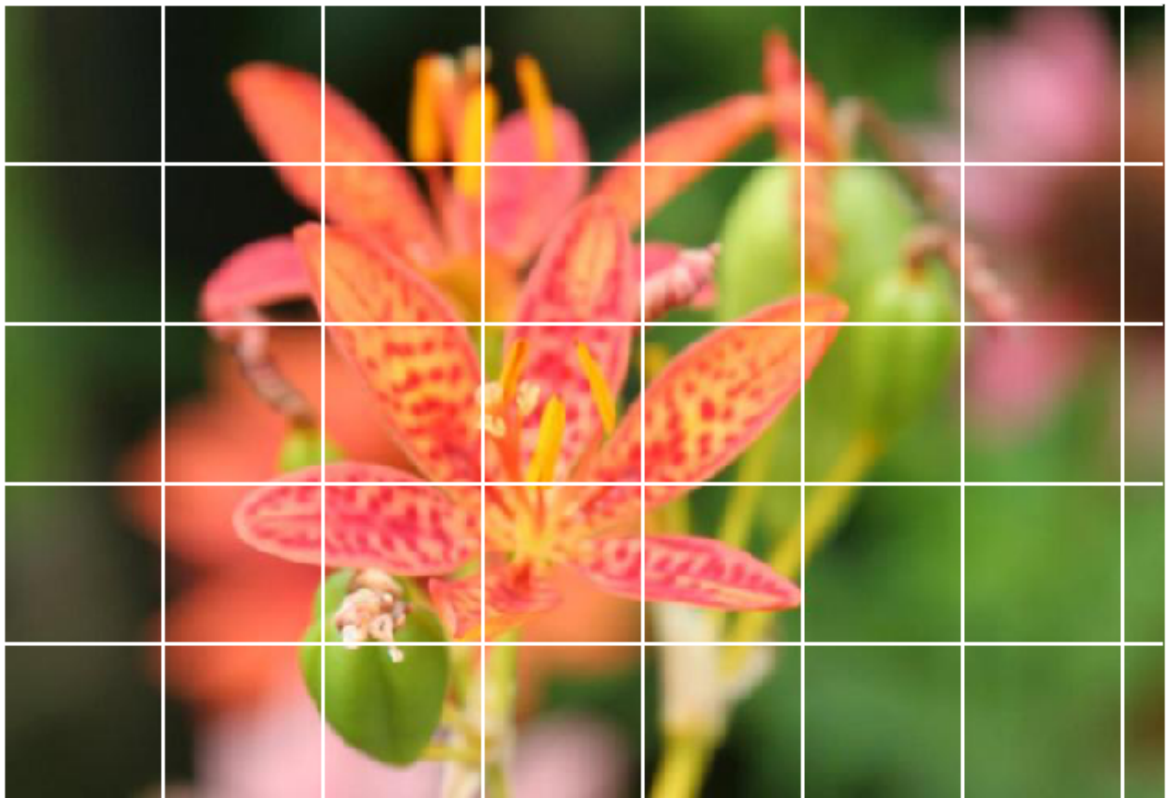


Build an Image Classifier to Classify Images of CIFAR-10 Data.

21.09.2020

Pintu Kumar Sah
Jorhat Engineering College
Roll-no:170710007040
Ph no. 8876490792
Email: shahpintu900@gmail.com

Overview

The model designed in the project correctly classifies the images of CIFAR-10 dataset with respect to its classes. There are a total of 10 classes namely Airplanes , Cars , Birds, Cats, Deer, Dogs, Frogs, Horses, Ships, Trucks.

The dataset consists of 60,000 32 x 32 color images, 6,000 images of each class.

Goals

1. To identify the image and classify it to its respective class.
2. To build a model which is precise and predict the classes of the images with a high accuracy and perfection.

Specifications

Here we are going to use Tensorflow and keras library of the python Language to build the model. We are using the CIFAR-10 dataset which 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.

Convolutional Neural Network(CNN)

Here we are Training a simple Convolutional Neural Network (CNN) to classify CIFAR images. This project uses the Keras Sequential API, Due to which creating and training our model will take just a few lines of code.

