# PyTorch routine

For a total beginener to PyTorch, visit this website

## 0. Setting up on Colab

Without a GPU, I always need to rely on Google Colab to train my model. A Colab instance is always come prepared with torch, but in case it does not, you need to set up with `pip`. I often install other modules to help with the post-training process. You need to pay Google for a CLI, but you could also make do with the code cell!

```
!pip install torch torchmetrics mlxtend
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public
/simple/
Requirement already satisfied: torch in /usr/local/lib/python3.8/dist-packages
(1.13.0+cu116)
Collecting torchmetrics
  Downloading torchmetrics-0.11.0-py3-none-any.whl (512 kB)
     |████████████████████████████████| 512 kB 4.5 MB/s
Requirement already satisfied: mlxtend in /usr/local/lib/python3.8/dist-packages (0.14.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.8/dist-packages
(from torch) (4.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-packages (from
torchmetrics) (21.3)
Requirement already satisfied: numpy>=1.17.2 in /usr/local/lib/python3.8/dist-packages
(from torchmetrics) (1.21.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-packages (from
mlxtend) (57.4.0)
Requirement already satisfied: pandas>=0.17.1 in /usr/local/lib/python3.8/dist-packages
(from mlxtend) (1.3.5)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.8/dist-packages
(from mlxtend) (1.0.2)
Requirement already satisfied: matplotlib>=1.5.1 in /usr/local/lib/python3.8/dist-packages
(from mlxtend) (3.2.2)
Requirement already satisfied: scipy>=0.17 in /usr/local/lib/python3.8/dist-packages (from
mlxtend) (1.7.3)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
/lib/python3.8/dist-packages (from matplotlib>=1.5.1->mlxtend) (3.0.9)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8/dist-
packages (from matplotlib>=1.5.1->mlxtend) (2.8.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages (from
matplotlib>=1.5.1->mlxtend) (0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages
(from matplotlib>=1.5.1->mlxtend) (1.4.4)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from
pandas>=0.17.1->mlxtend) (2022.6)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from
```

```
python-dateutil>=2.1->matplotlib>=1.5.1->mlxtend) (1.15.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.8/dist-packages (from
scikit-learn>=0.18->mlxtend) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-
packages (from scikit-learn>=0.18->mlxtend) (3.1.0)
Installing collected packages: torchmetrics
Successfully installed torchmetrics-0.11.0
```

```
!pip install --upgrade mlxtend
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public
/simple/
Requirement already satisfied: mlxtend in /usr/local/lib/python3.8/dist-packages (0.14.0)
Collecting mlxtend
  Downloading mlxtend-0.21.0-py2.py3-none-any.whl (1.3 MB)
     |████████████████████████████████| 1.3 MB 4.9 MB/s
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.8/dist-
packages (from mlxtend) (1.0.2)
Requirement already satisfied: matplotlib>=3.0.0 in /usr/local/lib/python3.8/dist-packages
(from mlxtend) (3.2.2)
Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-packages (from
mlxtend) (57.4.0)
Requirement already satisfied: scipy>=1.2.1 in /usr/local/lib/python3.8/dist-packages (from
mlxtend) (1.7.3)
Requirement already satisfied: numpy>=1.16.2 in /usr/local/lib/python3.8/dist-packages
(from mlxtend) (1.21.6)
Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.8/dist-packages
(from mlxtend) (1.2.0)
Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.8/dist-packages
(from mlxtend) (1.3.5)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
/lib/python3.8/dist-packages (from matplotlib>=3.0.0->mlxtend) (3.0.9)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages
(from matplotlib>=3.0.0->mlxtend) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages (from
matplotlib>=3.0.0->mlxtend) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8/dist-
packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from
pandas>=0.24.2->mlxtend) (2022.6)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from
python-dateutil>=2.1->matplotlib>=3.0.0->mlxtend) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-
packages (from scikit-learn>=1.0.2->mlxtend) (3.1.0)
Installing collected packages: mlxtend
  Attempting uninstall: mlxtend
    Found existing installation: mlxtend 0.14.0
    Uninstalling mlxtend-0.14.0:
      Successfully uninstalled mlxtend-0.14.0
Successfully installed mlxtend-0.21.0
```

```
# Any other of your choice!
```

# 1. Import modules

```
# Generic torch process
from torch import nn
import torch
from torch.utils.data import DataLoader

# Specifically for computer vision
import torchvision
from torchvision import datasets
from torchvision.transforms import ToTensor #Among others

# For post-training process
from torchmetrics import ConfusionMatrix, Accuracy #Among others
from mlxtend.plotting import plot_confusion_matrix #Among others

# Import tqdm for progress bar
from tqdm.auto import tqdm

# Other module(s)
import matplotlib.pyplot as plt
import gc
```

## A piece of device-agnostic code

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

# 2. Loading data

You load dataset in with `datasets`, and get them ready for the model with `DataLoader`, applying all the transformation you want or need.

