Front Matter

I want the Notebook to be as informative as possible, but model creating and training process follows some standard procedure that I do not want to repeat. Therefore, if you can, spend time reading the `PROLOGUE/Routine.ipynb` Notebook first.

Paper Implementation - VGG16

Continuing from the first notebook, now we shall define and train the model on the data!

In the notebook I will do this in three ways: we create a model from scratch, based on the architecture in the paper, with a slight different for the output. Next, we will try *transfer learning*, where we load a VGG16 model that has already been trained and try to adapt it to our problem. Typically, with transfer learning, you will want to *freeze* the parameters of the feature layers (prevent them from being updated during training) and only update the parameters of the output layer (also called head or classifier layer, for our particular problem).

1. Model from scratch

I use idea for a general implementation of all VGG models from Aladdin Persson.

0. Installing on Colab

In [ ]:

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

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheel
s/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 32.2 MB/s
Requirement already satisfied: scipy>=1.2.1 in /usr/local/lib/python3.8/dist-packa
ges (from mlxtend) (1.7.3)
Requirement already satisfied: numpy>=1.16.2 in /usr/local/lib/python3.8/dist-pack
ages (from mlxtend) (1.21.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-package
s (from mlxtend) (57.4.0)
Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.8/dist-pac
kages (from mlxtend) (1.2.0)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.8/dis
t-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: pandas>=0.24.2 in /usr/local/lib/python3.8/dist-pac
kages (from mlxtend) (1.3.5)
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: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/lo
cal/lib/python3.8/dist-packages (from matplotlib>=3.0.0->mlxtend) (3.0.9)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8/di
st-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packa
```

```
ges (from matplotlib>=3.0.0->mlxtend) (0.11.0)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packa
ges (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/di
st-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
```

In [ ]:

```
!pip install torchmetrics
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheel
s/public/simple/
Collecting torchmetrics
  Downloading torchmetrics-0.11.0-py3-none-any.whl (512 kB)
     |████████████████████████████████| 512 kB 32.8 MB/s
Requirement already satisfied: torch>=1.8.1 in /usr/local/lib/python3.8/dist-packa
ges (from torchmetrics) (1.13.0+cu116)
Requirement already satisfied: numpy>=1.17.2 in /usr/local/lib/python3.8/dist-pack
ages (from torchmetrics) (1.21.6)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.8/dist-
packages (from torchmetrics) (4.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-packages
(from torchmetrics) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.
8/dist-packages (from packaging->torchmetrics) (3.0.9)
Installing collected packages: torchmetrics
Successfully installed torchmetrics-0.11.0
```

1. Import modules

In [ ]:

```
import torch
from torch import nn
from torch.utils.data import DataLoader, random_split, Dataset

import torchvision
from torchvision import datasets, transforms

from torchmetrics import ConfusionMatrix, Accuracy
from mlxtend.plotting import plot_confusion_matrix
from tqdm.auto import tqdm

import matplotlib.pyplot as plt
import requests
import gc
import json
import shutil
import os
from pathlib import Path
```

2. Load data

In [ ]:

```
train_transforms = transforms.Compose([transforms.RandomResizedCrop(224),
                                       transforms.RandomRotation(35),
                                       transforms.RandomVerticalFlip(0.27),
                                       transforms.RandomHorizontalFlip(0.27),
                                       transforms.ToTensor(),
                                       transforms.Normalize([0.485, 0.456, 0.406],

valid_n_test_transforms = transforms.Compose([transforms.Resize((224,224)),
                                       transforms.ToTensor(),
                                       transforms.Normalize([0.485, 0.456, 0.406],
```

In [ ]:

```
# Should be download=True if you do not have the data
train_data = datasets.Food101(root='DATA',
                              download=False)
test_data = datasets.Food101(root='DATA',
                              split='test',
                              transform=valid_n_test_transforms,
                              download=False)
```

In [ ]:

```
print(train_data)
```

```
Dataset Food101
    Number of datapoints: 75750
    Root location: DATA
    split=train
```

In [ ]:

```
print(test_data)
```

```
Dataset Food101
    Number of datapoints: 25250
    Root location: DATA
    split=test
    StandardTransform
Transform: Compose(
               Resize(size=(224, 224), interpolation=bilinear, max_size=None, anti
alias=None)
               ToTensor()
               Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
           )
```

In [ ]:

```
train_data, valid_data = random_split(train_data, [0.7,0.3], generator=torch.Gener
```

In [ ]:

```
# Custom dataset object to load dataset from Subset
class MyDataset(Dataset):
    def __init__(self, subset, transform=None):
        self.subset = subset
        self.transform = transform

    def __getitem__(self, index):
        x, y = self.subset[index]
        if self.transform:
            x = self.transform(x)
        return x, y

    def __len__(self):
        return len(self.subset)
```

In [ ]:

