# 365 DataScience Tensorboard - Confusion matrix

```
# We will use the MNIST dataset to train the network for this example

# Importing the relevant packages
import io
import itertools

import numpy as np
import sklearn.metrics

import matplotlib.pyplot as plt

import tensorflow as tf
import tensorflow_datasets as tfds
```

## Downloading and preprocessing the data

```
# Defining some constants/hyperparameters
BUFFER_SIZE = 70_000 # for reshuffling
BATCH_SIZE = 128
NUM_EPOCHS = 20

# Downloading the MNIST dataset
mnist_dataset, mnist_info = tfds.load(name='mnist', with_info=True, as_
supervised=True)

mnist_train, mnist_test = mnist_dataset['train'], mnist_dataset['test']

# Creating a function to scale our data
def scale(image, label):
    image = tf.cast(image, tf.float32)
    image /= 255.

    return image, label

# Scaling the data
train_and_validation_data = mnist_train.map(scale)
test_data = mnist_test.map(scale)

# Defining the size of validation set
num_validation_samples = 0.1 * mnist_info.splits['train'].num_examples
num_validation_samples = tf.cast(num_validation_samples, tf.int64)

# Defining size of test set
num_test_samples = mnist_info.splits['test'].num_examples
num_test_samples = tf.cast(num_test_samples, tf.int64)
```

```python
# Reshuffling the dataset
train_and_validation_data = train_and_validation_data.shuffle(BUFFER_SI
ZE)

# Splitting the dataset into training + validation
train_data = train_and_validation_data.skip(num_validation_samples)
validation_data = train_and_validation_data.take(num_validation_samples)

# Batching the data
train_data = train_data.batch(BATCH_SIZE)
validation_data = validation_data.batch(num_validation_samples)
test_data = test_data.batch(num_test_samples)

# Extracting the numpy arrays from the validation data for the calculat
ion of the Confusion Matrix
for images, labels in validation_data:
    images_val = images.numpy()
    labels_val = labels.numpy()
```

## Creating the model and training it

```python
# Now that we have preprocessed the dataset, we can define our CNN and
train it

# Outlining the model/architecture of our CNN
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(50, 5, activation='relu', input_shape=(28, 2
8, 1)),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(50, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(10)
])

# A brief summary of the model and parameters
model.summary(line_length = 75)
```

```
Model: "sequential"
___________________________________________________________________________
Layer (type)                     Output Shape                   Param #
===========================================================================
conv2d (Conv2D)                  (None, 24, 24, 50)             1300
___________________________________________________________________________
max_pooling2d (MaxPooling2D)     (None, 12, 12, 50)             0
___________________________________________________________________________
```

```
____
conv2d_1 (Conv2D)            (None, 10, 10, 50)        22550

_________________________________________________________________
____
max_pooling2d_1 (MaxPooling2D)   (None, 5, 5, 50)      0

_________________________________________________________________
____
flatten (Flatten)            (None, 1250)              0

_________________________________________________________________
____
dense (Dense)                (None, 10)                12510

=================================================================
====
Total params: 36,360
Trainable params: 36,360
Non-trainable params: 0
_________________________________________________________________
____
```

```
# Defining the loss function
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tru
e)

# Compiling the model with Adam optimizer and the cathegorical crossent
ropy as a loss function
model.compile(optimizer='adam', loss=loss_fn, metrics=['accuracy'])

# Defining logging directory
log_dir = "Logs\\fit\\" + "run-1"

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 class
es
    """
    figure = plt.figure(figsize=(12, 12))
    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=co
lor)

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

    return figure

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

    # Save the plot to a PNG in memory.
    buf = io.BytesIO()
    plt.savefig(buf, format='png')

    # Closing the figure prevents it from being displayed directly insi
de the notebook.
    plt.close(figure)

    buf.seek(0)

    # Convert PNG buffer to TF image
    image = tf.image.decode_png(buf.getvalue(), channels=4)

    # Add the batch dimension
    image = tf.expand_dims(image, 0)

    return image

# Define a file writer variable for logging purposes
file_writer_cm = tf.summary.create_file_writer(log_dir + '/cm')

def log_confusion_matrix(epoch, logs):
    # Use the model to predict the values from the validation dataset.
    test_pred_raw = model.predict(images_val)
```

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

    # Calculate the confusion matrix.
    cm = sklearn.metrics.confusion_matrix(labels_val, test_pred)

    # Log the confusion matrix as an image summary. Define the class na
mes to be displayed.
    figure = plot_confusion_matrix(cm, class_names=['0', '1', '2', '3',
 '4', '5', '6', '7', '8', '9'])
    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)

# Defining the callbacks
cm_callback = tf.keras.callbacks.LambdaCallback(on_epoch_end=log_confus
ion_matrix)
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
histogram_freq=1, profile_batch=0)

# Defining early stopping to prevent overfitting
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor = 'val_loss',
    mode = 'auto',
    min_delta = 0,
    patience = 2,
    verbose = 0,
    restore_best_weights = True
)

