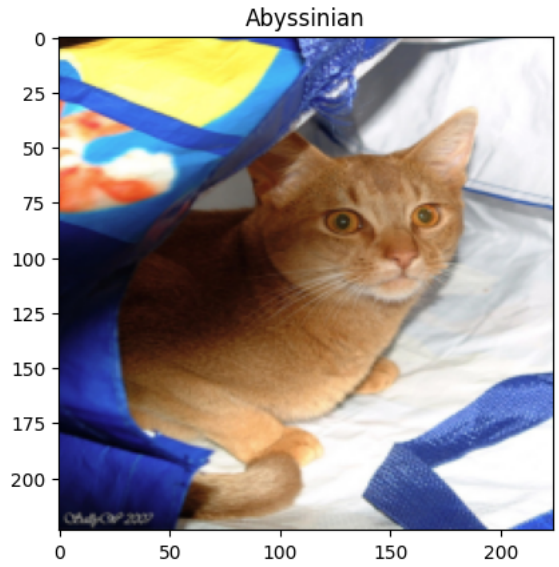
```
from torchvision.datasets import OxfordIIITPet
import torch
```

```
from torchvision.transforms import ToTensor,Resize
from torchvision import transforms

custom_transform = transforms.Compose([
    Resize((224,224)),
    ToTensor()
])
```

```
dataset = OxfordIIITPet(root='data', download=True, transform=custom_transform)
len(dataset)
```

```
3680
```

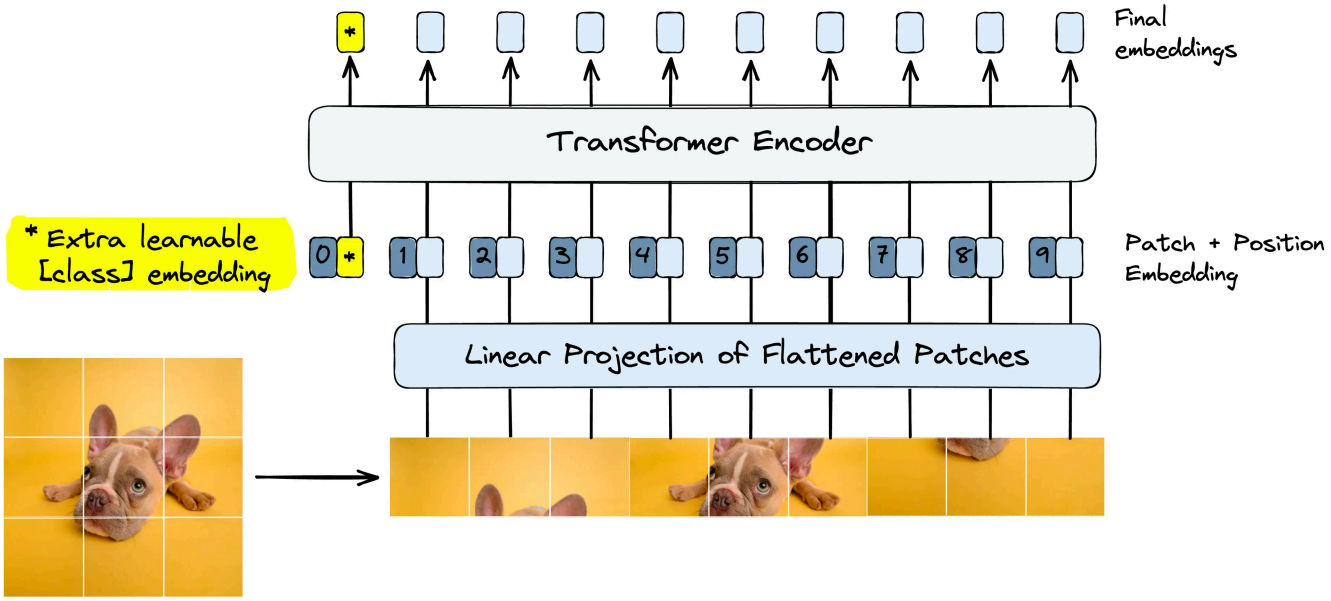
```
from matplotlib import pyplot as plt
image,label = dataset[0]

plt.imshow(image.permute(1,2,0))
plt.title(dataset.classes[label])
plt.show()
```




```
from torch.utils.data import DataLoader,random_split
import math

train_size = math.floor(0.8 * len(dataset))
test_size = len(dataset) - train_size
val_size = test_size//2
test_size -= val_size

train_dataset,test_dataset,val_dataset = random_split(dataset,[train_size,test_size,val_size])

train_dataloader = DataLoader(train_dataset,batch_size=32,shuffle=True)
test_dataloader = DataLoader(test_dataset,batch_size=32,shuffle=False)
val_dataloader = DataLoader(val_dataset,batch_size=32,shuffle=False)
```