The example is for the classic dataset - ~~MNIST~~ Kuzushiji-MNIST!

```
train_data = datasets.KMNIST(root='DATA',
                                transform=ToTensor(),
                                download=True)

test_data = datasets.KMNIST(root='DATA',
                               transform=ToTensor(),
                               download=True,
                               train=False)
```

```
Downloading http://codh.rois.ac.jp/kmnist/dataset/kmnist/train-images-idx3-ubyte.gz
Downloading http://codh.rois.ac.jp/kmnist/dataset/kmnist/train-images-idx3-ubyte.gz to
DATA/KMNIST/raw/train-images-idx3-ubyte.gz

Extracting DATA/KMNIST/raw/train-images-idx3-ubyte.gz to DATA/KMNIST/raw
```

```
Downloading http://codh.rois.ac.jp/kmnist/dataset/kmnist/train-labels-idx1-ubyte.gz
Downloading http://codh.rois.ac.jp/kmnist/dataset/kmnist/train-labels-idx1-ubyte.gz to
DATA/KMNIST/raw/train-labels-idx1-ubyte.gz

Extracting DATA/KMNIST/raw/train-labels-idx1-ubyte.gz to DATA/KMNIST/raw

Downloading http://codh.rois.ac.jp/kmnist/dataset/kmnist/t10k-images-idx3-ubyte.gz
Downloading http://codh.rois.ac.jp/kmnist/dataset/kmnist/t10k-images-idx3-ubyte.gz to
DATA/KMNIST/raw/t10k-images-idx3-ubyte.gz

Extracting DATA/KMNIST/raw/t10k-images-idx3-ubyte.gz to DATA/KMNIST/raw

Downloading http://codh.rois.ac.jp/kmnist/dataset/kmnist/t10k-labels-idx1-ubyte.gz
Downloading http://codh.rois.ac.jp/kmnist/dataset/kmnist/t10k-labels-idx1-ubyte.gz to
DATA/KMNIST/raw/t10k-labels-idx1-ubyte.gz

Extracting DATA/KMNIST/raw/t10k-labels-idx1-ubyte.gz to DATA/KMNIST/raw
```

