wPRACTICAL 10

Callbacks: 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 and use the Keras callbacks. 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*

Submit the work in the VLE.

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()
```

Question 1: Print the shape of the features_train, label_train, features_test, label_test and explain the output.

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**:

```
.........
.........
```

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(.......................)
```

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

```
val_img_gen = ImageDataGenerator(............)
```

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** 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 flattened 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()
..............................
..............................
```

```
................................
```

13. Print the summary of the model.

```
...................................
```

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']**:

```
.....................................
.....................................
```

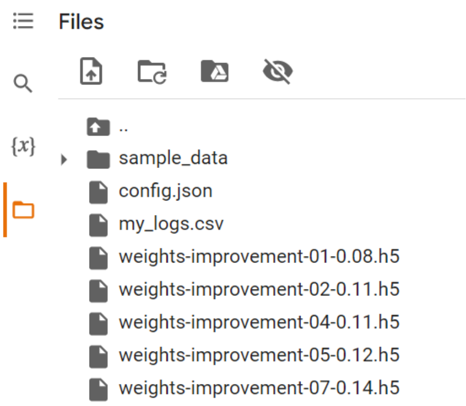
16. Use Keras callbacks functions for EarlyStopping, ModelCheckPoint and CSVLogger

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoin
t, Callback, CSVLogger

filepath = "weights-improvement-{epoch:02d}-{val_accuracy:.2f}.h5"
checkpoint = ModelCheckpoint(filepath, monitor = 'val_accuracy', sav
e_best_only = True, mode="max")
early_stop = EarlyStopping(monitor='val_loss', patience=3)

callbacks_list = [checkpoint, early_stop]
```

17. Fit the neural networks with **fit_generator()** and provide the train and validation data generators, **epochs=10**, the **steps per epoch**, and the **validation steps**:

```
model.fit(
    train_data_gen,
    steps_per_epoch=len(features_train) // batch_size,
    epochs = 10,
    validation_data=val_data_gen,
    validation_steps=len(features_test) // batch_size,
    callbacks = callbacks_list
    )
```

18. The output generated by callbacks can be viewed in the **files** bar.



my_logs.csv

1 to 10 of 10 entries Filter

epoch	accuracy	loss	val_accuracy	val_loss
0	0.10440000146627426	1.4606481790542603	0.0803999975323677	1.195829153060913
1	0.09907999634742737	1.1452711820602417	0.10809999704360962	1.0597715377807617
2	0.09988000243902206	1.0469633340835571	0.08160000294446945	1.0429524183273315
3	0.10013999789953232	0.9780482053756714	0.10830000042915344	0.9672864079475403
4	0.1004600003361702	0.928399920463562	0.12099999934434891	0.932385265827179
5	0.10174000263214111	0.8945835828781128	0.11190000176429749	0.9305040836334229
6	0.10056000202894211	0.860787570476532	0.13760000467300415	0.9668407440185547
7	0.10115999728441238	0.8386303186416626	0.11020000278949738	0.8798078298568726
8	0.10044000297784805	0.8160250186920166	0.11010000109672546	0.8476881980895996
9	0.10106000304222107	0.8029830455780029	0.08250000327825546	0.8705777525901794

Activity

1. Use **CSVLogger** callback function to save all the training logs. Refer CSVLogger documentation.
2. Use TensorBoard to visualize the trained models (Refer to slide)

****