```python
import matplotlib.pyplot as plt
import torch
import torchvision

from torch import nn
from torchvision import transforms
from helper_functions import set_seeds
```

```
/home/ex5/miniconda3/lib/python3.12/site-packages/torch/cuda/
__init__.py:141: UserWarning: CUDA initialization: The NVIDIA driver
on your system is too old (found version 11040). Please update your
GPU driver by downloading and installing a new version from the URL:
http://www.nvidia.com/Download/index.aspx Alternatively, go to:
https://pytorch.org to install a PyTorch version that has been
compiled with your version of the CUDA driver. (Triggered internally
at ../c10/cuda/CUDAFunctions.cpp:108.)
  return torch._C._cuda_getDeviceCount() > 0
```

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

```
'cpu'
```

```python
# 1. Get pretrained weights for ViT-Base
pretrained_vit_weights = torchvision.models.ViT_B_16_Weights.DEFAULT

# 2. Setup a ViT model instance with pretrained weights
pretrained_vit =
torchvision.models.vit_b_16(weights=pretrained_vit_weights).to(device)

# 3. Freeze the base parameters
for parameter in pretrained_vit.parameters():
    parameter.requires_grad = False

# 4. Change the classifier head
class_names = ['Paralysed','Not paralysed']

set_seeds()
pretrained_vit.heads = nn.Linear(in_features=768,
out_features=len(class_names)).to(device)
# pretrained_vit # uncomment for model output

from torchinfo import summary

# Print a summary using torchinfo (uncomment for actual output)
summary(model=pretrained_vit,
        input_size=(32, 3, 224, 224), # (batch_size, color_channels,
height, width)
        # col_names=["input_size"], # uncomment for smaller output
        col_names=["input_size", "output_size", "num_params",
"trainable"],
        col_width=20,
        row_settings=["var_names"]
)

======================================================================
======================================================================
Layer (type (var_name))                                      Input
Shape          Output Shape         Param #              Trainable
======================================================================
======================================================================
VisionTransformer (VisionTransformer)                        [32, 3,
224, 224]    [32, 2]              768                  Partial
├─Conv2d (conv_proj)                                         [32, 3,
224, 224]    [32, 768, 14, 14]    (590,592)            False
├─Encoder (encoder)                                          [32, 197,
768]       [32, 197, 768]       151,296              False
│    └─Dropout (dropout)                                     [32, 197,
768]       [32, 197, 768]       --                   --
│    └─Sequential (layers)                                   [32, 197,
768]       [32, 197, 768]       --                   False
│    │    └─EncoderBlock (encoder_layer_0)                   [32, 197,
768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_1)                   [32, 197,
768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_2)                   [32, 197,
768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_3)                   [32, 197,
768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_4)                   [32, 197,
768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_5)                   [32, 197,
768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_6)                   [32, 197,
768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_7)                   [32, 197,
768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_8)                   [32, 197,
768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_9)                   [32, 197,
768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_10)                  [32, 197,
768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_11)                  [32, 197,
768]       [32, 197, 768]       (7,087,872)          False
│    └─LayerNorm (ln)                                        [32, 197,
768]       [32, 197, 768]       (1,536)              False
├─Linear (heads)                                             [32, 768]
[32, 2]              1,538                True
======================================================================
======================================================================
Total params: 85,800,194
Trainable params: 1,538
Non-trainable params: 85,798,656
Total mult-adds (Units.GIGABYTES): 5.52
======================================================================
======================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 3330.74
Params size (MB): 229.20
Estimated Total Size (MB): 3579.20
======================================================================
======================================================================
```

Notice how only the output layer is trainable, where as, all of the rest of the layers are untrainable (frozen).

```
# Setup directory paths to train and test images
train_dir = '/home/ex5/Desktop/Paralysis/train'
test_dir = '/home/ex5/Desktop/Paralysis/test'
```

Remember, if you're going to use a pretrained model, it's generally important to ensure your own custom data is transformed/formatted in the same way the data the original model was trained on.

```
# Get automatic transforms from pretrained ViT weights
pretrained_vit_transforms = pretrained_vit_weights.transforms()
print(pretrained_vit_transforms)

ImageClassification(
    crop_size=[224]
    resize_size=[256]
    mean=[0.485, 0.456, 0.406]
    std=[0.229, 0.224, 0.225]
    interpolation=InterpolationMode.BILINEAR
)
```

