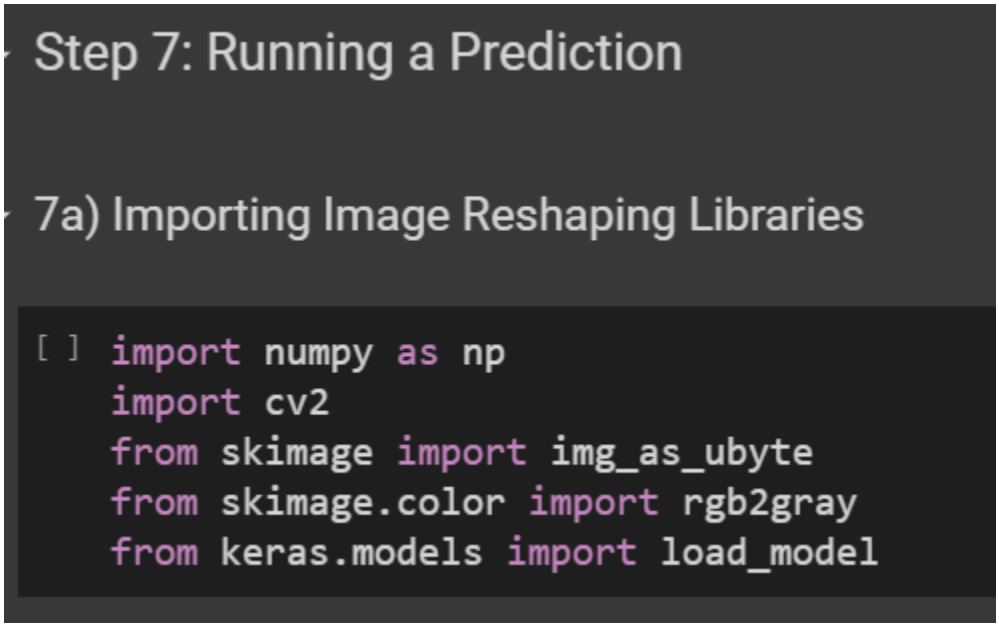
Step 5: Save the Model

```
[ ] model.save('classifying_common_objects.h5')

    #h5 means HDF5 format
    #it saves the model configuration, weights, biases and all hyperparameters
```

Step 6: Load the Model

```
[ ] # model = keras.models.load_model('classifying_common_objects.h5')

    #though not needed here, we will load this model for future use
    #so that we don't have to run the 100 epochs as it takes very long
```

A. IMPORTING IMAGE RESHAPING LIBRARIES

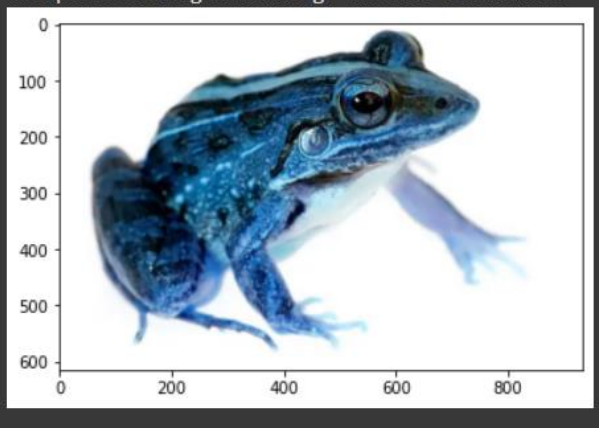


B. IMPORTING AND PREVIEWING THE FROG IMAGE

7b) Importing and Previewing the Frog Image

```
[33] img_1 = cv2.imread("/content/frog.jpg", 1)

     # 0 means grayscale, 1 means color

[34] import matplotlib.pyplot as plt

     plt.imshow(img_1)
```

```
<matplotlib.image.AxesImage at 0x7f479151c390>
```

