3. Build the Image classification model by dividing the model into following 4 stages:

a. Loading and preprocessing the image data

b. Defining the model's architecture

c. Training the model

d. Estimating the model's performance

```
#Import TensorFlow
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

```
#Download and prepare the CIFAR10 dataset
(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
```

```
#Verify the data for first five images in train dataset
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # 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 = 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'))
```

```
#Let's display the architecture of your model so far:
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 30, 30, 32)        896

 max_pooling2d_5 (MaxPoolin  (None, 15, 15, 32)        0
 g2D)

 conv2d_10 (Conv2D)          (None, 13, 13, 64)        18496

 max_pooling2d_6 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_11 (Conv2D)          (None, 4, 4, 64)          36928

=================================================================
Total params: 56320 (220.00 KB)
Trainable params: 56320 (220.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Above, you can see that the output of every Conv2D and MaxPooling2D layer is a 3D tensor of shape (height, width, channels). The width and height dimensions tend to shrink as you go deeper in the network. The number of output channels for each Conv2D layer is controlled by the first argument (e.g., 32 or 64). Typically, as the width and height shrink, you can afford (computationally) to add more output channels in each Conv2D layer.

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_9 (Conv2D)           (None, 30, 30, 32)        896       
                                                                 
 max_pooling2d_5 (MaxPoolin  (None, 15, 15, 32)        0         
 g2D)                                                            
                                                                 
 conv2d_10 (Conv2D)          (None, 13, 13, 64)        18496     
                                                                 
 max_pooling2d_6 (MaxPoolin  (None, 6, 6, 64)          0         
 g2D)                                                            
                                                                 
 conv2d_11 (Conv2D)          (None, 4, 4, 64)          36928     
                                                                 
 flatten (Flatten)           (None, 1024)              0         
                                                                 
 dense (Dense)               (None, 64)                65600     
                                                                 
 dense_1 (Dense)             (None, 10)                650       
                                                                 
=================================================================
Total params: 122570 (478.79 KB)
Trainable params: 122570 (478.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

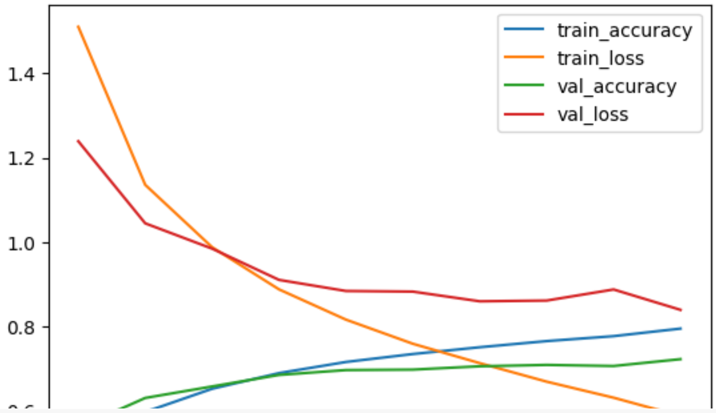
```
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              optimizer='adam',
              metrics=['accuracy'])

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

```
Epoch 1/10
1563/1563 [==============================] - 73s 46ms/step - loss: 1.5109 - accuracy: 0.4464 - val_loss: 1.2392 - val_accuracy: 0.5
Epoch 2/10
1563/1563 [==============================] - 72s 46ms/step - loss: 1.1357 - accuracy: 0.5963 - val_loss: 1.0443 - val_accuracy: 0.6
Epoch 3/10
1563/1563 [==============================] - 75s 48ms/step - loss: 0.9875 - accuracy: 0.6514 - val_loss: 0.9842 - val_accuracy: 0.6
Epoch 4/10
1563/1563 [==============================] - 76s 49ms/step - loss: 0.8874 - accuracy: 0.6886 - val_loss: 0.9100 - val_accuracy: 0.6
Epoch 5/10
1563/1563 [==============================] - 76s 48ms/step - loss: 0.8156 - accuracy: 0.7150 - val_loss: 0.8836 - val_accuracy: 0.6
Epoch 6/10
1563/1563 [==============================] - 73s 47ms/step - loss: 0.7583 - accuracy: 0.7339 - val_loss: 0.8821 - val_accuracy: 0.6
Epoch 7/10
1563/1563 [==============================] - 70s 45ms/step - loss: 0.7122 - accuracy: 0.7501 - val_loss: 0.8591 - val_accuracy: 0.7
Epoch 8/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.6683 - accuracy: 0.7646 - val_loss: 0.8609 - val_accuracy: 0.7
Epoch 9/10
1563/1563 [==============================] - 72s 46ms/step - loss: 0.6302 - accuracy: 0.7764 - val_loss: 0.8872 - val_accuracy: 0.7
Epoch 10/10
 591/1563 [==========>...................] - ETA: 42s - loss: 0.5661 - accuracy: 0.8025
```

```
plt.plot(np.arange(0,10), history.history['accuracy'], label='train_accuracy')
plt.plot(np.arange(0,10), history.history['loss'], label='train_loss')
plt.plot(np.arange(0,10), history.history['val_accuracy'], label='val_accuracy')
plt.plot(np.arange(0,10), history.history['val_loss'], label='val_loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7c7da4592500>
```




```
predictions = model.predict(test_images)
from sklearn.metrics import classification_report
```

```
313/313 [==============================] - 4s 12ms/step
```

```
print(classification_report(test_labels.argmax(axis=1),predictions.argmax(axis=1)))
```

```
              precision    recall  f1-score   support

           0       1.00      0.11      0.19     10000
           1       0.00      0.00      0.00         0
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         0
           5       0.00      0.00      0.00         0
           6       0.00      0.00      0.00         0
           7       0.00      0.00      0.00         0
           8       0.00      0.00      0.00         0
           9       0.00      0.00      0.00         0

    accuracy                           0.11     10000
   macro avg       0.10      0.01      0.02     10000
weighted avg       1.00      0.11      0.19     10000

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill
  _warn_prf(average, modifier, msg_start, len(result))
```