Day 63

DIY

Q1. Problem Statement: Convolutional Neural Network

The *CIFAR10* dataset contains 60,000 color images in 10 classes, with 6,000 images in each class. The dataset is divided into 50,000 training images and 10,000 testing images. The classes are mutually exclusive, and there is no overlap between them.

Import the *'CIFAR10'* dataset using the `tensorFlow.keras.datasets()` command and perform the following tasks:

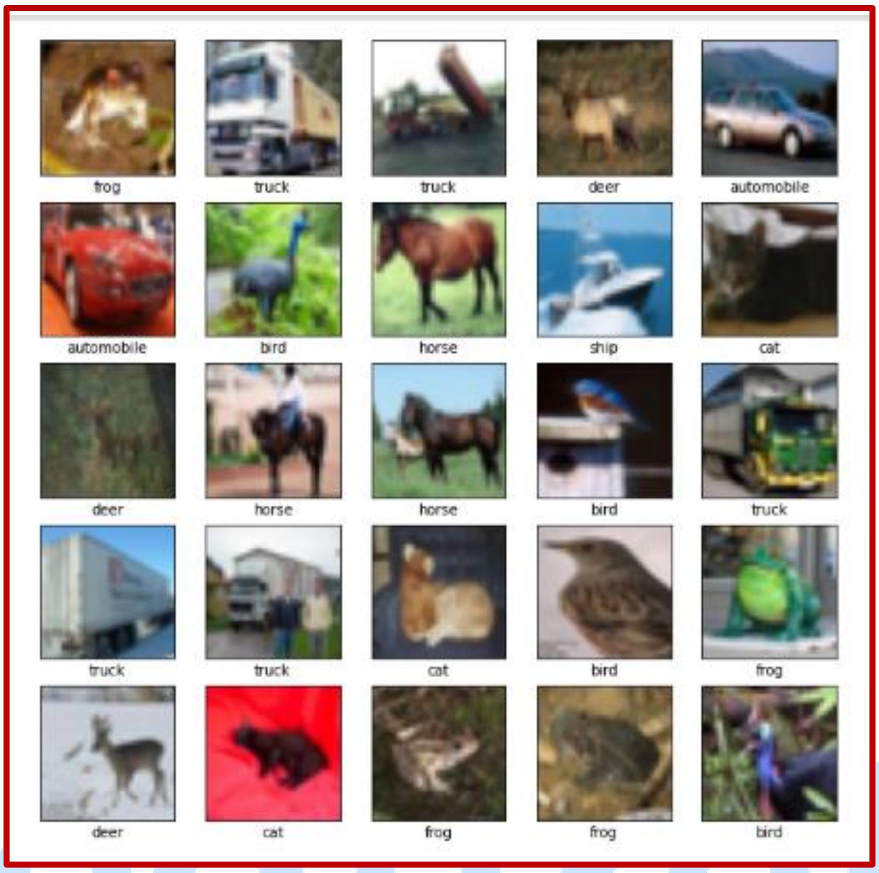
1. Download and import the Tensorflow and other required libraries
2. Import the '*CIFAR10*' test and train datasets using the Keras and the train and test labels. Also, scale the datasets by dividing with a common number
3. Define the class names as - `['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']` for class of images
4. Define a sequential convolutional base using common patterns Conv2D and MaxPooling2D layers (use `'relu'` (Rectified Linear Unit) activation function)

   Note: As input, a CNN takes tensors of shape (image_height, image_width, color_channels), ignoring the batch size. If you are new to these dimensions, color_channels refers to (R, G, B). In this example, you will configure your CNN to process inputs of shape (32, 32, 3), which is the format of CIFAR images. We can do this by passing the argument input_shape to your first layer.