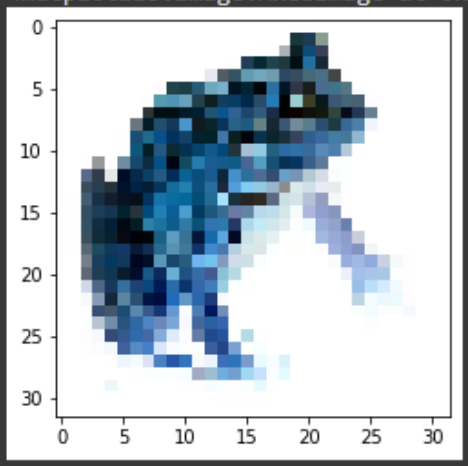


7c) Re-Coloring / Re-Shaping / Re-Sizing

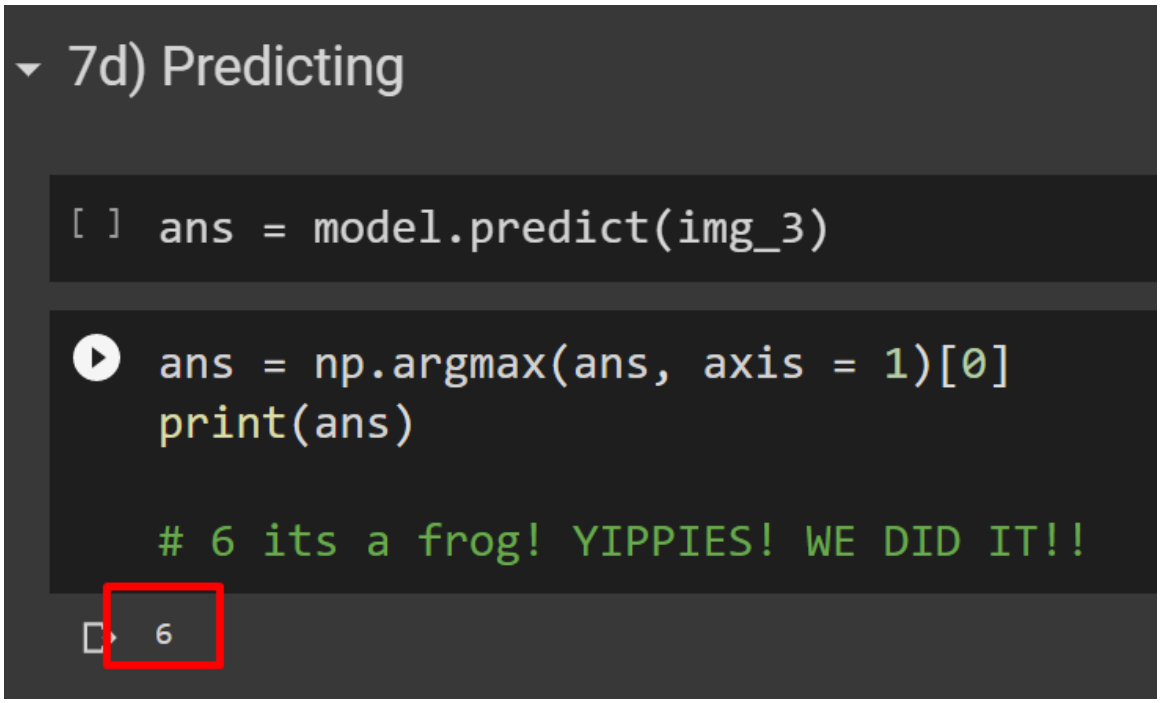
```
[35] img_2 = cv2.resize(img_1, (32, 32))
     #gives a shape of (32, 32, 3)
```

```
[36] plt.imshow(img_2)
```

```
<matplotlib.image.AxesImage at 0x7f4791480a50>
```



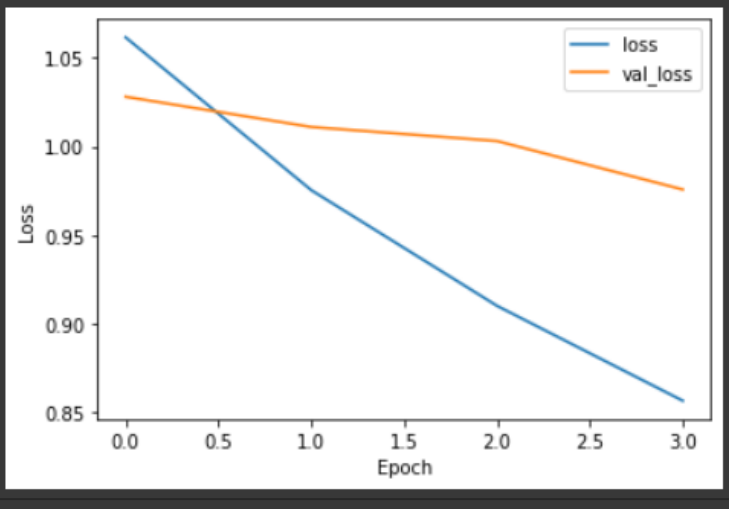

```
img_3 = np.expand_dims(img_2, 0)
#gives a shape of (1, 32, 32, 3),
#0 means first dim, we add a dimension in front
```