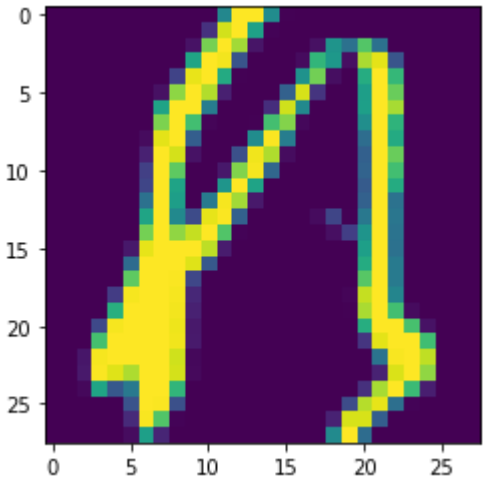
```
# Exploring training example
image, label = train_data[0]
image, label
```

```
(tensor([[[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.4627, 1.0000, 1.0000, 0.4863, 0.0039,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.1412, 0.9333, 1.0000, 0.5725, 0.0078, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0471, 0.7961, 1.0000, 0.8627, 0.0549, 0.0000, 0.0000,
           0.0000, 0.0431, 0.5176, 0.3725, 0.7333, 0.3725, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.5843, 1.0000, 0.9843, 0.2588, 0.0000, 0.0000, 0.0000,
           0.0275, 0.6588, 0.5333, 0.0392, 0.8745, 0.9608, 0.2627, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.2000, 0.9686, 1.0000, 0.6118, 0.0039, 0.0000, 0.0000, 0.0000,
           0.5098, 0.7882, 0.0353, 0.0000, 0.6196, 1.0000, 0.6667, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0431,
           0.8314, 1.0000, 0.8863, 0.0863, 0.0000, 0.0000, 0.0000, 0.3098,
           0.9412, 0.1490, 0.0000, 0.0000, 0.5608, 1.0000, 0.7725, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.5804,
           1.0000, 1.0000, 0.4196, 0.0000, 0.0000, 0.0000, 0.1294, 0.9294,
           0.4745, 0.0000, 0.0000, 0.0000, 0.5647, 1.0000, 0.8706, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.8039,
           1.0000, 0.6980, 0.0118, 0.0000, 0.0000, 0.0039, 0.7020, 0.8157,
```

```
 0.0196, 0.0000, 0.0000, 0.0000, 0.4941, 1.0000, 0.6510, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0275, 0.9608,
 1.0000, 0.2980, 0.0000, 0.0000, 0.0000, 0.4863, 0.9961, 0.4235,
 0.0000, 0.0000, 0.0000, 0.0000, 0.3569, 1.0000, 0.6118, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0902, 1.0000,
 0.9373, 0.0471, 0.0000, 0.0000, 0.2431, 0.9804, 0.8824, 0.0314,
 0.0000, 0.0000, 0.0000, 0.0000, 0.3059, 1.0000, 0.7725, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1882, 1.0000,
 0.7529, 0.0000, 0.0000, 0.0627, 0.8118, 0.9882, 0.3294, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.3176, 1.0000, 0.6980, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.2078, 1.0000,
 0.5804, 0.0000, 0.0000, 0.5216, 1.0000, 0.5804, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.2863, 1.0000, 0.6157, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.3725, 1.0000,
 0.5608, 0.0000, 0.4118, 0.9882, 0.8627, 0.0667, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.3098, 1.0000, 0.3765, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.5843, 1.0000,
 0.4745, 0.2275, 0.9647, 0.9961, 0.3098, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0275, 0.2078, 0.0039, 0.4000, 1.0000, 0.3922, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.7922, 1.0000,
 0.8353, 0.9137, 1.0000, 0.7294, 0.0078, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0392, 0.1843, 0.5176, 1.0000, 0.3804, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0627, 0.9647, 1.0000,
 1.0000, 1.0000, 0.9020, 0.0745, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.5647, 1.0000, 0.3725, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.3490, 1.0000, 1.0000,
 1.0000, 0.9529, 0.2784, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.6157, 1.0000, 0.4118, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.7529, 1.0000, 1.0000,
 0.9961, 0.2118, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.7412, 1.0000, 0.4078, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.1765, 0.9882, 1.0000, 1.0000,
 0.9882, 0.1098, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0118, 0.9137, 1.0000, 0.4549, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.6314, 1.0000, 1.0000, 1.0000,
 0.9843, 0.0902, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0039, 0.8745, 1.0000, 0.6980, 0.0275,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.2196, 0.9882, 1.0000, 1.0000, 1.0000,
 0.9765, 0.0667, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.6235, 1.0000, 1.0000, 0.7098,
 0.0471, 0.0000, 0.0000, 0.0000]
```

```
 0.0471, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.7059, 1.0000, 1.0000, 1.0000, 1.0000,
 0.9804, 0.0706, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.1529, 0.9608, 1.0000, 1.0000,
 0.6941, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0588, 0.9882, 1.0000, 1.0000, 1.0000, 1.0000,
 0.9294, 0.0196, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.3804, 0.9922, 1.0000,
 0.9059, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0667, 0.9961, 0.9529, 0.8784, 1.0000, 1.0000,
 0.9373, 0.0235, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0941, 0.9451, 1.0000,
 0.8353, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0196, 0.5882, 0.2745, 0.4039, 1.0000, 1.0000,
 0.6588, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.1373, 0.8706, 1.0000, 0.6824,
 0.0824, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.2549, 1.0000, 0.9961,
 0.2431, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.2314, 0.8902, 0.9647, 0.3647, 0.0078,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1176, 0.9922, 0.7294,
 0.0078, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0941, 0.9412, 0.9412, 0.2980, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0314, 0.5647, 0.1020,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.4784, 1.0000, 0.3529, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000]]]), 8)
```

```
plt.imshow(image.squeeze());
```




```
class_names = train_data.classes
class_names
```

```
['o', 'ki', 'su', 'tsu', 'na', 'ha', 'ma', 'ya', 're', 'wo']
```

```
# Creating DataLoader object
BATCH_SIZE = 2048 # Adjust the batch size based on GPU RAM

TrainLoader = DataLoader(train_data, BATCH_SIZE, shuffle=True)
TestLoader = DataLoader(test_data, BATCH_SIZE, shuffle=False)
```

```
print(f"Dataloaders: {TrainLoader, TestLoader}")
print(f"Length of train dataloader: {len(TrainLoader)} batches of {BATCH_SIZE}")
print(f"Length of test dataloader: {len(TestLoader)} batches of {BATCH_SIZE}")
```

```
Dataloaders: (<torch.utils.data.dataloader.DataLoader object at 0x7f5c7d21a700>,
<torch.utils.data.dataloader.DataLoader object at 0x7f5c7d21a9d0>)
Length of train dataloader: 30 batches of 2048
Length of test dataloader: 5 batches of 2048
```

