- apple machine learning research
  tesla

pytorch computervision

```
import torch ... from torch import nn ... import torchvision ... from torchvision import datasets
from torchvision.transforms import ToTensor ... import matplotlib.pyplot as plt   # for visualization
```

```
train_data = datasets.FashionMNIST(root="data", train=True, download=True, transform=ToTensor(),
    target_transform=None)
test_data = datasets.FashionMNIST(root="data", train=False, download=True, transform=ToTensor())
```

setup training data

```
from torch.utils.data import DataLoader
BATCH_SIZE = 32
train_dataloader = DataLoader(train_data, batch_size=BATCH_SIZE, shuffle=True)
test_dataloader =                test_data                          False
```

turn dataset into batches

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

device agnostic code

```
from torch import nn ... from tqdm.auto import tqdm ... import pandas as pd
from helper_functions import accuracy_fn
```

some redundancy

```
class FashionMNISTModelV2(nn.Module):
    def __init__(self, input_shape: int, hidden_units: int, output_shape: int):
        super().__init__()
        self.block_1 = nn.Sequential(
            nn.Conv2d(in_channels=input_shape, out_channels=hidden_units, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(                                                                    ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.block_2 = nn.Sequential(
            nn.Conv2d(hidden_units, hidden_units, 3, padding=1),
            nn.ReLU(),
            nn.Conv2d(                                          ),
            nn.ReLU(),
            nn.MaxPool2d(2))
```

CNN Explainer

```
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(in_features= hidden_units*7*7, out_features = output_shape))
    def forward(self, x: torch.Tensor):
        x = self.block_1(x) ... x = self.block_2(x) ... x = self.classifier(x)
        return x

torch.manual_seed(42)
model_2 = FashionMNISTModelV2(input_shape=1, hidden_units=10, output_shape= len(class_names)).to(device)

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(params=model_2.parameters(), lr=0.1)

epochs = 3
for epoch in tqdm(range(epochs)):
    print(f"Epoch: {epoch}\n-----")
    train_step(data_loader= train_data_loader, model= model_2, loss_fn= loss_fn, optimizer =optimizer,
        accuracy_fn= accuracy_fn, device= device)
    test_step(data_loader= test_data_loader, model= model_2, loss_fn= loss_fn,
        accuracy_fn= accuracy_fn, device=device)

model_2_results = eval_model(model= model_2, data_loader= test_data_loader, loss_fn = loss_fn,
    accuracy_fn =accuracy_fn)

# make and evaluate random predictions
# make confusion matrix for further prediction evaluation
from pathlib import Path
MODEL_PATH = Path("models")
MODEL_PATH.mkdir(parents= True, exist_ok= True)
MODEL_NAME = "03_pytorch_computer_vision_model_2.pth"
MODEL_SAVE_PATH = MODEL_PATH/ MODEL_NAME
torch.save(obj= model_2.state_dict(), f= MODEL_SAVE_PATH)
# load and test saved model
```

train and test model

get model results

create model directory

create model save path

save model state dict

pytorch computervision continued... confusion matrix

```
# vv make confusion matrix for further prediction evaluation vv
from tqdm.auto import tqdm
y_preds = []
model_2.eval()
with torch.inference_mode():
  for X, y in tqdm(test_dataloader, desc="Making predictions"):
    X, y = X.to(device), y.to(device)
    y_logit = model_2(X)                                   ] forward pass
    y_pred = torch.softmax(y_logit, dim=1).argmax(dim=1)   ] logits -> prediction probabilities -> prediction labels
    y_preds.append(y_pred.cpu())                           ] put predictions on CPU for evaluation
y_preds_tensor = torch.cat(y_preds)
```

make predictions with trained model

```
(might need) !pip install -q torchmetrics -U mlxtend
import torchmetrics, mlxtend
from torchmetrics import ConfusionMatrix
from mlxtend.plotting import plot_confusion_matrix
confmat = ConfusionMatrix(num_classes=len(class_names), task='multiclass')
confmat_tensor = confmat(preds=y_pred_tensor, target=test_data.targets)
fig, ax = plot_confusion_matrix(conf_mat=confmat_tensor.numpy(), class_names=class_names,
  figsize=(10,7));
```