```
train_data = MyDataset(train_data, transform=train_transforms)
valid_data = MyDataset(valid_data, transform=valid_n_test_transforms)
```

In [ ]:

```
print(len(train_data), train_data.transform)
```

```
53025 Compose(
    RandomResizedCrop(size=(224, 224), scale=(0.08, 1.0), ratio=(0.75, 1.3333), in
terpolation=bilinear), antialias=None)
    RandomRotation(degrees=[-35.0, 35.0], interpolation=nearest, expand=False, fil
l=0)
    RandomVerticalFlip(p=0.27)
    RandomHorizontalFlip(p=0.27)
    ToTensor()
    Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
)
```

In [ ]:

```
print(len(valid_data), valid_data.transform)
```

```
22725 Compose(
    Resize(size=(224, 224), interpolation=bilinear, max_size=None, antialias=None)
    ToTensor()
    Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
)
```

In [ ]:

```
print(test_data)
```

```
Dataset Food101
    Number of datapoints: 25250
    Root location: DATA
    split=test
    StandardTransform
Transform: Compose(
               Resize(size=(224, 224), interpolation=bilinear, max_size=None, anti
alias=None)
               ToTensor()
               Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
           )
```

In [ ]:

```
# Note that because of the way I
class_names = test_data.classes
class_names
```

```
Out[ ]: ['apple_pie',
         'baby_back_ribs',
         'baklava',
         'beef_carpaccio',
         'beef_tartare',
         'beet_salad',
         'beignets',
         'bibimbap',
         'bread_pudding',
         'breakfast_burrito',
         'bruschetta',
         'caesar_salad',
         'cannoli',
         'caprese_salad',
         'carrot_cake',
         'ceviche',
         'cheese_plate',
         'cheesecake',
         'chicken_curry',
         'chicken_quesadilla',
         'chicken_wings',
         'chocolate_cake',
         'chocolate_mousse',
         'churros',
         'clam_chowder',
         'club_sandwich',
         'crab_cakes',
         'creme_brulee',
         'croque_madame',
         'cup_cakes',
         'deviled_eggs',
         'donuts',
         'dumplings',
         'edamame',
         'eggs_benedict',
         'escargots',
         'falafel',
         'filet_mignon',
         'fish_and_chips',
         'foie_gras',
         'french_fries',
         'french_onion_soup',
         'french_toast',
         'fried_calamari',
         'fried_rice',
         'frozen_yogurt',
         'garlic_bread',
         'gnocchi',
         'greek_salad',
         'grilled_cheese_sandwich',
         'grilled_salmon',
         'guacamole',
         'gyoza',
         'hamburger',
         'hot_and_sour_soup',
         'hot_dog',
         'huevos_rancheros',
         'hummus',
         'ice_cream',
         'lasagna',
         'lobster_bisque',
         'lobster_roll_sandwich',
         'macaroni_and_cheese',
         'macarons',
         'miso_soup',
         'mussels',
         'nachos',
         'omelette',
         'onion_rings',
         'oysters',
         'pad_thai',
         'paella',
         'pancakes',
         'panna_cotta',
         'peking_duck',
         'pho',
         'pizza',
         'pork_chop',
         'poutine',
         'prime_rib',
         'pulled_pork_sandwich',
         'ramen',
         'ravioli',
         'red_velvet_cake',
         'risotto',
         'samosa',
         'sashimi',
         'scallops',
         'seaweed_salad',
         'shrimp_and_grits',
         'spaghetti_bolognese',
         'spaghetti_carbonara',
         'spring_rolls',
         'steak',
         'strawberry_shortcake',
         'sushi',
         'tacos',
         'takoyaki',
         'tiramisu',
         'tuna_tartare',
         'waffles']
```

In [ ]:

```
train_loader = torch.utils.data.DataLoader(train_data, batch_size=16, shuffle=True
valid_loader = torch.utils.data.DataLoader(valid_data, batch_size=16)
test_loader = torch.utils.data.DataLoader(test_data, batch_size=16)
```

In [ ]:

```
# Seed
torch.manual_seed = 17
```

In [ ]:

```
# Device-agnostic code
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
```

