```
#from google.colab import drive
#drive.mount('/content/drive')
```

```
     Mounted at /content/drive
```

```
%cd /content/drive/MyDrive/youtube_tutorial
#!unzip test.csv.zip
#!unzip train.csv.zip
```

```
     /content/drive/MyDrive/youtube_tutorial
     Archive:  test.csv.zip
       inflating: test.csv
     Archive:  train.csv.zip
       inflating: train.csv
```

```
  import torch
  from torch import nn
  import pandas as pd
  from torch import optim
  from torch.utils.data import DataLoader, Dataset
  from torchvision import transforms
  from sklearn.model_selection import train_test_split
  import matplotlib.pyplot as plt
  import numpy as np
  import random
  import timeit
  from tqdm import tqdm
```

```
RANDOM_SEED=42
BATCH_SIZE=512
EPOCHS=40
#Classification from learnable token
LEARNING_RATE=1e-4
NUM_CLASS=10
PATCH_SIZE=4 # One size of a length of square size (4*4), like a respective field in CNN
IMAGE_SIZE=28 #Image is 28*28
IN_CHANNELS=1
NUM_HEADS=8
DROPOUT=0.001
HIDDEN_DIM=768
ADAM_WEIGHT_DECAY=0
ADAM_BETAS=(0.9,0.999)
ACTIVATION="gelu"
NUM_ENCODERS=4 #Stack 4 different encoder
EMBED_DIM=(PATCH_SIZE**2)*IN_CHANNELS #16
NUM_PATCHES=(IMAGE_SIZE//PATCH_SIZE)**2 #49

random.seed(RANDOM_SEED)
np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)
torch.cuda.manual_seed(RANDOM_SEED)
torch.cuda.manual_seed_all(RANDOM_SEED)
torch.backends.cudnn.deterministic=True
torch.backends.cudnn.benchmark=False

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

```
class PatchEmbedding(nn.Module):
        def __init__(self, embed_dim, patch_size, num_patches, dropout, in_channels):
                super().__init__()
                #Init x=(512,1,28,28)
                self.patcher = nn.Sequential(
                        nn.Conv2d(
                                in_channels=in_channels, #1
                                out_channels=embed_dim, #16
                                kernel_size=patch_size, #4
                                stride=patch_size, #4
                        ),
                        #As stride=4, the shape become (512,16,7,7)
                        nn.Flatten(2))
                        # Keep the first 2 dimension of the data, then concat all of them after it
                        #(512,16,7*7)

                self.cls_token = nn.Parameter(torch.randn(size=(1, in_channels, embed_dim)), requires_grad=True) #Classification token, learnable
                #CLS stands for Classification token, torch.randn creates a tensor filled with random number (normal distribution with mea
                #indicate it is a part of gradient descent
                #size (1 (Indicate it is a single classification tasks), in_channels (Indicate the image is grayscale), embed_dim stands

                self.position_embeddings = nn.Parameter(torch.randn(size=(1, num_patches+1, embed_dim)), requires_grad=True)
                #size=(1(Indicate it is a single classification tasks, number of total patches+1, number of input embedding ))
                self.dropout = nn.Dropout(p=dropout)

        def forward(self, x):
                cls_token = self.cls_token.expand(x.shape[0], -1, -1) #Classification token

                x = self.patcher(x).permute(0, 2, 1)
                #x (512,49,16)
                x = torch.cat([cls_token, x], dim=1)
                x = self.position_embeddings + x
                x = self.dropout(x)
                return x

model = PatchEmbedding(EMBED_DIM, PATCH_SIZE, NUM_PATCHES, DROPOUT, IN_CHANNELS).to(device)
x = torch.randn(512, 1, 28, 28).to(device) #(Batch size, number of Channel, Image dimension x, Image dimension y)
print(model(x).shape)
```

```
torch.Size([512, 50, 16])
```

```
class ViT(nn.Module):
    def __init__(self, num_patches, img_size, num_classes, patch_size, embed_dim, num_encoders, num_heads, hidden_dim, dropout, activation, ir
        super().__init__()
        self.embeddings_block=PatchEmbedding(embed_dim, patch_size, num_patches, dropout, in_channels)

        encoder_layer=nn.TransformerEncoderLayer(d_model=embed_dim, nhead=num_heads, dropout=dropout, activation=activation, batch_first=True, norm_f
        self.encoder_block=nn.TransformerEncoder(encoder_layer, num_layers=num_encoders)

        self.mlp_head=nn.Sequential(
            nn.LayerNorm(normalized_shape=embed_dim),
            nn.Linear(in_features=embed_dim, out_features=num_classes)

        )

    def forward(self,x):
        x=self.embeddings_block(x)
        x=self.encoder_block(x)
        x=self.mlp_head(x[:,0,:])
        return x

model=ViT(NUM_PATCHES, IMAGE_SIZE, NUM_CLASS, PATCH_SIZE, EMBED_DIM, NUM_ENCODERS, NUM_HEADS, HIDDEN_DIM, DROPOUT, ACTIVATION, IN_CHANNELS).to(c