## 3. Creating the model

Here, we will do an unusual thing: creating a model from scratch. Why unusual? Normally, you would want to start with a pre-trained model with pre-trained weights and only adapt the parts (normally the “head” - the last layer) of the model to the task at hand. We do not exactly know how the model really “learn”, but there is a good chance that the early layers of the model pick up patterns that are also needed for our task. Take a look at the visualizations from this paper: the early layers seem to pick up on edges, curves, and then circles, etc. which are kind of “universal patterns” in anything that you want to classify or generate. But well, I am getting ahead of myself, back to the model.

Now, since this is routine, I will create the most basic neural network model: fully-connected neural networks with 3 hidden layers.

```
class BasicModel(nn.Module):
    def __init__(self, input_shape: int, output_shape: int):
        super().__init__()
        self.classifier = nn.Sequential(
            nn.Flatten(), # Flatten the image out
            nn.Linear(in_features=input_shape, out_features=256),
            nn.ReLU(),
            nn.Linear(in_features=256, out_features=128),
            nn.ReLU(),
            nn.Linear(in_features=128, out_features=64),
            nn.ReLU(),
            nn.Linear(in_features=64, out_features=32),
            nn.ReLU(),
            nn.Linear(in_features=32, out_features=output_shape),
            nn.ReLU()
        )

    def forward(self, x: torch.Tensor):
        return self.classifier(x)
```

For the problem, we want to set `input_shape = 784`, as the dimensions for each image are 28*28,

and `output_shape = len(class_names)`. This is a multiclass classification problem, so technically we will need to help a `softmax` last layer. However, it is better practice to output raw results (logits) and pass them into `softmax` later (the reason has to do with precision - go Google and Wikipedia, or probably just somewhere in the PyTorch documentation).

```
# Instantiate the model. Set the seed for reproducibility
torch.manual_seed(17)

model = BasicModel(input_shape = 784,
                   output_shape = len(class_names)).to(device)
next(model.parameters()).device
```

```
device(type='cuda', index=0)
```

## 4. Pick the loss function (criterion), optimizer, and metric

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters(),
                             lr=0.1)
accuracy = Accuracy(task='multiclass', num_classes=len(class_names)).to(device)
```

## 5. Training

Training a model in PyTorch follows a pretty standard procedure. So much so that we can functionize the train and test step!

```
def train_step(model: torch.nn.Module,
               data_loader: torch.utils.data.DataLoader,
               criterion: torch.nn.Module,
               optimizer: torch.optim.Optimizer,
               metric: Accuracy,
               device: torch.device = device):
    train_loss, train_acc = 0, 0
    for batch, (X,y) in enumerate(data_loader):
        X, y = X.to(device), y.to(device)
        # 1. Forward pass
        y_pred = model(X)

        # 2. Calculate loss & accuracy
        loss = criterion(y_pred, y)
        train_loss += loss
        train_acc += metric(y_pred.argmax(dim=1), y)

        # 3. Empty out gradient
        optimizer.zero_grad()

        # 4. Backpropagation
        loss.backward()
```

```
        # 5. Optimize 1 step
        optimizer.step()

    train_loss /= len(data_loader)
    train_acc /= len(data_loader)
    print(f"Train loss: {train_loss:.5f} | Train accuracy: {train_acc:.2f}")
```

```
def test_step(model: torch.nn.Module,
              data_loader: torch.utils.data.DataLoader,
              criterion: torch.nn.Module,
              metric: Accuracy,
              device: torch.device = device):
    test_loss, acc = 0, 0
    model.eval()
    with torch.inference_mode():
        for (X,y) in data_loader:
            X, y = X.to(device), y.to(device)
        # 1. Forward pass
            y_pred = model(X)