```
cuda
```

In [ ]:

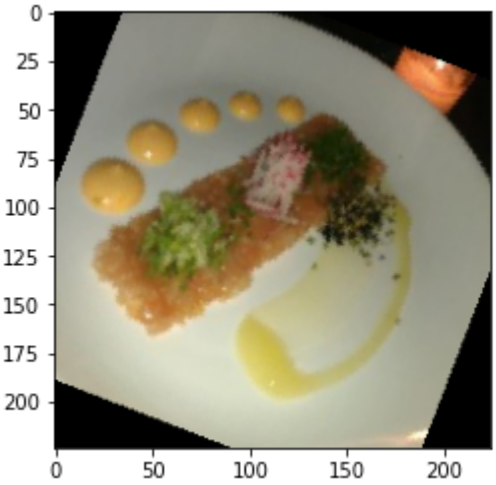
```
if Path("helper_functions.py").is_file():
    print("helper_functions.py already exists, skipping download")
else:
    print("Downloading helper_functions.py")
    request = requests.get("https://raw.githubusercontent.com/HangenYuu/vision_lea
    with open("helper_functions.py", "wb") as f:
        f.write(request.content)

from helper_functions import imshow
```

```
helper_functions.py already exists, skipping download
```

In [ ]:

```
train_features, train_labels = next(iter(train_loader))
img = train_features[0].squeeze()
label = train_labels[0]
imshow(img);
print(f'Image label no.: {label}')
```

```
Image label no.: 99
```



3. Create the model

First, because there are different architectures, we will need a dictionary to contain the architecture a.k.a the number of neurons at each layer.

In [ ]:

```
VGG_TYPES = {
    "VGG11": [64, "M", 128, "M", 256, 256, "M", 512, 512, "M", 512, 512, "M"],
    "VGG13": [64, 64, "M", 128, 128, "M", 256, 256, "M", 512, 512, "M", 512, 512,
    "VGG16": [64, 64, "M", 128, 128, "M", 256, 256, 256, "M", 512, 512, 512, "M",
    "VGG19": [64, 64, "M", 128, 128, "M", 256, 256, 256, 256, "M", 512, 512, 512,
}
```

Let's sneak a peek at the target model from PyTorch:

In [ ]:

```
model = torchvision.models.vgg16()
```

In [ ]:

```
model
```

Out[ ]:

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
```

There is a different between the pre-defined model in PyTorch and the model from the paper: there is an additional average pooling layer after all the convolution layers.

In [ ]:

```
class VGG16_net(nn.Module):
    def __init__(self, in_channels=3, out_classess=101):
        super().__init__()
        self.in_channels = in_channels
        self.conv_layers = self.create_conv_layers(VGG_TYPES["VGG16"])
        self.avgpool = nn.AdaptiveAvgPool2d(output_size=(7,7))
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(512*7*7, 4096), # The 7*7 from the previous pooling layer
            nn.ReLU(), # I do not do it in place
            nn.Dropout(p=0.5),
            nn.Linear(4096,4096),
            nn.ReLU(),
            nn.Dropout(p=0.5),
            nn.Linear(4096, out_classess)
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = self.avgpool(x)
        x = self.classifier(x)
        return x

    def create_conv_layers(self, arch):
        # Stack the layers into a list before passing to nn.Sequential
        layers = []
        in_channels = self.in_channels

        for layer in arch:
            if type(layer) == int:
                out_channels = layer

                layers.extend(
                    [nn.Conv2d(in_channels=in_channels,
                               out_channels=out_channels,
                               kernel_size=3,
                               stride=1,
                               padding=1),
                     nn.BatchNorm2d(layer), # Invented after the original paper, b
                     nn.ReLU()
                     ]

                )
                in_channels = layer
                # Number of output in the last layer is the number
                # of inputs for the next layer
            elif layer=='M':
                layers.append(nn.MaxPool2d(kernel_size=2, stride=2))

        return nn.Sequential(*layers)
```

In [ ]:

```
model1 = VGG16_net().to(device)
model1
```

Out[ ]:

```
VGG16_net(
  (conv_layers): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
    (2): ReLU()
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
    (5): ReLU()
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    (9): ReLU()
    (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    (12): ReLU()
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    (16): ReLU()
    (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    (19): ReLU()
    (20): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    (22): ReLU()
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (24): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    (26): ReLU()
    (27): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (28): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    (29): ReLU()
    (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (31): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    (32): ReLU()
    (33): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (35): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    (36): ReLU()
    (37): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (38): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    (39): ReLU()
    (40): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (41): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    (42): ReLU()
    (43): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=25088, out_features=4096, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU()
    (6): Dropout(p=0.5, inplace=False)
    (7): Linear(in_features=4096, out_features=101, bias=True)
  )
```

In [ ]:

```
next(model1.parameters()).device
```

Out[ ]:

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

4. Pick loss function, optimizer, metric

In [ ]:

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model1.parameters(), lr=0.1)
accuracy = Accuracy(task='multiclass', num_classes=len(class_names)).to(device)
```

In [ ]:

```
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}")
```

In [ ]:

```
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}")
```

In [ ]:

```
del model1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-92-b66c1454c6a6> in <module>
----> 1 del model1

NameError: name 'model1' is not defined
```

In [ ]:

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

In [ ]:

```
epochs = 2
for epoch in tqdm(range(epochs)):
    print(f"Epoch: {epoch}\n---------")
    train_step(data_loader=train_loader,
        model=model1,
        criterion=criterion,
        optimizer=optimizer,
        metric=accuracy,
        device=device
    )
    gc.collect()
    torch.cuda.empty_cache()
    test_step(data_loader=valid_loader,
        model=model1,
        criterion=criterion,
        metric=accuracy,
        device=device
    )
    gc.collect()
    torch.cuda.empty_cache()
```

```
Epoch: 0
---------
Train loss: 4.63904 | Train accuracy: 0.01
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-96-de29cd664eb7> in <module>
     11     gc.collect()
     12     torch.cuda.empty_cache()
---> 13     test_step(data_loader=valid_loader,
     14         model=model1,
     15         criterion=criterion,

<ipython-input-77-d1dbf3faa64a> in test_step(model, data_loader, criterion, metri
c, device)
     14         # 2. Calculate loss & accuracy
     15             test_loss += criterion(y_pred, y)
---> 16             acc += metric(y_pred.argmax(dim=1), y)
     17 
     18         test_loss /= len(data_loader)

/usr/local/lib/python3.8/dist-packages/torch/nn/modules/module.py in _call_impl(se
lf, *input, **kwargs)
   1188         if not (self._backward_hooks or self._forward_hooks or self._forwa
rd_pre_hooks or _global_backward_hooks
   1189                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1190             return forward_call(*input, **kwargs)
   1191         # Do not call functions when jit is used
   1192         full_backward_hooks, non_full_backward_hooks = [], []

/usr/local/lib/python3.8/dist-packages/torchmetrics/metric.py in forward(self, *ar
gs, **kwargs)
    232             self._forward_cache = self._forward_full_state_update(*args, *
*kwargs)
    233         else:
--> 234             self._forward_cache = self._forward_reduce_state_update(*args,
**kwargs)
    235 
    236         return self._forward_cache

/usr/local/lib/python3.8/dist-packages/torchmetrics/metric.py in _forward_reduce_s
tate_update(self, *args, **kwargs)
    298 
    299         # calculate batch state and compute batch value
--> 300         self.update(*args, **kwargs)
    301         batch_val = self.compute()
    302 

/usr/local/lib/python3.8/dist-packages/torchmetrics/metric.py in wrapped_func(*arg
s, **kwargs)
    386             with torch.set_grad_enabled(self._enable_grad):
    387                 try:
--> 388                     update(*args, **kwargs)
    389                 except RuntimeError as err:
    390                     if "Expected all tensors to be on" in str(err):

/usr/local/lib/python3.8/dist-packages/torchmetrics/classification/stat_scores.py
in update(self, preds, target)
    314         """
    315         if self.validate_args:
--> 316             _multiclass_stat_scores_tensor_validation(
    317                 preds, target, self.num_classes, self.multidim_average, se
lf.ignore_index
    318             )

/usr/local/lib/python3.8/dist-packages/torchmetrics/functional/classification/stat
_scores.py in _multiclass_stat_scores_tensor_validation(preds, target, num_classe
s, multidim_average, ignore_index)
    304         )
    305
--> 306     num_unique_values = len(torch.unique(target))
    307     if ignore_index is None:
    308         check = num_unique_values > num_classes

/usr/local/lib/python3.8/dist-packages/torch/_jit_internal.py in fn(*args, **kwarg
s)
    483             return if_true(*args, **kwargs)
    484         else:
--> 485             return if_false(*args, **kwargs)
    486
    487     if if_true.__doc__ is None and if_false.__doc__ is not None:

/usr/local/lib/python3.8/dist-packages/torch/_jit_internal.py in fn(*args, **kwarg
s)
    483             return if_true(*args, **kwargs)
    484         else:
--> 485             return if_false(*args, **kwargs)
    486
    487     if if_true.__doc__ is None and if_false.__doc__ is not None:

/usr/local/lib/python3.8/dist-packages/torch/functional.py in _return_output(inpu
t, sorted, return_inverse, return_counts, dim)
    875         return _unique_impl(input, sorted, return_inverse, return_counts,
dim)
    876
--> 877     output, _, _ = _unique_impl(input, sorted, return_inverse, return_coun
ts, dim)
    878     return output
    879

/usr/local/lib/python3.8/dist-packages/torch/functional.py in _unique_impl(input,
sorted, return_inverse, return_counts, dim)
    789         )
    790     else:
--> 791         output, inverse_indices, counts = torch._unique2(
    792             input,
    793             sorted=sorted,
```

Okay, on Colab, 1 epoch was trained for 15 minutes, with a phenomenal accuracy of 1 percent. This is clearly not what I want.

```
In [ ]:
```