Vision Transformer

Architecture



source:https://www.pinecone.io/learn/series/image-search/vision-transformers/

Step one Turning images to patches

- input_image = H * W * C
- H = image height
- W = image width
- C = color channels
- output = N*(P^2)*C
- P = Patch size
- N = (H * W)/(P^2)

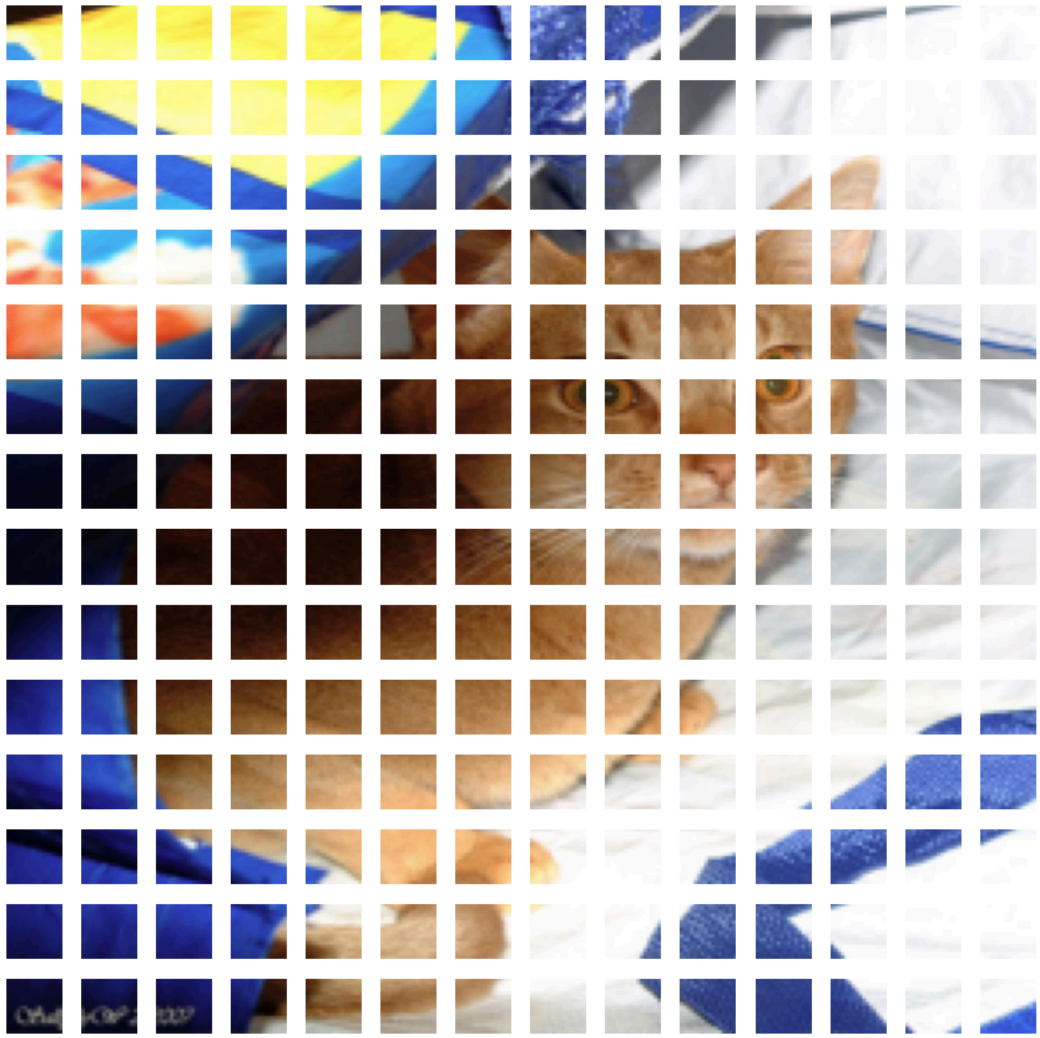
```
C,H,W = dataset[0][0].shape
C,H,W
```

```
(3, 224, 224)
```

```
P = 16
N = (H*W)//(P**2)
N
```

```
196
```

```
fig,axes = plt.subplots(nrows=H//P,ncols=W//P,figsize=(P/2, P/2))
image = image.permute(1,2,0)
for i ,r_patch in enumerate(range(0,H,P)):
    for j , c_patch in enumerate(range(0,W,P)):
        axes[i,j].imshow(image[r_patch:r_patch+P,c_patch:c_patch+P,:])
        axes[i,j].axis('off')
plt.tight_layout()
plt.show()
```



```
convo_layer = torch.nn.Conv2d(in_channels=3,out_channels=124,kernel_size=P,stride=P,padding=0,bias=False)
```

```
result = convo_layer(image.permute(2,1,0).unsqueeze(0))
result.shape
```

```
torch.Size([1, 124, 14, 14])
```

```
flatten_layer = torch.nn.Flatten(start_dim=2)
flatten_layer(result).shape
```

```
torch.Size([1, 124, 196])
```