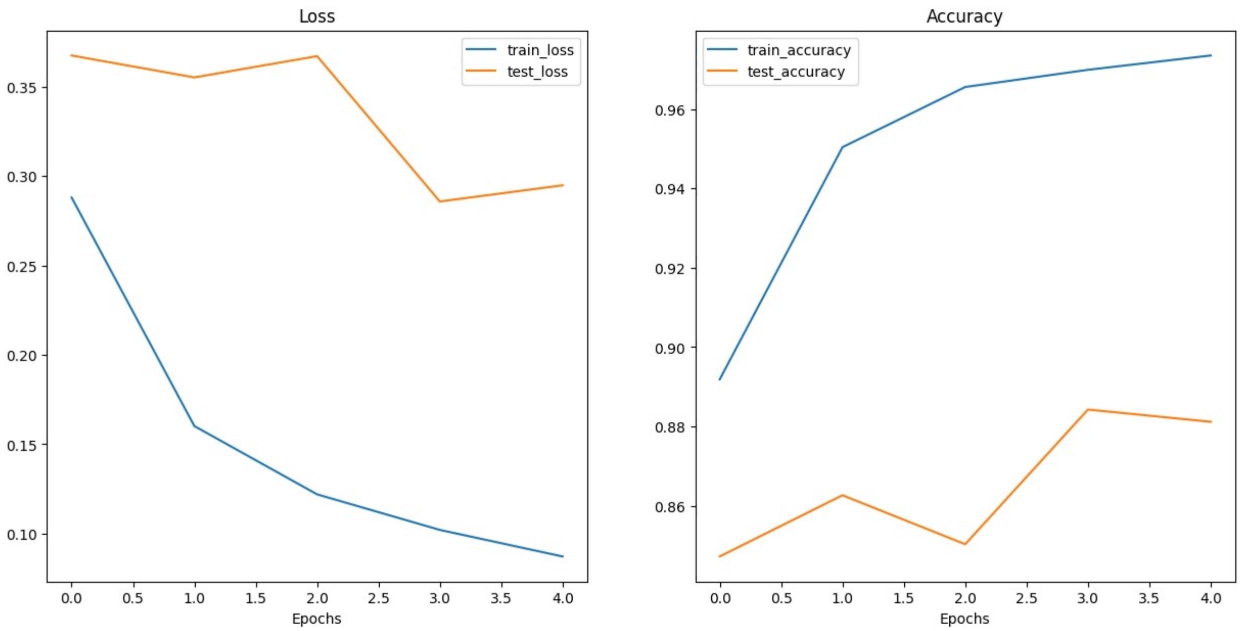
And now we've got transforms ready, we can turn our images into DataLoaders using the create_dataloaders()

```
import os

from torchvision import datasets, transforms
from torch.utils.data import DataLoader

NUM_WORKERS = os.cpu_count()

def create_dataloaders(
    train_dir: str,
    test_dir: str,
    transform: transforms.Compose,
    batch_size: int,
    num_workers: int=NUM_WORKERS
):

  # Use ImageFolder to create dataset(s)
  train_data = datasets.ImageFolder(train_dir, transform=transform)
  test_data = datasets.ImageFolder(test_dir, transform=transform)

  # Get class names
  class_names = train_data.classes

  # Turn images into data loaders
  train_dataloader = DataLoader(
      train_data,
      batch_size=batch_size,
      shuffle=True,
      num_workers=num_workers,
      pin_memory=True,
  )
  test_dataloader = DataLoader(
      test_data,
      batch_size=batch_size,
      shuffle=False,
      num_workers=num_workers,
      pin_memory=True,
  )

  return train_dataloader, test_dataloader, class_names

# Setup dataloaders
train_dataloader_pretrained, test_dataloader_pretrained, class_names =
create_dataloaders(train_dir=train_dir,

test_dir=test_dir,

transform=pretrained_vit_transforms,

batch_size=8) # Could increase if we had more samples, such as here:
https://arxiv.org/abs/2205.01580 (there are other improvements there
too...)

from going_modular.going_modular import engine

# Create optimizer and loss function
optimizer = torch.optim.Adam(params=pretrained_vit.parameters(),
                             lr=1e-3)
loss_fn = torch.nn.CrossEntropyLoss()
```

```
# Train the classifier head of the pretrained ViT feature extractor
model
set_seeds()
pretrained_vit_results = engine.train(model=pretrained_vit,

train_dataloader=train_dataloader_pretrained,

test_dataloader=test_dataloader_pretrained,
                                      optimizer=optimizer,
                                      loss_fn=loss_fn,
                                      epochs=5,
                                      device=device)

/home/ex5/miniconda3/lib/python3.12/site-packages/tqdm/auto.py:21:
TqdmWarning: IProgress not found. Please update jupyter and
ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
 20%|██        | 1/5 [07:09<28:39, 429.99s/it]

Epoch: 1 | train_loss: 0.2880 | train_acc: 0.8919 | test_loss: 0.3675
| test_acc: 0.8472

 40%|████      | 2/5 [14:18<21:26, 428.95s/it]

Epoch: 2 | train_loss: 0.1602 | train_acc: 0.9504 | test_loss: 0.3552
| test_acc: 0.8627

 60%|██████    | 3/5 [21:27<14:17, 428.97s/it]

Epoch: 3 | train_loss: 0.1220 | train_acc: 0.9656 | test_loss: 0.3671
| test_acc: 0.8503

 80%|████████  | 4/5 [28:36<07:09, 429.11s/it]

Epoch: 4 | train_loss: 0.1021 | train_acc: 0.9699 | test_loss: 0.2858
| test_acc: 0.8843

100%|██████████| 5/5 [35:44<00:00, 428.99s/it]

Epoch: 5 | train_loss: 0.0872 | train_acc: 0.9735 | test_loss: 0.2949
| test_acc: 0.8812
```

pretrained ViT performed far better than our custom ViT model trained from scratch (in the same amount of time).

```
# Plot the loss curves
from helper_functions import plot_loss_curves

plot_loss_curves(pretrained_vit_results)
```