Milestones

I. Getting the CIFAR-10 DataSet

II. Building the model which predicts the image classes accurately

Import TensorFlow and various libraries

```
[1]  import tensorflow as tf

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

Download and prepare the CIFAR10 dataset

```
[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
```

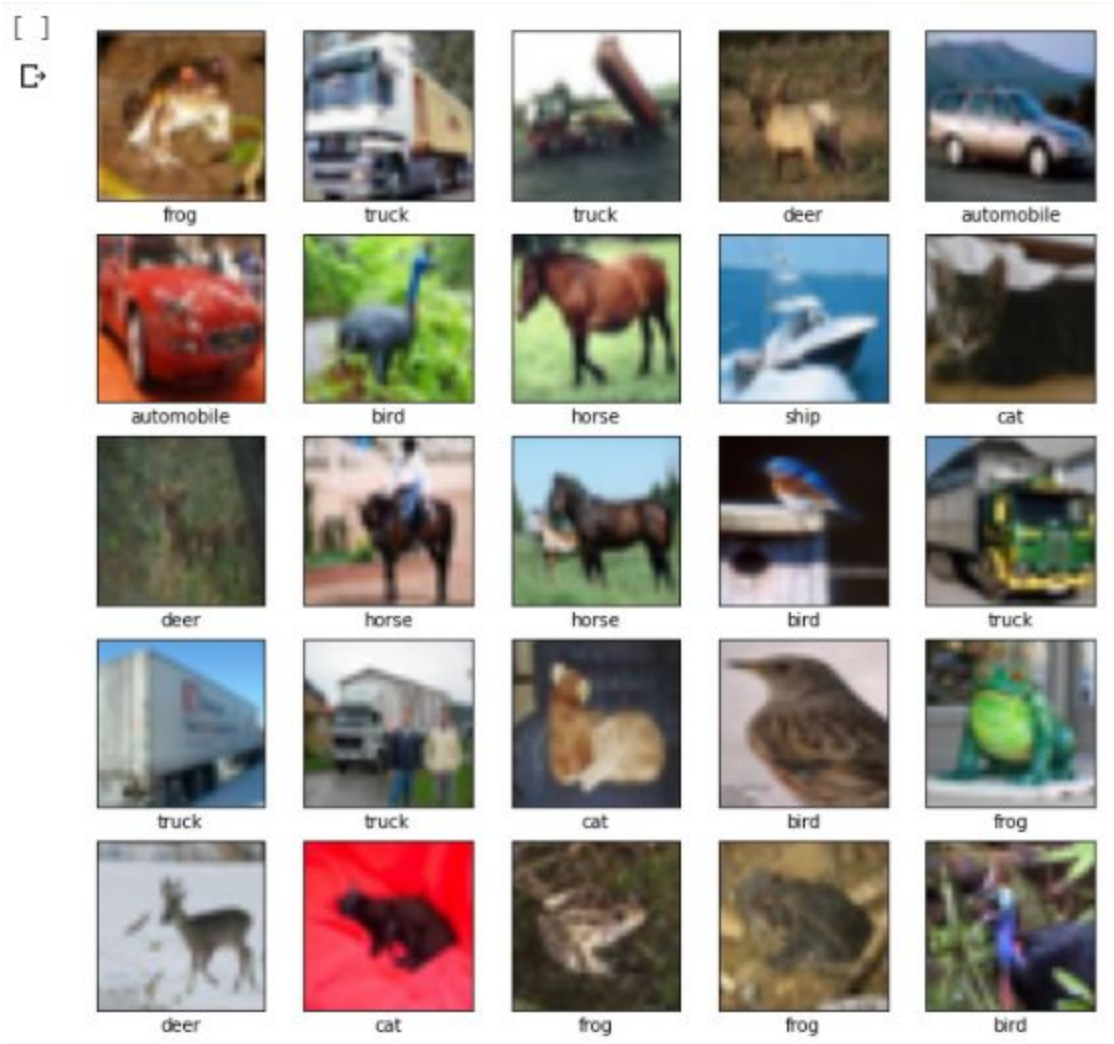
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
```

Verify the data

To verify that the dataset looks correct, let's plot the first 25 images from the training set and display the class name below each image.

```
[ ]  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], 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()
```



Create the convolutional base

The 6 lines of code below define the convolutional base using a common pattern: a stack of Conv2D and MaxPooling2D layers.

As input, a CNN takes tensors of shape (image_height, image_width, color_channels), ignoring the batch size. we will configure our 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 our first layer.

```
[ ]  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 our model so far.

```
[ ]  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
_________________________________________________________________
```

Above, we 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, we can afford (computationally) to add more output channels in each Conv2D layer.

Add Dense layers on top

To complete our model, we will feed the last output tensor from the convolutional base (of shape (4, 4, 64)) into one or more Dense layers to perform classification. Dense layers take vectors as input (which are 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 10 output classes, so we use a final Dense layer with 10 outputs and a softmax activation.

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

Here's the complete architecture of our model.