5. Feed the final output tensor from the convolutional base into one or more Dense layers to perform classification.

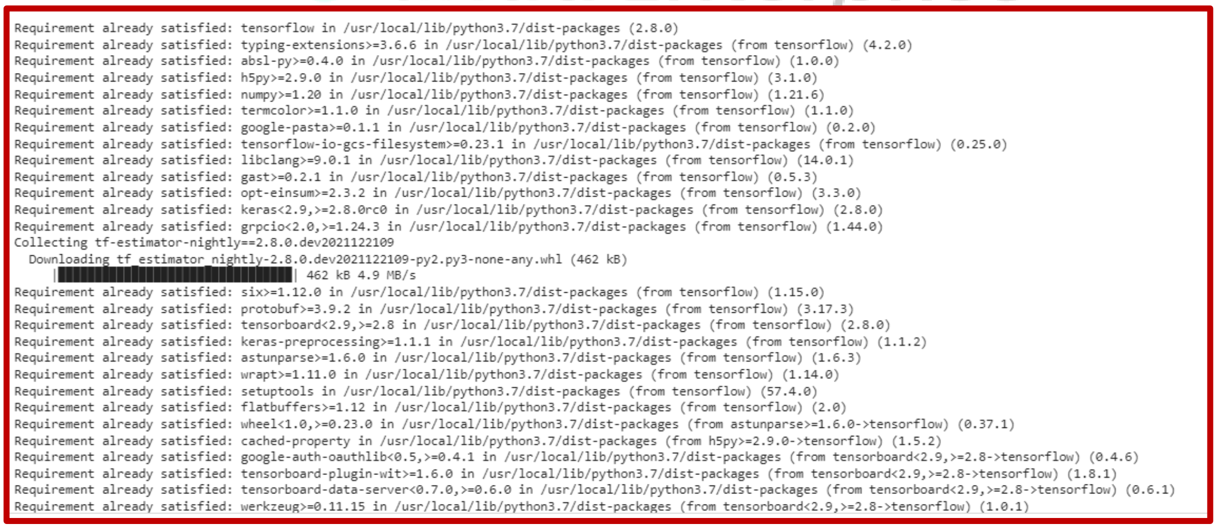
   Note: Dense layers take vectors as input (1D), while the current output is a 3D tensor. First, We will flatten (or unroll) the 3D output to 1D, then add one or more Dense layers on top. CIFAR has ten output classes, so we use a Dense final layer with ten outputs.
6. Compile and train the model
7. Calculate and print the test accuracy of the model

Dataset:



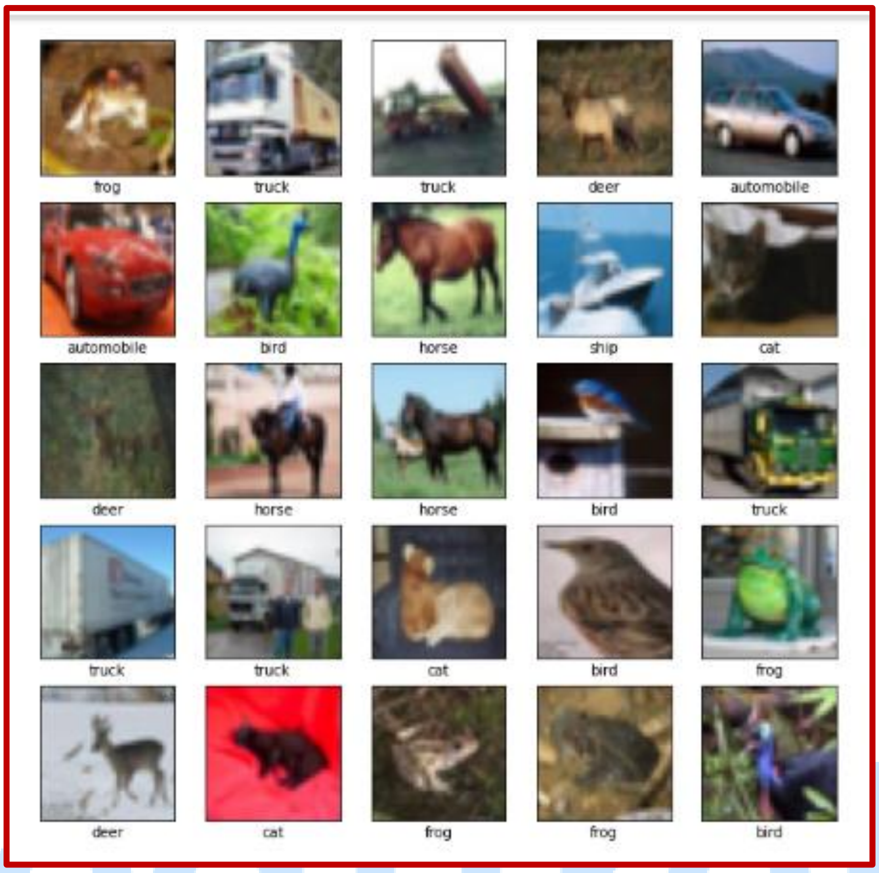
Sample Output:

1. Download and import the Tensorflow and other required libraries

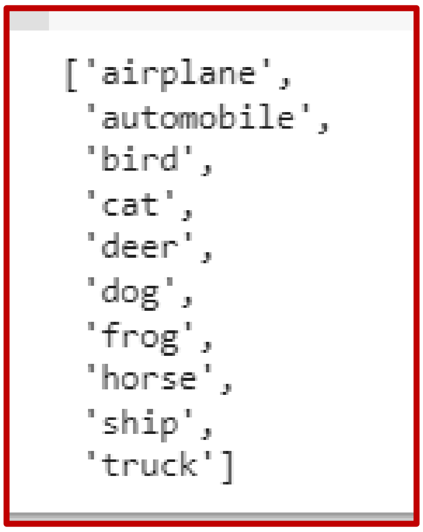


```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.7/dist-packages (2.8.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (4.2.0)
Requirement already satisfied: absl-py>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.0.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.21.6)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.1.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.25.0)
Requirement already satisfied: libclang>=9.0.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (14.0.1)
Requirement already satisfied: gast>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.5.3)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: keras<2.9,>=2.8.0rc0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.8.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.44.0)
Collecting tf-estimator-nightly==2.8.0.dev2021122109
  Downloading tf_estimator_nightly-2.8.0.dev2021122109-py2.py3-none-any.whl (462 kB)
     |████████████████████████████████| 462 kB 4.9 MB/s
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.15.0)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.17.3)
Requirement already satisfied: tensorboard<2.9,>=2.8 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.8.0)
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.1.2)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.14.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from tensorflow) (57.4.0)
Requirement already satisfied: flatbuffers>=1.12 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.7/dist-packages (from astunparse>=1.6.0->tensorflow) (0.37.1)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py>=2.9.0->tensorflow) (1.5.2)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow) (0.4.6)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow) (1.8.1)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow) (0.6.1)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow) (1.0.1)
```

2. Import the '*CIFAR10'* test and train datasets using the Keras and the train and test labels. Also, scale the datasets by dividing with a common number



3. Define the class names as - `['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']` for class of images

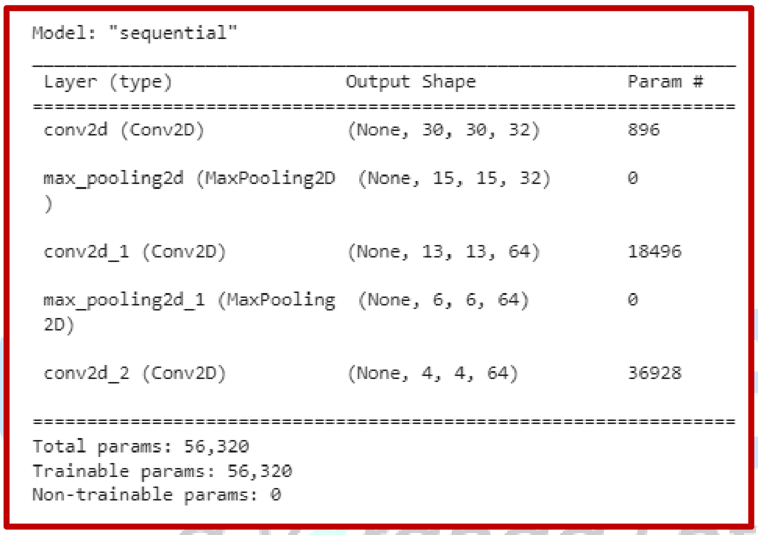


```
['airplane',
 'automobile',
 'bird',
 'cat',
 'deer',
 'dog',
 'frog',
 'horse',
 'ship',
 'truck']
```

4. Define a sequential convolutional base using common patterns Conv2D and MaxPooling2D layers (use '`relu`' (Rectified Linear Unit)

activation function)

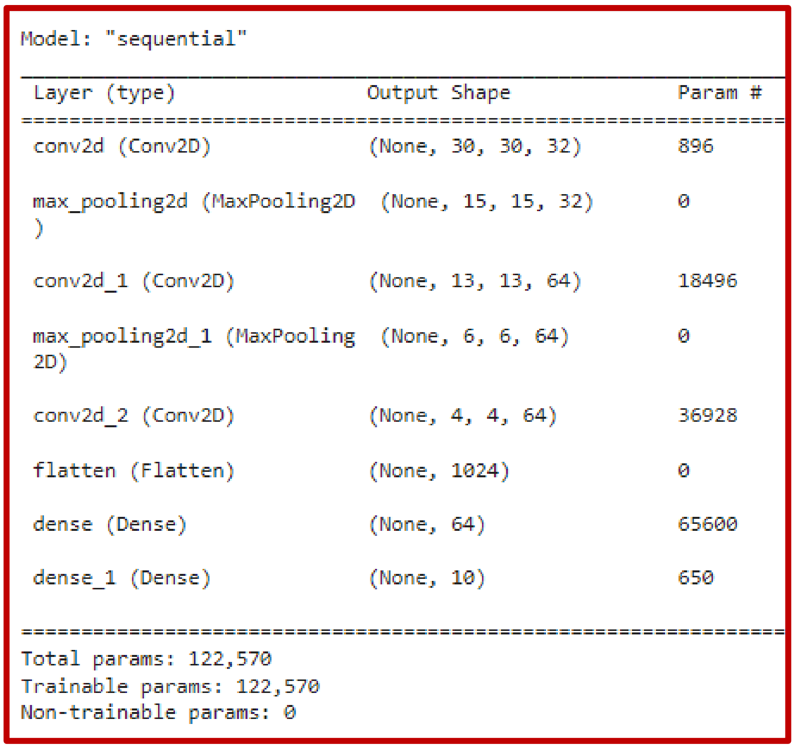
Note: As input, a CNN takes tensors of shape (image_height, image_width, color_channels), ignoring the batch size. If you are new to these dimensions, color_channels refers to (R, G, B). In this example, you will configure your CNN to process inputs of shape (32, 32, 3), which is the format of CIFAR images. We can do this by passing the argument input_shape to your first layer.



```
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 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
```