Making patch embedding module

```
class PatchEmbedding(torch.nn.Module):
  def __init__(self,input=3,emb_dim=124,patch=16) -> None:
     super().__init__()
     self.patch = patch
     self.convo_layer = torch.nn.Conv2d(in_channels=input,out_channels=emb_dim,kernel_size=patch,stride=patch,paddin
     self.flatten_layer = torch.nn.Flatten(start_dim=2)

  def forward(self,x):
    _,c,h,w = x.shape
    assert h % self.patch == 0 and w % self.patch == 0, f"Wrong image shape"
    x = self.convo_layer(x)
    x = self.flatten_layer(x)
    return x.permute(0,2,1)
```

```
patch_embedding = PatchEmbedding()
patch_embedding(image.permute(2,0,1).unsqueeze(0)).shape
```

```
torch.Size([1, 196, 124])
```

Creating class token embedding

```
cls = torch.nn.Parameter(torch.randn((1,1,124),requires_grad=True))
```

```
torch.concat([cls,patch_embedding(image.permute(2,0,1).unsqueeze(0))],dim=1)
```

```
tensor([[[-0.3241,  1.0390, -0.4463,  ...,  1.1355,  0.2858, -0.6562],
         [-0.0127, -0.1933, -0.0441,  ...,  0.6304, -0.4491,  0.0538],
         [ 0.0818, -0.1453, -0.0730,  ...,  0.5984, -0.4503, -0.4029],
         ...,
         [ 0.2723,  0.2628,  0.1704,  ..., -0.2532, -0.0590, -0.2091],
         [ 0.3476,  0.1734,  0.1944,  ...,  0.2821, -0.1979, -0.3144],
         [ 0.5062,  0.4224,  0.1708,  ...,  0.1969, -0.1735, -0.3429]]],
       grad_fn=<CatBackward0>)
```

Positional Embedding

Positional Encoding Formula

- $PE_{(pos,2i)} = \sin(pos/10000^{2i/dim\_model})$
- $PE_{(pos,2i+1)} = \cos(pos/10000^{2i/dim\_model})$

where

- i is i^{th} dim from 0-(dim_model/2 - 1)
- pos is index from 0-(seq_length-1)
- dim_model is dimenstion of the model

Example:

for i = 0,pos=0

2i=0, 2i+1=1

- $PE_{(0,0)} = \sin(0/10000^{0/dim\_model})$
- $PE_{(0,1)} = \cos(0/10000^{0/dim\_model})$

```
class PositionalEmbedding(torch.nn.Module):
  def __init__(self,dim_model,seq_length,dropout=0.0) -> None:
    super().__init__()

    assert dim_model % 2 == 0, "dim_model must be even"

    self.dim_model = dim_model//2
    self.seq_length = seq_length

  def forward(self,x):

    pos = torch.arange(self.seq_length,dtype=torch.float).reshape(self.seq_length,1)
    depths = torch.arange(self.dim_model,dtype=torch.float).reshape(1,self.dim_model)

    pe = pos/torch.pow(10000,depths/self.dim_model)
    pe = torch.concat([torch.sin(pe),torch.cos(pe)],dim=1)

    return x + pe.unsqueeze(0).to(x.device)
```

```
pe = PositionalEmbedding(124,196)
pe(patch_embedding(image.permute(2,0,1).unsqueeze(0))).shape
```

```
torch.Size([1, 196, 124])
```

```
class ImageEmbedding(torch.nn.Module):
  def __init__(self,shape:tuple[int],input=3,emb_dim=124,patch=16) -> None:
    super().__init__()
    self.patch_embedding = PatchEmbedding(input,emb_dim,patch)
    self.cls = torch.nn.Parameter(torch.randn((1,1,emb_dim),requires_grad=True))
    self.emb_dim = emb_dim
    self.patch = patch
    assert (shape[0] * shape[1])/(patch**2) % 2 == 0, "Image size must be divisible by patch size"
    self.seq = (shape[0] * shape[1])//(patch**2)
```

```
    self.positional_embedding = PositionalEmbedding(emb_dim,self.seq+1)
    self.shape = shape

  def forward(self,x):
    batch_size = x.shape[0]
    x = self.patch_embedding(x)
    cls_tokens = self.cls.expand(batch_size, -1, -1)
    x = torch.concat([cls_tokens, x],dim=1)
    x = self.positional_embedding(x)
    return x
```

```
ie = ImageEmbedding(shape=(224,224))
ie(image.permute(2,0,1).unsqueeze(0)).shape
```

```
torch.Size([1, 197, 124])
```

Attention block

```
class AttentionBlock(torch.nn.Module):
  def __init__(self,emb_dim,num_heads) -> None:
    super().__init__()
    self.emb_dim = emb_dim
    self.num_heads = num_heads
    assert emb_dim % num_heads == 0, "emb_dim must be divisible by num_heads"
    self.norm1 = torch.nn.LayerNorm(emb_dim)
    self.attention = torch.nn.MultiheadAttention(emb_dim,num_heads)


  def forward(self,x):
    x = x + self.attention(self.norm1(x),self.norm1(x),self.norm1(x))[0]
    return x
```

```
atten_block = AttentionBlock(124,4)
att = atten_block(ie(image.permute(2,0,1).unsqueeze(0)))
att.shape
```

```
torch.Size([1, 197, 124])
```

MLP BLOCK

```
class MultiLayerPerceptron(torch.nn.Module):
  def __init__(self,emb_dim,mlp_dim,drop_out=0.1) -> None:
    super().__init__()
    self.normalize = torch.nn.LayerNorm(emb_dim)
    self.linear1 = torch.nn.Linear(emb_dim,mlp_dim)
    self.gelu = torch.nn.GELU()
    self.linear2 = torch.nn.Linear(mlp_dim,emb_dim)
    self.dropout = torch.nn.Dropout(drop_out)

  def forward(self,x):
    x = self.normalize(x)
    x = self.linear1(x)
    x = self.gelu(x)
    x = self.linear2(x)
    x = self.dropout(x)
    return x
```

```
mlp = MultiLayerPerceptron(124,256)
mlp(att).shape
```

```
torch.Size([1, 197, 124])
```

```
Start coding or generate with AI.
```

Encode Block

```
class EncoderBlock(torch.nn.Module):
  def __init__(self,emb_dim=124,num_heads=4,mlp_dim=512,drop_out=0.1) -> None:
    super().__init__()
    self.attention = AttentionBlock(emb_dim,num_heads)
    self.mlp = MultiLayerPerceptron(emb_dim,mlp_dim,drop_out)
```

```
  def forward(self,x):
    x = self.attention(x)
    x = self.mlp(x)
    return x
```

```
encoder_block = EncoderBlock()
encoder_block(ie(image.permute(2,0,1).unsqueeze(0))).shape
```

```
torch.Size([1, 197, 124])
```

```
Start coding or generate with AI.
```