#x=torch.randn(512,1,28,28).to(device)
#print(model(x).shape)
```

```
torch.Size([512, 10])
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/transformer.py:282: UserWarning: enable_nested_tensor is True, but self.use_nested_tensor is
  warnings.warn(f"enable_nested_tensor is True, but self.use_nested_tensor is False because {why_not_sparsity_fast_path}")
```

```
train_df=pd.read_csv("train.csv")
test_df=pd.read_csv('test.csv')
submission_df=pd.read_csv('sample_submission.csv')
```

按兩下 (或按 Enter 鍵) 即可編輯

```
train_df.head()
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pi
0	1	0	0	0	0	0	0	0	0	0	...	
1	0	0	0	0	0	0	0	0	0	0	...	
2	1	0	0	0	0	0	0	0	0	0	...	
3	4	0	0	0	0	0	0	0	0	0	...	
4	0	0	0	0	0	0	0	0	0	0	...	

5 rows × 785 columns

```
test_df.head()
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	p
0	0	0	0	0	0	0	0	0	0	0	...	
1	0	0	0	0	0	0	0	0	0	0	...	
2	0	0	0	0	0	0	0	0	0	0	...	
3	0	0	0	0	0	0	0	0	0	0	...	
4	0	0	0	0	0	0	0	0	0	0	...	

5 rows × 784 columns

```
submission_df.head()
```

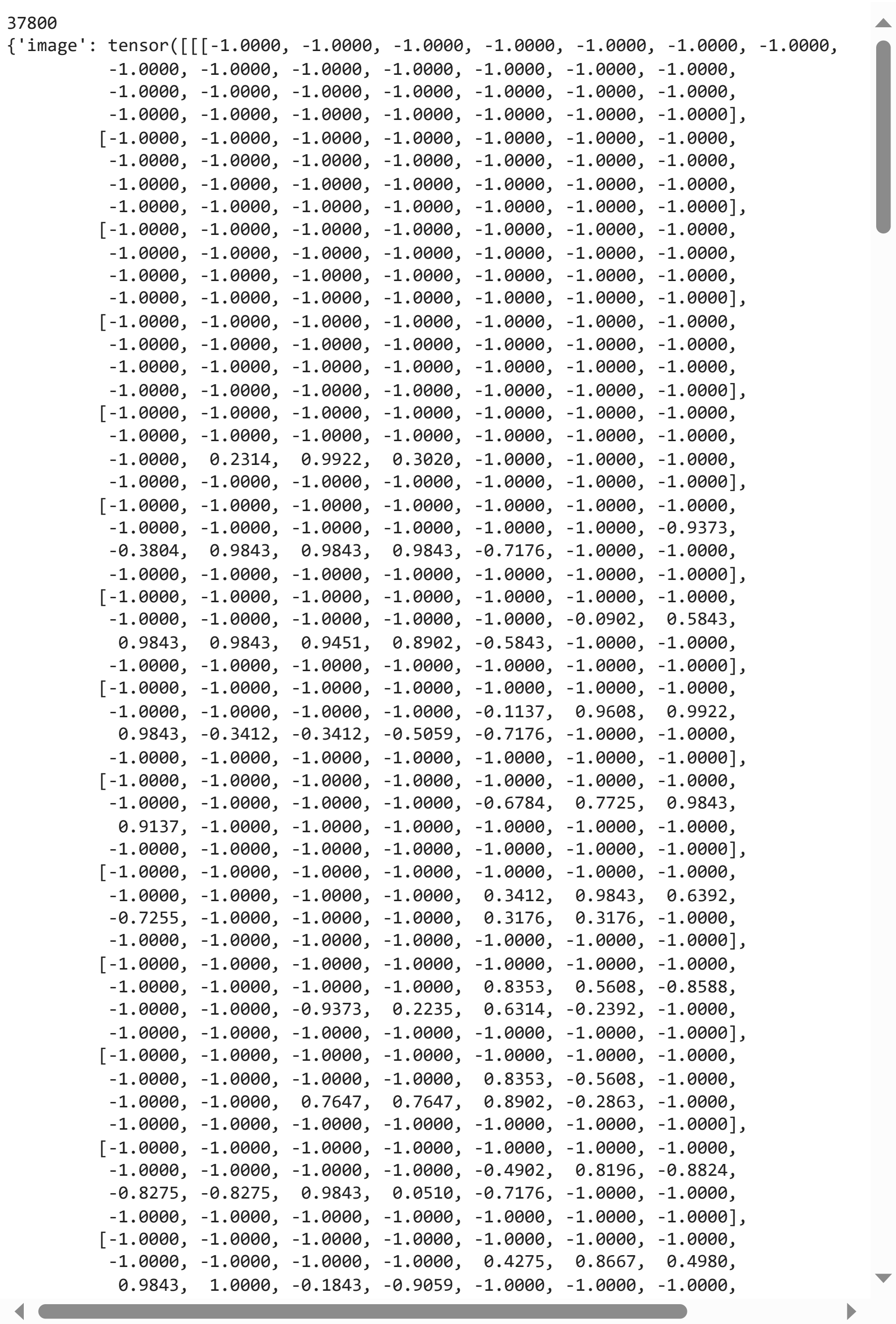
	ImageId	Label
0	1	0
1	2	0
2	3	0
3	4	0
4	5	0

Next steps: View recommended plots