        # 2. Calculate loss & accuracy
            test_loss += criterion(y_pred, y)
            acc += metric(y_pred.argmax(dim=1), y)

        test_loss /= len(data_loader)
        acc /= len(data_loader)
        print(f"Test loss: {test_loss:.5f} | Test accuracy: {acc:.2f}")
```

```
# Check if function is defined correctly
epochs = 2
for epoch in tqdm(range(epochs)):
    print(f"Epoch: {epoch}\n---------")
    train_step(data_loader=TrainLoader,
        model=model,
        criterion=criterion,
        optimizer=optimizer,
        metric=accuracy,
        device=device
    )
    test_step(data_loader=TestLoader,
        model=model,
        criterion=criterion,
        metric=accuracy,
        device=device
    )
```

```
Epoch: 0
---------
Train loss: 8.50348 | Train accuracy: 0.15
Test loss: 2.21395 | Test accuracy: 0.17
Epoch: 1
```

```
---------
Train loss: 2.06660 | Train accuracy: 0.22
Test loss: 2.18012 | Test accuracy: 0.18
```

```python
# Go big!
epochs = 12
for epoch in tqdm(range(epochs)):
    print(f"Epoch: {epoch}\n---------")
    train_step(data_loader=TrainLoader,
        model=model,
        criterion=criterion,
        optimizer=optimizer,
        metric=accuracy,
        device=device
    )
    test_step(data_loader=TestLoader,
        model=model,
        criterion=criterion,
        metric=accuracy,
        device=device
    )
```

```
Epoch: 0
---------
Train loss: 2.04674 | Train accuracy: 0.26
Test loss: 2.33638 | Test accuracy: 0.14
Epoch: 1
---------
Train loss: 2.09693 | Train accuracy: 0.25
Test loss: 2.43485 | Test accuracy: 0.20
Epoch: 2
---------
Train loss: 2.07174 | Train accuracy: 0.25
Test loss: 2.11362 | Test accuracy: 0.20
Epoch: 3
---------
Train loss: 1.99235 | Train accuracy: 0.26
Test loss: 2.08018 | Test accuracy: 0.23
Epoch: 4
---------
Train loss: 1.96704 | Train accuracy: 0.27
Test loss: 2.10831 | Test accuracy: 0.24
Epoch: 5
---------
Train loss: 2.78250 | Train accuracy: 0.26
Test loss: 13.34397 | Test accuracy: 0.17
Epoch: 6
---------
Train loss: 2.60214 | Train accuracy: 0.12
Test loss: 2.30258 | Test accuracy: 0.10
Epoch: 7
---------
Train loss: 2.30258 | Train accuracy: 0.10
```

```
Test loss: 2.30258 | Test accuracy: 0.10
Epoch: 8
---------
Train loss: 2.30258 | Train accuracy: 0.10
Test loss: 2.30258 | Test accuracy: 0.10
Epoch: 9
---------
Train loss: 2.30258 | Train accuracy: 0.10
Test loss: 2.30258 | Test accuracy: 0.10
Epoch: 10
---------
Train loss: 2.30258 | Train accuracy: 0.10
Test loss: 2.30258 | Test accuracy: 0.10
Epoch: 11
---------
Train loss: 2.30258 | Train accuracy: 0.10
Test loss: 2.30258 | Test accuracy: 0.10
```

Uh oh, look like our model only peaked at roughly 20% accuracy and then regressed to random guesses! This is clearly overfitting! I will run it again back down to see if things get better with a smaller learning rate.

But before that, let's use a little routine to empty out GPU RAM.