Step 8: Visualize the Loss or Error

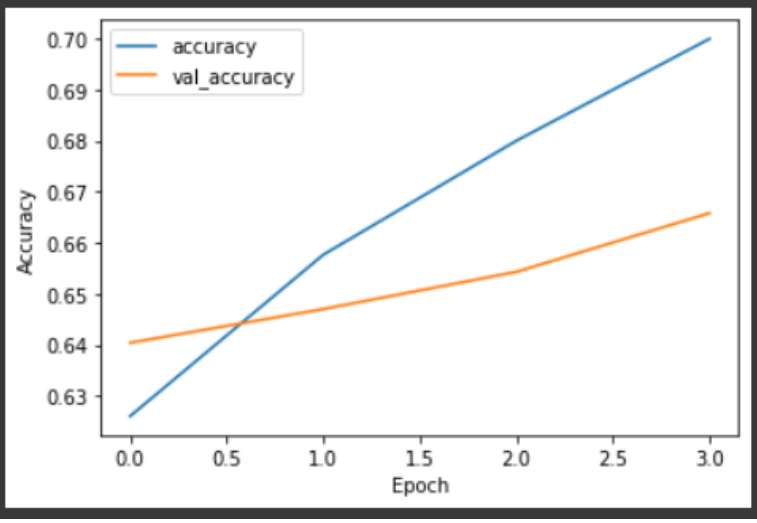
```
[ ] loss = history.history['loss']
    val_loss = history.history['val_loss']
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    epoch = range(len(loss))
```

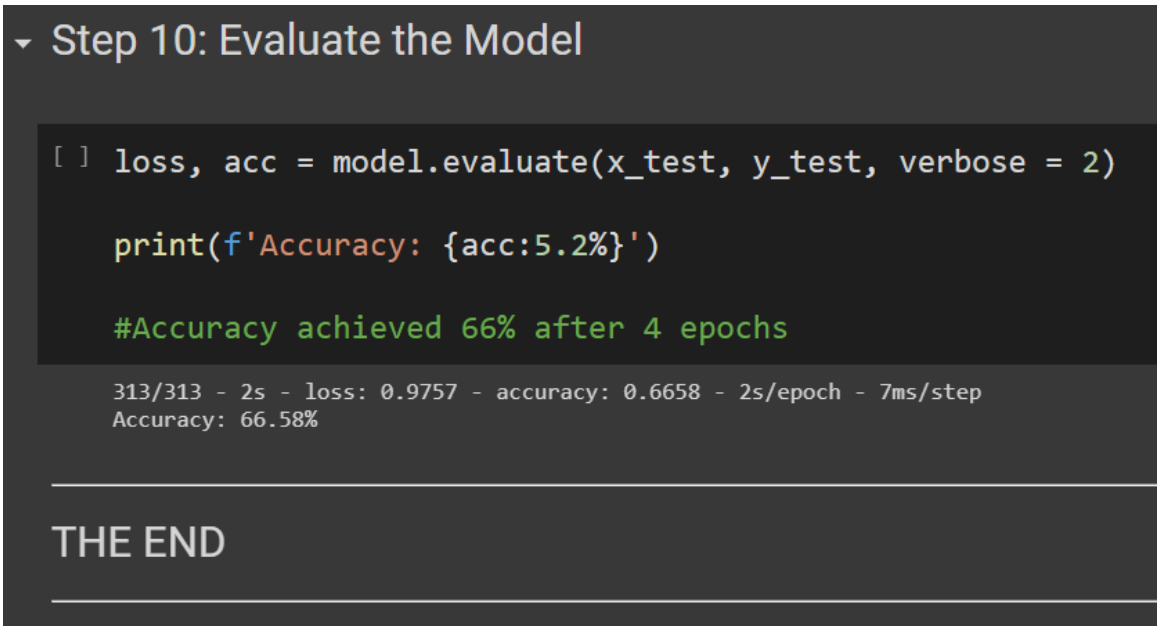
```
import matplotlib.pyplot as plt

plt.plot(epoch, loss, label = "loss")
plt.plot(epoch, val_loss, label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Step 9: Visualize the Accuracy

```
[ ] plt.plot(epoch, acc, label = 'accuracy')
    plt.plot(epoch, val_acc, label = 'val_accuracy')
    plt.xlabel("Epoch")
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
```





ABOUT DR. ALVIN ANG

Dr. Alvin Ang earned his Ph.D., Masters and Bachelor degrees from NTU, Singapore. He is a scientist, entrepreneur, as well as a personal/business advisor. More about him at www.AlvinAng.sg.