PRACTICAL 9

Image Classification (CIFAR-10) with Data Augmentation

ACTIVITY - Submission Required!

In this exercise, we will be working on the CIFAR-10 dataset (Canadian Institute for Advanced Research), which is composed of 60,000 images of 10 different classes: aeroplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. We will build a CNN model and use data augmentation to recognize these categories.

Perform the following steps to complete this exercise:

***NOTE**: You can read more about this dataset on TensorFlow's website: https://www.tensorflow.org/api_docs/python/tf/keras/datasets/cifar10*

1. Open a new Jupyter Notebook file.
2. Import **tensorflow.keras.datasets.cifar10**

```
from tensorflow.keras.datasets import cifar10
```

3. Load the CIFAR-10 dataset using:

```
(features_train, label_train), (features_test, label_test) = cifar10
.load_data()
```

4. Create three variables called **batch_size**, **img_height**, and **img_width** that take the values **16**, **32**, and **32**, respectively:

```
batch_size = 16
img_height = 32
img_width = 32
```

Batch size is the term used in machine learning and refers to the number of training samples utilized in one iteration. For instance, there are a total of **3000** training samples and the batch size is **32**. The epoch to train the samples is **500. 32** samples will be taken at a time to train the network. To go through all 3000 samples, it takes 94 (3000/32) iterations -> 1 epoch.

5. Import **ImageDataGenerator** from **tensorflow.keras.preprocessing**:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

6. Create an **ImageDataGenerator** instance called **train_img_gen** with data augmentation: **rescaling** (by dividing by 255), **width_shift_range = 0.1**, **height_shift_range=0.1**, and horizontal flipping:

```
train_img_gen = ImageDataGenerator(
            rescale=1./255,
```

```
            width_shift_range=0.1,
            height_shift_range=0.1,
            horizontal_flip=True)
```

7. Create an **ImageDataGenerator** instance called **val_img_gen** with rescaling (by dividing by 255):

```
val_img_gen = ImageDataGenerator(rescale=1./255)
```

8. Create a data generator called **train_data_gen** using the **.flow()** method and specify the batch size, features, and labels from the training set:

```
train_data_gen = train_img_gen.flow(
    features_train,
    label_train,
    batch_size = batch_size)
```

9. Create a data generator called **val_data_gen** using the **.flow()** method and specify the batch size, features, and labels from the testing set:

```
val_data_gen = train_img_gen.flow(
    features_test,
    label_test,
    batch_size=batch_size)
```

10. Import **numpy** as **np**, **tensorflow** as **tf**, and **layers** from **tensorflow.keras**:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
```

11. Set **8** as the seed for **numpy** and **tensorflow** using **np.random_seed()** and **tf.random.set_seed().**

```
np.random.seed(8)
tf.random.set_seed(8)
```

12. Instantiate a **tf.keras.Sequential()** class into a variable called **model** with the and add the following layers: a convolution layer with **64** kernels of shape **3**, ReLU as the activation function, and the necessary input dimensions; a max pooling layer; a convolution layer with **128** kernels of shape **3** and ReLU as the activation function; a max pooling layer; a flatten layer; a fully connected layer with **128** units and ReLU as the activation function; a fully connected layer with **10** units and Softmax as the activation function.

```
model = tf.keras.Sequential()
model.add(layers.Conv2D(64, 3, activation='relu', input_shape=(img_h
eight, img_width, 3)))
model.add(layers.MaxPooling2D())
model.add(layers.Conv2D(128, 3, activation='relu'))
```

```
model.add(layers.MaxPooling2D())
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add( layers.Dense(10, activation='softmax'))
```

13. Print the summary of the model.

```
model.summary()
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 30, 30, 64)        1792

 max_pooling2d (MaxPooling2D  (None, 15, 15, 64)       0
 )

 conv2d_2 (Conv2D)           (None, 13, 13, 128)       73856

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 128)               589952

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 666,890
Trainable params: 666,890
Non-trainable params: 0
```

14. Instantiate a **tf.keras.optimizers.Adam()** class with **0.001** as the learning rate and save it to a variable called **optimizer**:

```
optimizer = tf.keras.optimizers.Adam(0.001)
```

15. Compile the neural network using **.compile()** with **loss = 'sparse_categorical_crossentropy', optimizer=optimizer, metrics=['accuracy']**:

```
model.compile(loss='sparse_categorical_crossentropy', optimizer=opti
mizer, metrics=['accuracy'])
```

16. Fit the neural networks with **fit_generator()** and provide the train and validation data generators, **epochs=5**, the steps per epoch, and the validation steps:

```
model.fit_generator(
    train_data_gen,
    steps_per_epoch=len(features_train) // batch_size,
    epochs=5,
    validation_data=val_data_gen,
    validation_steps=len(features_test) // batch_size
```

```
)
```

```
Epoch 1/5
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning: `Model.fit_generator` is deprecated and will be removed in a

3125/3125 [==============================] - 177s 56ms/step - loss: 1.4925 - accuracy: 0.0994 - val_loss: 1.2488 - val_accuracy: 0.1086
Epoch 2/5
3125/3125 [==============================] - 177s 57ms/step - loss: 1.1793 - accuracy: 0.0987 - val_loss: 1.0931 - val_accuracy: 0.0762
Epoch 3/5
3125/3125 [==============================] - 179s 57ms/step - loss: 1.0734 - accuracy: 0.0994 - val_loss: 1.0361 - val_accuracy: 0.1142
Epoch 4/5
3125/3125 [==============================] - 179s 57ms/step - loss: 1.0130 - accuracy: 0.1000 - val_loss: 0.9872 - val_accuracy: 0.1050
Epoch 5/5
3125/3125 [==============================] - 180s 57ms/step - loss: 0.9602 - accuracy: 0.1002 - val_loss: 1.0235 - val_accuracy: 0.0858
<keras.callbacks.History at 0x7f3438535f10>
```

We've trained our CNN for five epochs and achieved an accuracy score of **0.1002** for the training set, and **0.0858** for the validation set. Our model is overfitting quite a lot. You may want to try the training with different architectures to see whether you can improve this score and reduce overfitting. You can also try feeding this model with some images of cats or dogs of your choice and see the output predictions.

Activity

1. Write a code to print the shape of **features_train, label_train, features_test, and label_test** and explain the output.
2. Explain the output generated by the ***model.fit_generator.***
3. Modify the ImageDataGenerator function by adding vertical_flip, shear_range, rotation_range and brightness_range and fit the model with the same network design. Compare the result with the first model.

****