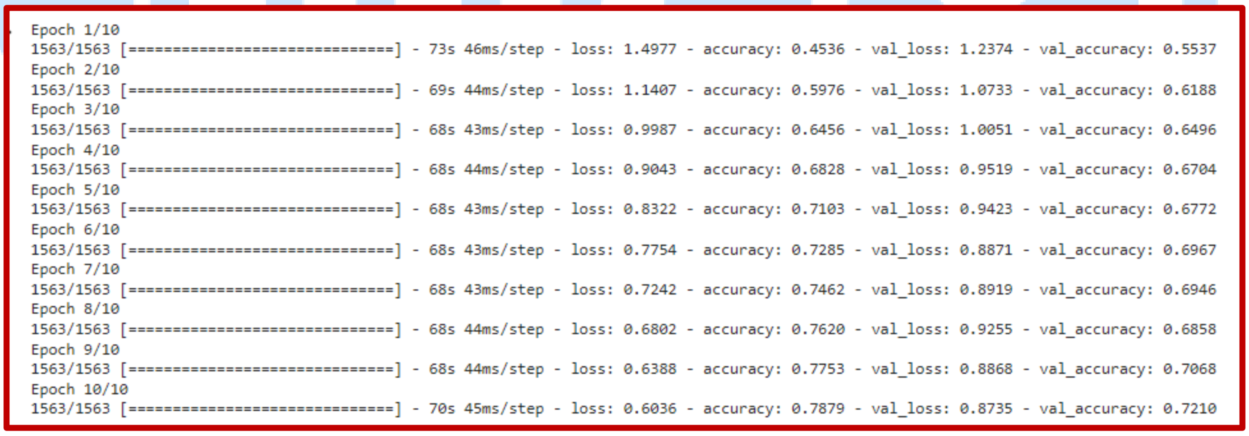
5. Feed the final output tensor from the convolutional base into one or more Dense layers to perform classification.

Note: Dense layers take vectors as input (1D), while the current output is a 3D tensor. First, We will flatten (or unroll) the 3D output to 1D, then add one or more Dense layers on top. CIFAR has ten output classes, so we use a Dense final layer with ten outputs.



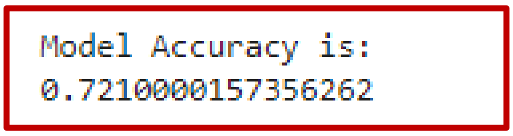
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 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
```

6. Compile and train the model



```
Epoch 1/10
1563/1563 [==============================] - 73s 46ms/step - loss: 1.4977 - accuracy: 0.4536 - val_loss: 1.2374 - val_accuracy: 0.5537
Epoch 2/10
1563/1563 [==============================] - 69s 44ms/step - loss: 1.1407 - accuracy: 0.5976 - val_loss: 1.0733 - val_accuracy: 0.6188
Epoch 3/10
1563/1563 [==============================] - 68s 43ms/step - loss: 0.9987 - accuracy: 0.6456 - val_loss: 1.0051 - val_accuracy: 0.6496
Epoch 4/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.9043 - accuracy: 0.6828 - val_loss: 0.9519 - val_accuracy: 0.6704
Epoch 5/10
1563/1563 [==============================] - 68s 43ms/step - loss: 0.8322 - accuracy: 0.7103 - val_loss: 0.9423 - val_accuracy: 0.6772
Epoch 6/10
1563/1563 [==============================] - 68s 43ms/step - loss: 0.7754 - accuracy: 0.7285 - val_loss: 0.8871 - val_accuracy: 0.6967
Epoch 7/10
1563/1563 [==============================] - 68s 43ms/step - loss: 0.7242 - accuracy: 0.7462 - val_loss: 0.8919 - val_accuracy: 0.6946
Epoch 8/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.6802 - accuracy: 0.7620 - val_loss: 0.9255 - val_accuracy: 0.6858
Epoch 9/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.6388 - accuracy: 0.7753 - val_loss: 0.8868 - val_accuracy: 0.7068
Epoch 10/10
1563/1563 [==============================] - 70s 45ms/step - loss: 0.6036 - accuracy: 0.7879 - val_loss: 0.8735 - val_accuracy: 0.7210
```

7. Calculate and print the test accuracy of the model



```
Model Accuracy is:
0.7210000157356262
```