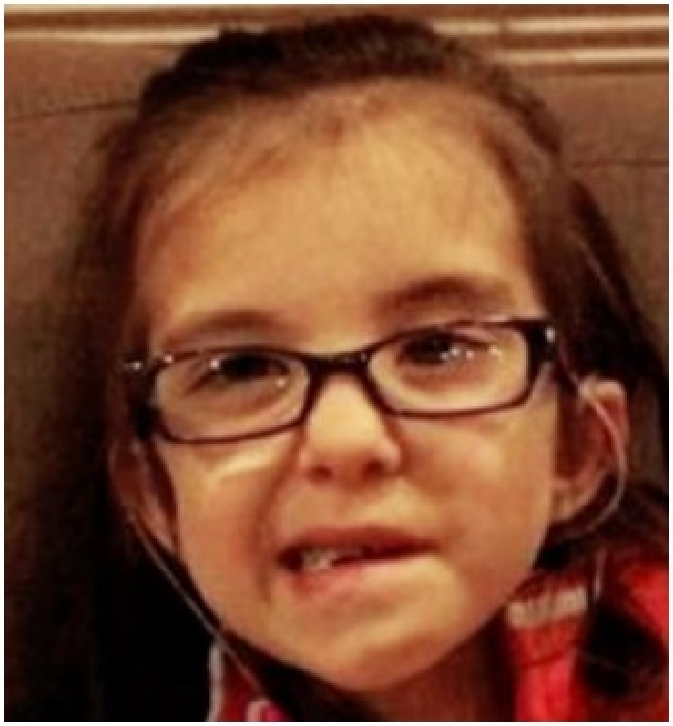
That's the power of transfer learning!

We managed to get outstanding results with the same model architecture, except our custom implementation was trained from scratch (worse performance) and this feature extractor model has the power of pretrained weights from ImageNet behind it.

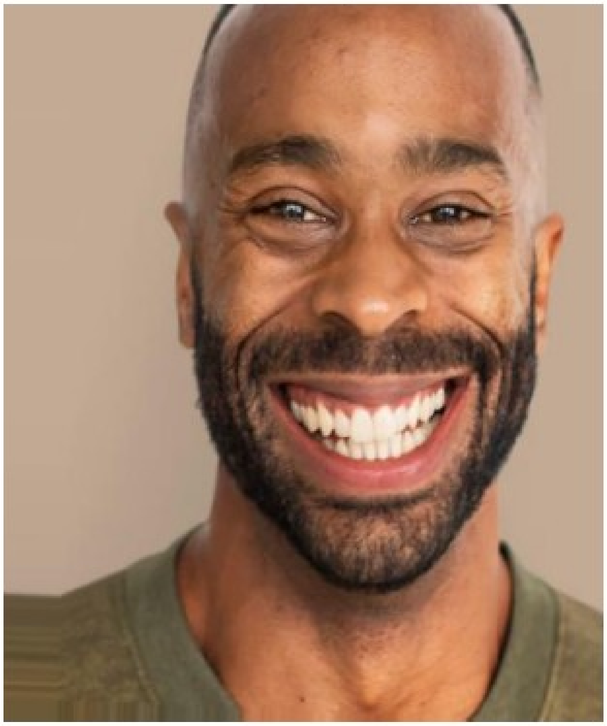
Let's make Prediction:

```
import requests

# Import function to make predictions on images and plot them
from going_modular.going_modular.predictions import pred_and_plot_image

# Setup custom image path
custom_image_path = "/home/ex5/Desktop/Paralysis/train/Stroke/a-26-278x300.jpg"

# Predict on custom image
pred_and_plot_image(model=pretrained_vit,
                    image_path=custom_image_path,
                    class_names=class_names)
```

Pred: Stroke | Prob: 0.988



```
# Import function to make predictions on images and plot them
from going_modular.going_modular.predictions import
pred_and_plot_image

# Setup custom image path
custom_image_path =
"/home/ex5/Desktop/Paralysis/train/Non-stroke/aug_0_121.jpg"

# Predict on custom image
pred_and_plot_image(model=pretrained_vit,
                    image_path=custom_image_path,
                    class_names=class_names)
```



Pred: Non-stroke | Prob: 0.893