```
gc.collect()
torch.cuda.empty_cache()
```

```
model1 = BasicModel(input_shape = 784,
                    output_shape = len(class_names)).to(device)
optimizer = torch.optim.Adam(params=model1.parameters(),
                             lr=0.01)
```

```
epochs = 12
for epoch in tqdm(range(epochs)):
    print(f"Epoch: {epoch}\n---------")
    train_step(data_loader=TrainLoader,
        model=model1,
        criterion=criterion,
        optimizer=optimizer,
        metric=accuracy,
        device=device
    )
    test_step(data_loader=TestLoader,
        model=model1,
        criterion=criterion,
        metric=accuracy,
        device=device
    )
```

```
Epoch: 0
---------
Train loss: 1.71525 | Train accuracy: 0.40
```

```
Train loss: 1.71929 | Train accuracy: 0.40
Test loss: 1.64009 | Test accuracy: 0.42
Epoch: 1
---------
Train loss: 1.21122 | Train accuracy: 0.54
Test loss: 1.41001 | Test accuracy: 0.47
Epoch: 2
---------
Train loss: 1.08863 | Train accuracy: 0.57
Test loss: 1.31356 | Test accuracy: 0.50
Epoch: 3
---------
Train loss: 1.04061 | Train accuracy: 0.58
Test loss: 1.25006 | Test accuracy: 0.53
Epoch: 4
---------
Train loss: 1.01216 | Train accuracy: 0.58
Test loss: 1.23056 | Test accuracy: 0.52
Epoch: 5
---------
Train loss: 0.98900 | Train accuracy: 0.59
Test loss: 1.24966 | Test accuracy: 0.54
Epoch: 6
---------
Train loss: 0.97788 | Train accuracy: 0.59
Test loss: 1.29083 | Test accuracy: 0.54
Epoch: 7
---------
Train loss: 0.97544 | Train accuracy: 0.59
Test loss: 1.21861 | Test accuracy: 0.54
Epoch: 8
---------
Train loss: 0.96460 | Train accuracy: 0.59
Test loss: 1.25219 | Test accuracy: 0.54
Epoch: 9
---------
Train loss: 0.96443 | Train accuracy: 0.59
Test loss: 1.24812 | Test accuracy: 0.54
Epoch: 10
---------
Train loss: 0.95962 | Train accuracy: 0.59
Test loss: 1.25274 | Test accuracy: 0.54
Epoch: 11
---------
Train loss: 0.95726 | Train accuracy: 0.59
Test loss: 1.24894 | Test accuracy: 0.54
```

Better results, but our model still hits a plateau at 59% training accuracy and 54% testing accuracy, and if you check the test loss, you will see it fluctuate around 1.25... something. Let's try again with a different optimizer: `SGD`

```
gc.collect()
torch.cuda.empty_cache()
```

```
model2 = BasicModel(input_shape = 784,
                    output_shape = len(class_names)).to(device)
optimizer = torch.optim.SGD(params=model1.parameters(),
                            lr=0.01)
```

```
epochs = 12
for epoch in tqdm(range(epochs)):
    print(f"Epoch: {epoch}\n---------")
    train_step(data_loader=TrainLoader,
        model=model2,
        criterion=criterion,
        optimizer=optimizer,
        metric=accuracy,
        device=device
    )
    test_step(data_loader=TestLoader,
        model=model2,
        criterion=criterion,
        metric=accuracy,
        device=device
    )
```

```
Epoch: 0
---------
Train loss: 2.30361 | Train accuracy: 0.10
Test loss: 2.30377 | Test accuracy: 0.10
Epoch: 1
---------
Train loss: 2.30357 | Train accuracy: 0.10
Test loss: 2.30377 | Test accuracy: 0.10
Epoch: 2
---------
Train loss: 2.30359 | Train accuracy: 0.10
Test loss: 2.30377 | Test accuracy: 0.10
Epoch: 3
---------
Train loss: 2.30355 | Train accuracy: 0.10
Test loss: 2.30377 | Test accuracy: 0.10
Epoch: 4
---------
Train loss: 2.30364 | Train accuracy: 0.10
Test loss: 2.30377 | Test accuracy: 0.10
Epoch: 5
---------
Train loss: 2.30356 | Train accuracy: 0.10
Test loss: 2.30377 | Test accuracy: 0.10
Epoch: 6
---------
Train loss: 2.30352 | Train accuracy: 0.10
Test loss: 2.30377 | Test accuracy: 0.10
Epoch: 7
```