```
train_df,  val_df  =  train_test_split(train_df,  test_size=0.1,  random_state=RANDOM_SEED,  shuffle=True)
```

```
class MNISTTrainDataset(Dataset):
        def __init__(self, images, labels, indicies):
                self.images = images
                self.labels = labels
                self.indicies = indicies
                self.transform = transforms.Compose([
                        transforms.ToPILImage(),
                        transforms.RandomRotation(15),
                        transforms.ToTensor(),
                        transforms.Normalize([0.5], [0.5])
                ])

        def __len__(self):
                return len(self.images)

        def __getitem__(self, idx):
                image = self.images[idx].reshape((28, 28)).astype(np.uint8)
                label = self.labels[idx]
                index = self.indicies[idx]
                image = self.transform(image)

                return {"image": image, "label": label, "index": index}

class MNISTValDataset(Dataset):
        def __init__(self, images, labels, indicies):
                self.images = images
                self.labels = labels
                self.indicies = indicies
                self.transform = transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize([0.5], [0.5])
                ])

        def __len__(self):
                return len(self.images)

        def __getitem__(self, idx):
                image = self.images[idx].reshape((28, 28)).astype(np.uint8)
                label = self.labels[idx]
                index = self.indicies[idx]
                image = self.transform(image)

                return {"image": image, "label": label, "index": index}

class MNISTSubmitDataset(Dataset):
        def __init__(self, images, indicies):
                self.images = images
                self.indicies = indicies
                self.transform = transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize([0.5], [0.5])
                ])

        def __len__(self):
                return len(self.images)

        def __getitem__(self, idx):
                image = self.images[idx].reshape((28, 28)).astype(np.uint8)
                index = self.indicies[idx]
                image = self.transform(image)

                return {"image": image, "index": index}


plt.figure()
f, axarr = plt.subplots(1, 3)

train_dataset = MNISTTrainDataset(train_df.iloc[:, 1:].values.astype(np.uint8), train_df.iloc[:, 0].values, train_df.index.values)
print(len(train_dataset))
print(train_dataset[0])
axarr[0].imshow(train_dataset[0]["image"].squeeze(), cmap="gray")
axarr[0].set_title("Train Image")
print("-"*30)

val_dataset = MNISTValDataset(val_df.iloc[:, 1:].values.astype(np.uint8), val_df.iloc[:, 0].values, val_df.index.values)
print(len(val_dataset))
print(val_dataset[0])
axarr[1].imshow(val_dataset[0]["image"].squeeze(), cmap="gray")
```

```
axarr[1].set_title("Val Image")
print("-"*30)

test_dataset = MNISTSubmitDataset(test_df.values.astype(np.uint8), test_df.index.values)
print(len(test_dataset))
print(test_dataset[0])
axarr[2].imshow(test_dataset[0]["image"].squeeze(), cmap="gray")
axarr[2].set_title("Test Image")
print("-"*30)

plt.show()
```



```
37800
{'image': tensor([[[-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000,  0.2314,  0.9922,  0.3020, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -0.9373,
          -0.3804,  0.9843,  0.9843,  0.9843, -0.7176, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -0.0902,  0.5843,
           0.9843,  0.9843,  0.9451,  0.8902, -0.5843, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -0.1137,  0.9608,  0.9922,
           0.9843, -0.3412, -0.3412, -0.5059, -0.7176, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -0.6784,  0.7725,  0.9843,
           0.9137, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000,  0.3412,  0.9843,  0.6392,
          -0.7255, -1.0000, -1.0000, -1.0000,  0.3176,  0.3176, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000,  0.8353,  0.5608, -0.8588,
          -1.0000, -1.0000, -0.9373,  0.2235,  0.6314, -0.2392, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000,  0.8353, -0.5608, -1.0000,
          -1.0000, -1.0000,  0.7647,  0.7647,  0.8902, -0.2863, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -0.4902,  0.8196, -0.8824,
          -0.8275, -0.8275,  0.9843,  0.0510, -0.7176, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000],
         [-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
          -1.0000, -1.0000, -1.0000, -1.0000,  0.4275,  0.8667,  0.4980,
           0.9843,  1.0000, -0.1843, -0.9059, -1.0000, -1.0000, -1.0000,
```