Convolutional Neural Network (CNN)

Training a simple [Convolutional Neural Network

Import TensorFlow

```
import tensorflow as tf

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

Download and prepare the CIFAR10 dataset

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.

```
(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 [==============================] - 8s 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])
    # 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

In this example, we 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 = 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'))
```

```
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 (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
_________________________________________________________________
```

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.

Add Dense layers on top

To complete the model, you 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, you will flatten (or unroll) the 3D output to 1D, then add one or more Dense layers on top. CIFAR has 10 output classes, so you use a final Dense layer with 10 outputs.

```
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)        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
_________________________________________________________________
```

The network summary shows that (4, 4, 64) outputs were flattened into vectors of shape (1024) before going through two Dense layers.

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 [==============================] - 17s 5ms/step - loss: 1.5095 - accuracy: 0.4497 - val_loss: 1.2525 - val_accuracy: 0.5539
Epoch 2/10
1563/1563 [==============================] - 7s 5ms/step - loss: 1.1469 - accuracy: 0.5928 - val_loss: 1.0676 - val_accuracy: 0.6244
Epoch 3/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.9936 - accuracy: 0.6498 - val_loss: 1.0064 - val_accuracy: 0.6456
Epoch 4/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.8924 - accuracy: 0.6865 - val_loss: 0.9440 - val_accuracy: 0.6718
Epoch 5/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.8134 - accuracy: 0.7145 - val_loss: 0.9230 - val_accuracy: 0.6751
Epoch 6/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.7490 - accuracy: 0.7363 - val_loss: 0.8562 - val_accuracy: 0.7044
Epoch 7/10
1563/1563 [==============================] - 7s 4ms/step - loss: 0.6962 - accuracy: 0.7552 - val_loss: 0.8469 - val_accuracy: 0.7078
Epoch 8/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.6512 - accuracy: 0.7716 - val_loss: 0.8740 - val_accuracy: 0.7065
Epoch 9/10
1563/1563 [==============================] - 7s 4ms/step - loss: 0.6175 - accuracy: 0.7825 - val_loss: 0.8751 - val_accuracy: 0.7111
Epoch 10/10
1563/1563 [==============================] - 7s 4ms/step - loss: 0.5789 - accuracy: 0.7966 - val_loss: 0.8970 - val_accuracy: 0.7058
```

Evaluate the model

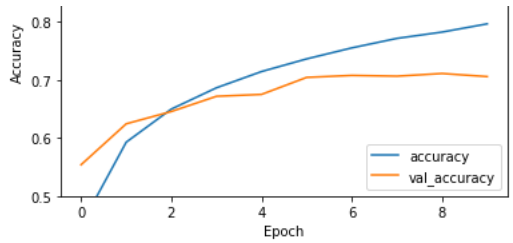
```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 1s - loss: 0.8970 - accuracy: 0.7058 - 703ms/epoch - 2ms/step
```

```
1 print(test_acc)
```

```
0.7057999968528748
```



✓ 0s completed at 12:44 AM