# Train the network
model.fit(
    train_data,
    epochs = NUM_EPOCHS,
    callbacks = [tensorboard_callback, cm_callback, early_stopping],
    validation_data = validation_data,
    verbose = 2
)
```

```
Epoch 1/20
422/422 - 19s - loss: 0.2657 - accuracy: 0.9233 - val_loss: 0.0915 - va
l_accuracy: 0.9755
Epoch 2/20
422/422 - 20s - loss: 0.0719 - accuracy: 0.9776 - val_loss: 0.0670 - va
l_accuracy: 0.9820
Epoch 3/20
422/422 - 20s - loss: 0.0522 - accuracy: 0.9845 - val_loss: 0.0427 - va
l_accuracy: 0.9858
Epoch 4/20
```

```
422/422 - 20s - loss: 0.0442 - accuracy: 0.9863 - val_loss: 0.0325 - va
l_accuracy: 0.9910
Epoch 5/20
422/422 - 20s - loss: 0.0367 - accuracy: 0.9886 - val_loss: 0.0273 - va
l_accuracy: 0.9912
Epoch 6/20
422/422 - 20s - loss: 0.0319 - accuracy: 0.9899 - val_loss: 0.0334 - va
l_accuracy: 0.9908
Epoch 7/20
422/422 - 20s - loss: 0.0289 - accuracy: 0.9910 - val_loss: 0.0227 - va
l_accuracy: 0.9925
Epoch 8/20
422/422 - 20s - loss: 0.0250 - accuracy: 0.9924 - val_loss: 0.0245 - va
l_accuracy: 0.9918
Epoch 9/20
422/422 - 20s - loss: 0.0208 - accuracy: 0.9937 - val_loss: 0.0170 - va
l_accuracy: 0.9943
Epoch 10/20
422/422 - 20s - loss: 0.0199 - accuracy: 0.9936 - val_loss: 0.0230 - va
l_accuracy: 0.9918
Epoch 11/20
422/422 - 21s - loss: 0.0175 - accuracy: 0.9945 - val_loss: 0.0169 - va
l_accuracy: 0.9940
Epoch 12/20
422/422 - 21s - loss: 0.0154 - accuracy: 0.9952 - val_loss: 0.0136 - va
l_accuracy: 0.9950
Epoch 13/20
422/422 - 21s - loss: 0.0130 - accuracy: 0.9957 - val_loss: 0.0097 - va
l_accuracy: 0.9967
Epoch 14/20
422/422 - 20s - loss: 0.0126 - accuracy: 0.9963 - val_loss: 0.0090 - va
l_accuracy: 0.9973
Epoch 15/20
422/422 - 20s - loss: 0.0108 - accuracy: 0.9966 - val_loss: 0.0072 - va
l_accuracy: 0.9973
Epoch 16/20
422/422 - 20s - loss: 0.0085 - accuracy: 0.9975 - val_loss: 0.0048 - va
l_accuracy: 0.9992
Epoch 17/20
422/422 - 20s - loss: 0.0087 - accuracy: 0.9971 - val_loss: 0.0047 - va
l_accuracy: 0.9987
Epoch 18/20
422/422 - 20s - loss: 0.0072 - accuracy: 0.9979 - val_loss: 0.0064 - va
l_accuracy: 0.9978
Epoch 19/20
422/422 - 21s - loss: 0.0082 - accuracy: 0.9973 - val_loss: 0.0076 - va
l_accuracy: 0.9977

<tensorflow.python.keras.callbacks.History at 0x22d08cdffd0>
```

## Testing our model

```
# Testing our model
test_loss, test_accuracy = model.evaluate(test_data)
```

```
1/1 [==============================] - ETA: 0s - loss: 0.0311 - accurac
y: 0.99 - 0s 3ms/step - loss: 0.0311 - accuracy: 0.9911
```

```
# Printing the test results
print('Test loss: {0:.4f}. Test accuracy: {1:.2f}%'.format(test_loss, t
est_accuracy*100.))
```

```
Test loss: 0.0311. Test accuracy: 99.11%
```

## Visualizing in Tensorboard

```
# The confusion matrix can be found in the 'Image' tab
```

```
# Loading the Tensorboard extension
%load_ext tensorboard
%tensorboard --logdir "logs/fit"
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6006 (pid 1256), started 0:01:06 ago. (Use
'!kill 1256' to kill it.)
```

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

```
# NOTE: On Windows, TensorBoard has trouble starting if the extension h
as been running. So, the first time you start it,
# it will run properly. But if you subsequently try to restart it, or o
pen a different directory,
# the extension will encounter an error. Luckily, there is a quick work
around. All you need to do
# is write 2 commands in the shell. First, open the command prompt, or
‘cmd.exe’. In there,
# you need to paste the following 2 lines , one after another:
#
# taskkill /im TensorBoard.exe /f
# del /q %TMP%\.TensorBoard-info\*
#
# These will end already active TensorBoard processes and clean the tem
porary data associated with TensorBoard,
# so you can run it again. If either of those gives an error, that’s ok
ay: you can ignore it.
```

Start your 365 Journey!