Basics

`from torch.utils.tensorboard import SummaryWriter`

Define a writer

- `writer = SummaryWriter(f'runs/MNIST/test)`
  - We are going to write to the runs folder, MNIST, test folder, inside our project dir

At the end of every batch, we add:

- `writer.add_scalar('Training loss', loss, global_step=step)`
  - Step starts at 0 and is iterated every batch
  - Prints out the loss as it is calculated in a graph titled Training loss

Go to the dir you are training in and do:

- `tensorboard --logdir runs`
  - You get localhost:6006

You can create a new writer for every hyper parameter setting and save it with a slightly different name and then you can sort by the name in tensorboard

- You can also use `writer.add_hparams(dictionary_of_hyperparams, dictionary_of_results)`
  - This gives you a new tab called hparams where you can toggle metrics on or off
  - You can then go to **Parallel Coordinates View** so you can see the impact of hparams on results easily

Data Types We Can Display

- Scalars, Images, Distributions, Histograms, HParams

Displaying Images

- `img_grid = torchvision.utils.make_grid(data)`
- Then every batch, we do `writer.add_image('mnist_images', img_grid)`

Displaying Weights

- `writer.add_historgram('fc1', model.fc1.weight)`
  - Prints out weights in the last layer of a cnn, which is a fully connected linear layer
  - We get distributions of weights every iteration

Embedding Projector

- Visual embeddings to visualise how model does the predictions
- `writer.add_embedding(features, metadata=class_labels, label_img=data.unsqueeze(1), global_step=batch_idx)`
  - `features = data.reshape(data.shape[0], -1)`
    - `data` = the inputs (batch_size, dim1, dim2)
    - Embeddings expects (batch_size, number_of_channels, dim1, dim2), so we need to unsqueeze 1
      - If we had multi-channel inputs, we wouldn't need this
        - Some single-channel inputs like MNIST are already in this format
  - `class_labels = [classes[label] for label in predictions]`
    - `classes` = a list of possible label names (e.g. 'cat', 'dog', etc)
    - Basically creates a list of the names of the predicted outputs
- Uses things like PCA projection to lower the dimensions into fewer (like 3 for 3D projections) dimensions so we can see how the class predictions are distributed

Callbacks

We can use model.compile(), then we can create a tensorboard_callback

- ```
  Tensorboard_callback = keras.callbacks.TensorBoard(
          log_dir ="tb_callback_dir", historgram_freq=1
  )
  ```
- Then add callback=[tensorboard_callback] to model.fit()
- This plots the epoch loss and accuracy every step automatically

Confusion Matrix

We want to update the confusion matrix as we do every batch
Create an empty cm: np.zeros((len(class_names), len(class_names)))
Each batch, do confusion += get_confusion_matrix(y, y_pred, class_names)
Then do:

```
with train_writer.as_default():
    tf.summary.image(
        "Confusion Matrix",
        plot_confusion_matrix(cm / batch_idx, class_names),
        step=epoch
    )
```

We have to define get_confusion_matrix, and plot_confusion_matrix:

- Get_confusion_matrix:
  - Get preds with argmax
  - Use sklearn metrics confusion matrix for that epoch
- Plot_confusion_matrix:
  - Plot the matrix with matplotlib
  - Normalize cm with np.around
  - Threshold the cm
  - Return the image

Profiler

Install tensorflow_plugin_profile
Import tensorflow_datasets as tfds

Normalize image
Create a model
Do some logs using datetime.now() so you get a new file every time you run the code

We can then open up tensorboard

```
%load_ext tensorboard #  load tensorboard notebook extension
%tensorboard --logdir=logs #  launches tensorboard and navigates to profile tab
```

This displays a performance summary page, with a couple of tabs, which shows the summary of how long everything takes, the the amount of operations
Gives you some tips on how to improve the performance
Things like using **`.cache()`** and **`prefetch(autotune)`** (careful with running out of memory)
```