TensorBoard with Fashion MNIST

In this week's exercise you will train a convolutional neural network to classify images of the Fashion MNIST dataset and you will use TensorBoard to explore how it's confusion matrix evolves over time.

Setup

```
# Load the TensorBoard notebook extension.
%load_ext tensorboard

import io
import itertools
import numpy as np
import sklearn.metrics
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow import keras
from datetime import datetime

from os import getcwd


print("TensorFlow version: ", tf.__version__)
```

```
TensorFlow version:  2.9.1
```

Load the Fashion-MNIST Dataset

We are going to use a CNN to classify images in the the Fashion-MNIST dataset. This dataset consist of 70,000 grayscale images of fashion products from 10 categories, with 7,000 images per category. The images have a size of 28×28 pixels.

First, we load the data. Even though these are really images, we will load them as NumPy arrays and not as binary image objects. The data is already divided into training and testing sets.

```
# Load the data.
train_images = np.load(f"{getcwd()}/../tmp2/train_images.npy")
train_labels = np.load(f"{getcwd()}/../tmp2/train_labels.npy")

test_images = np.load(f"{getcwd()}/../tmp2/test_images.npy")
test_labels = np.load(f"{getcwd()}/../tmp2/test_labels.npy")

# The labels of the images are integers representing classes.
# Here we set the Names of the integer classes, i.e., 0 ->
T-short/top, 1 -> Trouser, etc.
```

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Format the Images

`train_images` is a NumPy array with shape `(60000, 28, 28)` and `test_images` is a NumPy array with shape `(10000, 28, 28)`. However, our model expects arrays with shape `(batch_size, height, width, channels)` . Therefore, we must reshape our NumPy arrays to also include the number of color channels. Since the images are grayscale, we will set `channels` to `1`. We will also normalize the values of our NumPy arrays to be in the range `[0,1]`.

```
# Pre-process images
train_images = train_images.reshape(60000, 28, 28, 1)
train_images = train_images / 255.0

test_images = test_images.reshape(10000, 28, 28, 1)
test_images = test_images / 255.0
```

Build the Model

We will build a simple CNN and compile it.

```
# Build the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64, (3,3), activation='relu',
input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

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

Plot Confusion Matrix

When training a classifier, it's often useful to see the confusion matrix. The confusion matrix gives you detailed knowledge of how your classifier is performing on test data.

In the cell below, we will define a function that returns a Matplotlib figure containing the plotted confusion matrix.

```
def plot_confusion_matrix(cm, class_names):
    """
    Returns a matplotlib figure containing the plotted confusion
matrix.

    Args:
       cm (array, shape = [n, n]): a confusion matrix of integer
classes
       class_names (array, shape = [n]): String names of the integer
classes
    """

    figure = plt.figure(figsize=(8, 8))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title("Confusion matrix")
    plt.colorbar()
    tick_marks = np.arange(len(class_names))
    plt.xticks(tick_marks, class_names, rotation=45)
    plt.yticks(tick_marks, class_names)

    # Normalize the confusion matrix.
    cm = np.around(cm.astype('float') / cm.sum(axis=1)[:, np.newaxis],
decimals=2)

    # Use white text if squares are dark; otherwise black.
    threshold = cm.max() / 2.

    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        color = "white" if cm[i, j] > threshold else "black"
        plt.text(j, i, cm[i, j], horizontalalignment="center",
color=color)

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    return figure
```

TensorBoard Callback

We are now ready to train the CNN and regularly log the confusion matrix during the process. In the cell below, you will create a Keras TensorBoard callback to log basic metrics.

```
# UNQ_C1
# GRADED CODE: tensorboard_callback

# Clear logs prior to logging data.
!rm -rf logs/image
```

```
# Create log directory
logdir = "logs/image/" + datetime.now().strftime("%Y%m%d-%H%M%S")

# EXERCISE: Define a TensorBoard callback. Use the log_dir parameter
# to specify the path to the directory where you want to save the
# log files to be parsed by TensorBoard.
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=logdir)

file_writer_cm = tf.summary.create_file_writer(logdir + '/cm')
```

Convert Matplotlib Figure to PNG

Unfortunately, the Matplotlib file format cannot be logged as an image, but the PNG file format can be logged. So, you will create a helper function that takes a Matplotlib figure and converts it to PNG format so it can be written.

```
# UNQ_C2
# GRADED function: plot_to_image

def plot_to_image(figure):
    """
    Converts the matplotlib plot specified by 'figure' to a PNG image
and
    returns it. The supplied figure is closed and inaccessible after
this call.
    """

    buf = io.BytesIO()

    # EXERCISE: Use plt.savefig to save the plot to a PNG in memory.
    plt.savefig(buf, format='png')

    # Closing the figure prevents it from being displayed directly
inside
    # the notebook.
    plt.close(figure)
    buf.seek(0)

    # EXERCISE: Use tf.image.decode_png to convert the PNG buffer
    # to a TF image. Make sure you use 4 channels.
    image = tf.image.decode_png(buf.getvalue(), channels=4)

    # EXERCISE: Use tf.expand_dims to add the batch dimension
    image = tf.image.decode_png(buf.getvalue(), channels=4)

    return image
```

Confusion Matrix

In the cell below, you will define a function that calculates the confusion matrix.

```
# UNQ_C3
# GRADED function: log_confusion_matrix

def log_confusion_matrix(epoch, logs):

    # EXERCISE: Use the model to predict the values from the 
test_images.
    test_pred_raw = model.predict(test_images)

    test_pred = np.argmax(test_pred_raw, axis=1)

    # EXERCISE: Calculate the confusion matrix using sklearn.metrics
    cm = sklearn.metrics.confusion_matrix(test_labels, test_pred)

    figure = plot_confusion_matrix(cm, class_names=class_names)
    cm_image = plot_to_image(figure)

    # Log the confusion matrix as an image summary.
    with file_writer_cm.as_default():
        tf.summary.image("Confusion Matrix", cm_image, step=epoch)

# Define the per-epoch callback.
cm_callback = 
keras.callbacks.LambdaCallback(on_epoch_end=log_confusion_matrix)
```

Running TensorBoard

The next step will be to run the code shown below to render the TensorBoard. Unfortunately, TensorBoard cannot be rendered within the Coursera environment. Therefore, we won't run the code below.

```
# Start TensorBoard.
%tensorboard --logdir logs/image

# Train the classifier.
model.fit(train_images,
          train_labels,
          epochs=5,
          verbose=0, # Suppress chatty output
          callbacks=[tensorboard_callback, cm_callback],
          validation_data=(test_images, test_labels))
```

However, you are welcome to download the notebook and run the above code locally on your machine or in Google's Colab to see TensorBoard in action. Below are some example screenshots that you should see when executing the code:

Submission Instructions

```
# Now click `File -> Save and Checkpoint` and press the 'Submit
Assignment' button above.
```

When you're done or would like to take a break, please run the two cells below to save your work and close the Notebook. This frees up resources for your fellow learners.

```
%%javascript
<!-- Save the notebook -->
IPython.notebook.save_checkpoint();
```

```
<IPython.core.display.Javascript object>
```

```
%%javascript
<!-- Shutdown and close the notebook -->
window.onbeforeunload = null
window.close();
IPython.notebook.session.delete();
```

```
<IPython.core.display.Javascript object>
```