```
---------
Train loss: 2.30354 | Train accuracy: 0.10
Test loss: 2.30377 | Test accuracy: 0.10
Epoch: 8
---------
Train loss: 2.30358 | Train accuracy: 0.10
Test loss: 2.30377 | Test accuracy: 0.10
Epoch: 9
---------
Train loss: 2.30362 | Train accuracy: 0.10
Test loss: 2.30377 | Test accuracy: 0.10
Epoch: 10
---------
Train loss: 2.30355 | Train accuracy: 0.10
Test loss: 2.30377 | Test accuracy: 0.10
Epoch: 11
---------
Train loss: 2.30360 | Train accuracy: 0.10
Test loss: 2.30377 | Test accuracy: 0.10
```

Okay, disappointing result. But well, that is the spirit of hyperparameter tuning - experimenting to see what is the best set-up! But this is getting too long, so I will stop here. Now, `model1` seems to be the best choice so far, so let's plot a confusion matrix for it.

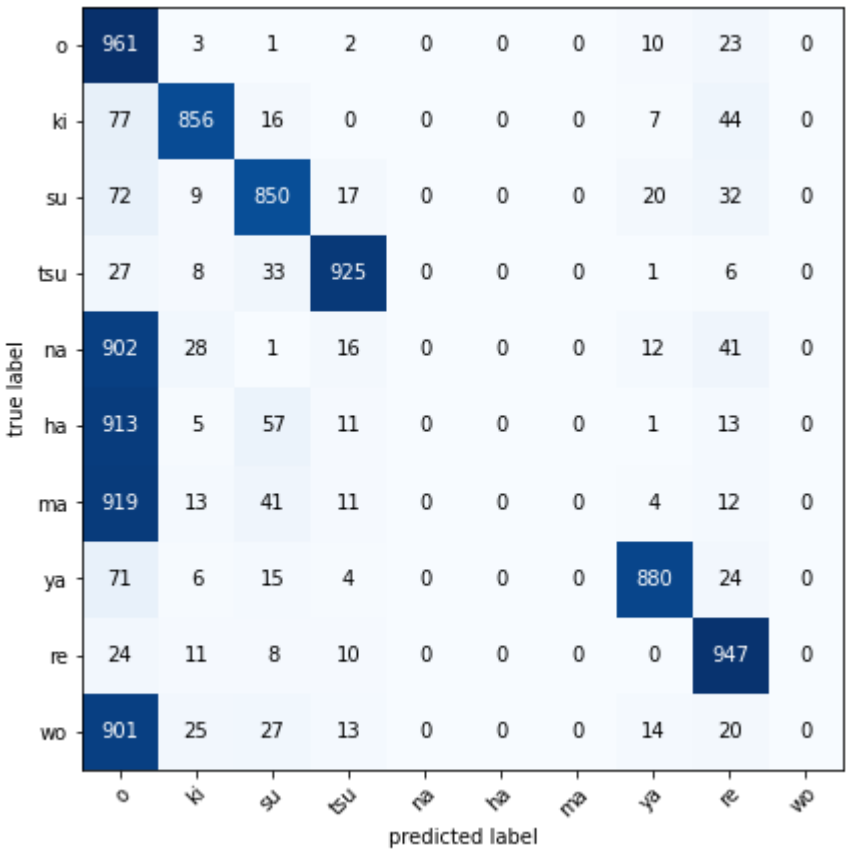
## 6. Post-training analysis

```
# 1. Making predictions with the trained model
y_preds = []
model1.eval()
with torch.inference_mode():
  for X, y in tqdm(TestLoader, desc="Making predictions"):
    # Send data and targets to target device
    X, y = X.to(device), y.to(device)
    # Do the forward pass
    y_logit = model1(X)
    # Turn predictions from logits -> prediction probabilities -> predictions labels
    y_pred = torch.softmax(y_logit, dim=1).argmax(dim=1)
    # Put predictions on CPU for evaluation
    y_preds.append(y_pred.cpu())
# Concatenate list of predictions into a tensor
y_pred_tensor = torch.cat(y_preds)
```

```
# 2. Setup confusion matrix instance and compare predictions to targets
confmat = ConfusionMatrix(num_classes=len(class_names), task='multiclass')
confmat_tensor = confmat(preds=y_pred_tensor,
                         target=test_data.targets)

# 3. Plot the confusion matrix
fig, ax = plot_confusion_matrix(
    conf_mat=confmat_tensor.numpy(), # matplotlib likes working with NumPy
    class_names = class_names  # turn the row and column labels into class names
```

```
    figsize=(10, 7)
);
```




I am not familiar with Kanji characters, but I think we should take a look at the ‘na’, ‘ha’, ‘ma’, ‘wo’ and ‘o’ characters, which are really alike in the pictures, and think of ways to help our model. However, that is outside the scope of this routine notebook, so I will stop here. Hope your, reader, is familiar with the PyTorch routine by now, and thank you for reading!