```
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 as tf

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

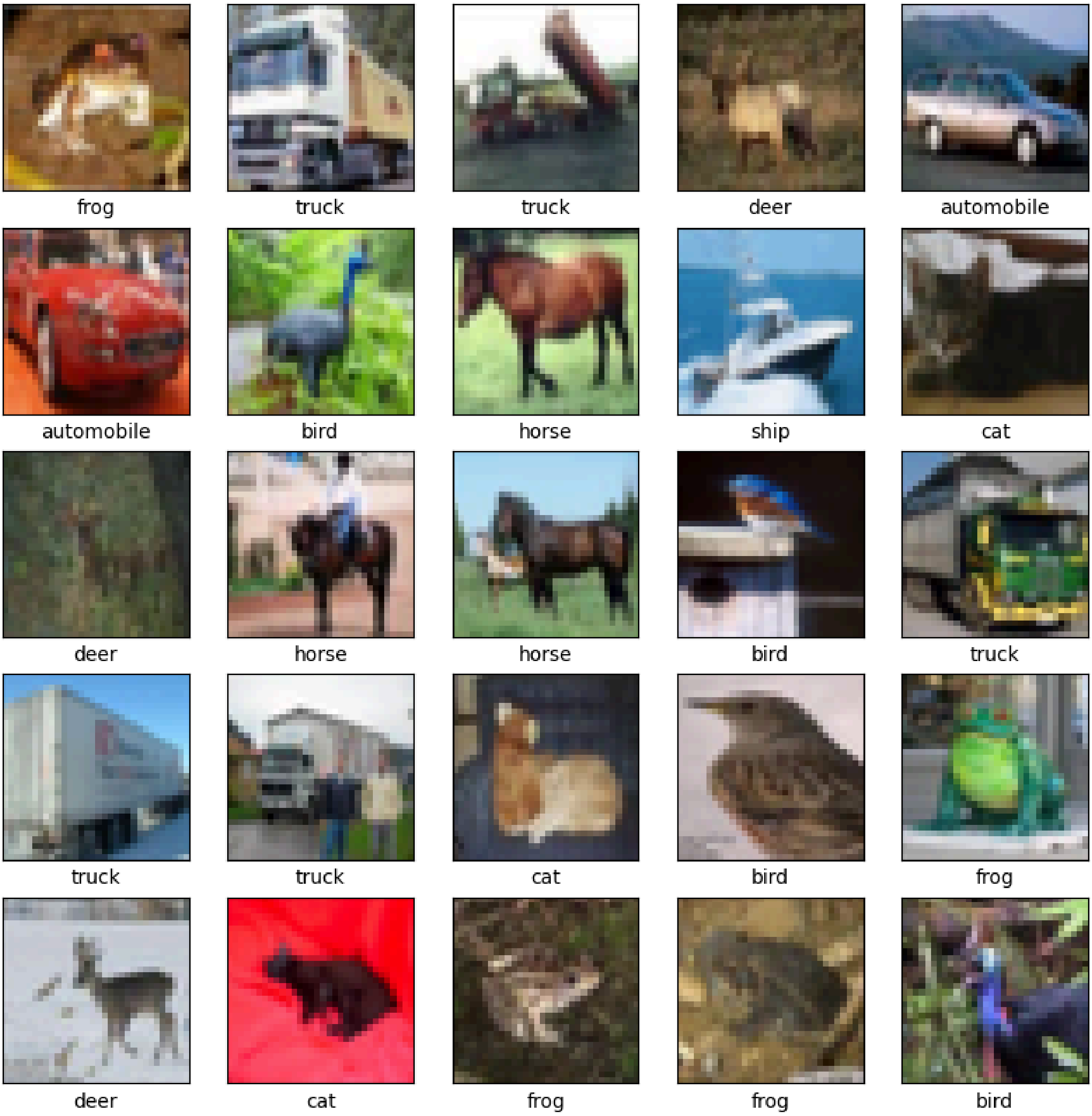
```
#a. Loading and preprocessing the image data
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

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

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

```
#b. Defining the model’s architecture
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    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'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
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)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36,928

Total params: 56,320 (220.00 KB)

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

```
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)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36,928
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65,600
dense_1 (Dense)	(None, 10)	650

Total params: 122,570 (478.79 KB)

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

history = model.fit(train_images, train_labels, epochs=20,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 69s 43ms/step - accuracy: 0.9029 - loss: 0.2706 - val_accuracy: 0.6953 - val_
Epoch 2/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 43ms/step - accuracy: 0.9173 - loss: 0.2282 - val_accuracy: 0.6936 - val_
Epoch 3/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 79s 41ms/step - accuracy: 0.9196 - loss: 0.2246 - val_accuracy: 0.7014 - val_
Epoch 4/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 80s 40ms/step - accuracy: 0.9278 - loss: 0.2029 - val_accuracy: 0.6920 - val_
Epoch 5/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 62s 40ms/step - accuracy: 0.9285 - loss: 0.2020 - val_accuracy: 0.6900 - val_
Epoch 6/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 86s 42ms/step - accuracy: 0.9337 - loss: 0.1846 - val_accuracy: 0.6872 - val_
Epoch 7/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 42ms/step - accuracy: 0.9383 - loss: 0.1749 - val_accuracy: 0.6854 - val_
Epoch 8/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 41ms/step - accuracy: 0.9417 - loss: 0.1620 - val_accuracy: 0.6883 - val_
```

```
Epoch 9/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 64s 41ms/step - accuracy: 0.9420 - loss: 0.1570 - val_accuracy: 0.6836 - val_
Epoch 10/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 66s 42ms/step - accuracy: 0.9436 - loss: 0.1538 - val_accuracy: 0.6810 - val_
Epoch 11/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 42ms/step - accuracy: 0.9507 - loss: 0.1394 - val_accuracy: 0.6918 - val_
Epoch 12/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 65s 42ms/step - accuracy: 0.9498 - loss: 0.1406 - val_accuracy: 0.6795 - val_
Epoch 13/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 41ms/step - accuracy: 0.9526 - loss: 0.1345 - val_accuracy: 0.6798 - val_
Epoch 14/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 41ms/step - accuracy: 0.9531 - loss: 0.1346 - val_accuracy: 0.6839 - val_
Epoch 15/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 80s 39ms/step - accuracy: 0.9538 - loss: 0.1340 - val_accuracy: 0.6809 - val_
Epoch 16/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 85s 41ms/step - accuracy: 0.9543 - loss: 0.1311 - val_accuracy: 0.6799 - val_
Epoch 17/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 64s 41ms/step - accuracy: 0.9565 - loss: 0.1221 - val_accuracy: 0.6819 - val_
Epoch 18/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 64s 41ms/step - accuracy: 0.9597 - loss: 0.1164 - val_accuracy: 0.6791 - val_
Epoch 19/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 80s 40ms/step - accuracy: 0.9576 - loss: 0.1216 - val_accuracy: 0.6793 - val_
Epoch 20/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 41ms/step - accuracy: 0.9597 - loss: 0.1147 - val_accuracy: 0.6661 - val_
```

```
#d. Estimating the model’s performance
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 - 7s - 21ms/step - accuracy: 0.7062 - loss: 1.2744
```