```
[ ]  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
```

Compile and train the model

```
[ ] 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))
```

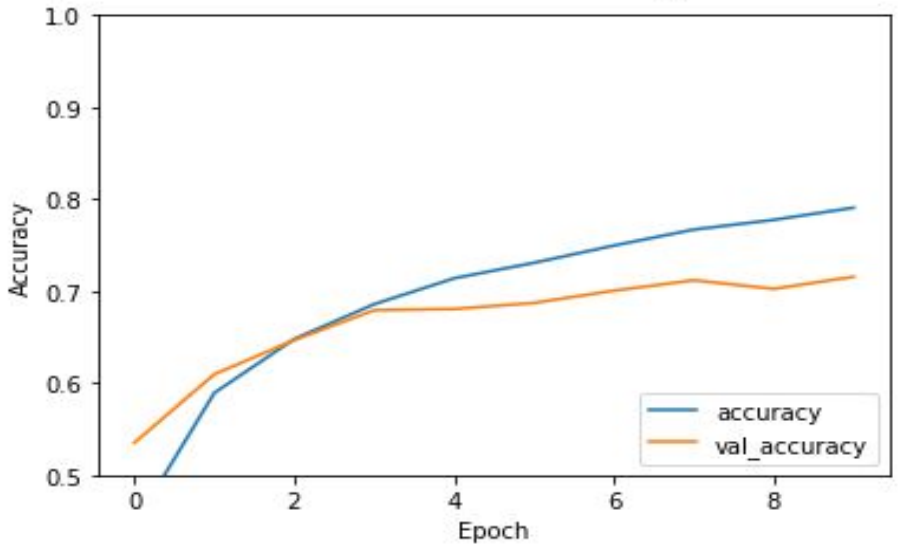
```
Epoch 1/10
1563/1563 [==============================] - 65s 41ms/step - loss: 1.5174 - accuracy: 0.4460 - val_loss: 1.3047 - val_accuracy: 0.5351
Epoch 2/10
1563/1563 [==============================] - 62s 40ms/step - loss: 1.1636 - accuracy: 0.5892 - val_loss: 1.0937 - val_accuracy: 0.6095
Epoch 3/10
1563/1563 [==============================] - 60s 39ms/step - loss: 0.9998 - accuracy: 0.6478 - val_loss: 1.0066 - val_accuracy: 0.6466
Epoch 4/10
1563/1563 [==============================] - 60s 38ms/step - loss: 0.8974 - accuracy: 0.6857 - val_loss: 0.9202 - val_accuracy: 0.6792
Epoch 5/10
1563/1563 [==============================] - 60s 38ms/step - loss: 0.8219 - accuracy: 0.7139 - val_loss: 0.9463 - val_accuracy: 0.6805
Epoch 6/10
1563/1563 [==============================] - 60s 38ms/step - loss: 0.7639 - accuracy: 0.7307 - val_loss: 0.9104 - val_accuracy: 0.6871
Epoch 7/10
1563/1563 [==============================] - 60s 38ms/step - loss: 0.7125 - accuracy: 0.7495 - val_loss: 0.8582 - val_accuracy: 0.7004
Epoch 8/10
1563/1563 [==============================] - 60s 38ms/step - loss: 0.6657 - accuracy: 0.7669 - val_loss: 0.8662 - val_accuracy: 0.7117
Epoch 9/10
1563/1563 [==============================] - 61s 39ms/step - loss: 0.6341 - accuracy: 0.7774 - val_loss: 0.8885 - val_accuracy: 0.7024
Epoch 10/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.5967 - accuracy: 0.7906 - val_loss: 0.8459 - val_accuracy: 0.7155
```

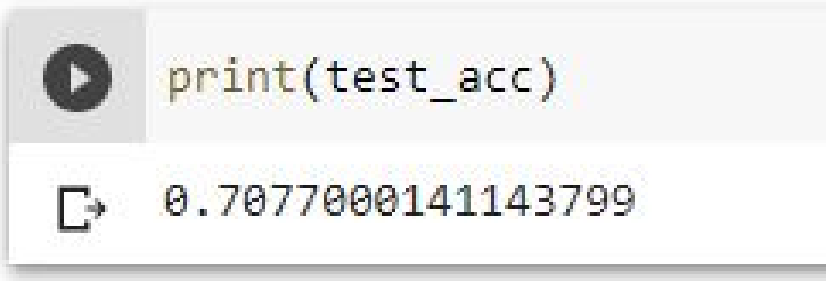
Evaluate the model

```
[ ]  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 - 4s - loss: 0.8459 - accuracy: 0.7155
```





Our simple CNN has achieved a test accuracy of over 70%.