Transformer

```
class VisionTransformer(torch.nn.Module):
  def __init__(self,shape:tuple[int],out,input=3,emb_dim=124,patch=16,num_heads=4,mlp_dim=512,drop_out=0.1,num_block
    super().__init__()
    self.image_embedding = ImageEmbedding(shape,input,emb_dim,patch)
    self.encoder_blocks = torch.nn.Sequential(*[EncoderBlock(emb_dim,num_heads,mlp_dim,drop_out) for _ in range(num_
    self.linear = torch.nn.Linear(emb_dim,out)
    self.norm = torch.nn.LayerNorm(emb_dim)

  def forward(self,x):
    x = self.image_embedding(x)
    x = self.encoder_blocks(x)
    x = self.norm(x)
    x = x[:,0,:]
    x = self.linear(x)
    return x
```

```
vision_transformer = VisionTransformer(shape=(224,224),out=5)
vision_transformer(image.permute(2,0,1).unsqueeze(0)).shape
```

```
torch.Size([1, 5])
```

```
!pip install torchinfo
```

```
Requirement already satisfied: torchinfo in /usr/local/lib/python3.12/dist-packages (1.8.0)
```

```
from torchinfo import summary

summary(vision_transformer,input_size=(1,3,224,224))
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
VisionTransformer                        [1, 5]                    --
├─ImageEmbedding: 1-1                    [1, 197, 124]             124
│    └─PatchEmbedding: 2-1               [1, 196, 124]             --
│    │    └─Conv2d: 3-1                  [1, 124, 14, 14]          95,232
│    │    └─Flatten: 3-2                 [1, 124, 196]             --
│    └─PositionalEmbedding: 2-2          [1, 197, 124]             --
├─Sequential: 1-2                        [1, 197, 124]             --
│    └─EncoderBlock: 2-3                 [1, 197, 124]             --
│    │    └─AttentionBlock: 3-3          [1, 197, 124]             62,248
│    │    └─MultiLayerPerceptron: 3-4    [1, 197, 124]             127,860
│    └─EncoderBlock: 2-4                 [1, 197, 124]             --
│    │    └─AttentionBlock: 3-5          [1, 197, 124]             62,248
│    │    └─MultiLayerPerceptron: 3-6    [1, 197, 124]             127,860
│    └─EncoderBlock: 2-5                 [1, 197, 124]             --
│    │    └─AttentionBlock: 3-7          [1, 197, 124]             62,248
│    │    └─MultiLayerPerceptron: 3-8    [1, 197, 124]             127,860
│    └─EncoderBlock: 2-6                 [1, 197, 124]             --
│    │    └─AttentionBlock: 3-9          [1, 197, 124]             62,248
│    │    └─MultiLayerPerceptron: 3-10   [1, 197, 124]             127,860
│    └─EncoderBlock: 2-7                 [1, 197, 124]             --
│    │    └─AttentionBlock: 3-11         [1, 197, 124]             62,248
│    │    └─MultiLayerPerceptron: 3-12   [1, 197, 124]             127,860
│    └─EncoderBlock: 2-8                 [1, 197, 124]             --
│    │    └─AttentionBlock: 3-13         [1, 197, 124]             62,248
│    │    └─MultiLayerPerceptron: 3-14   [1, 197, 124]             127,860
├─LayerNorm: 1-3                         [1, 197, 124]             248
├─Linear: 1-4                            [1, 5]                    625
==========================================================================================
Total params: 1,236,877
Trainable params: 1,236,877
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 19.44
==========================================================================================
```

```
Input size (MB): 0.60
Forward/backward pass size (MB): 11.09
Params size (MB): 3.46
Estimated Total Size (MB): 15.16
==========================================================================================
```

Training

```
model = VisionTransformer(shape=(224,224),out=len(dataset.classes),emb_dim=512,mlp_dim=1024)
summary(model,input_size=(1,3,224,224))
```

```
==========================================================================================
Layer (type:depth-idx)                             Output Shape              Param #
==========================================================================================
VisionTransformer                                  [1, 37]                   --
├─ImageEmbedding: 1-1                              [1, 197, 512]             512
│    └─PatchEmbedding: 2-1                         [1, 196, 512]             --
│    │    └─Conv2d: 3-1                            [1, 512, 14, 14]          393,216
│    │    └─Flatten: 3-2                           [1, 512, 196]             --
│    └─PositionalEmbedding: 2-2                    [1, 197, 512]             --
├─Sequential: 1-2                                  [1, 197, 512]             --
│    └─EncoderBlock: 2-3                           [1, 197, 512]             --
│    │    └─AttentionBlock: 3-3                    [1, 197, 512]             1,051,648
│    │    └─MultiLayerPerceptron: 3-4              [1, 197, 512]             1,051,136
│    └─EncoderBlock: 2-4                           [1, 197, 512]             --
│    │    └─AttentionBlock: 3-5                    [1, 197, 512]             1,051,648
│    │    └─MultiLayerPerceptron: 3-6              [1, 197, 512]             1,051,136
│    └─EncoderBlock: 2-5                           [1, 197, 512]             --
│    │    └─AttentionBlock: 3-7                    [1, 197, 512]             1,051,648
│    │    └─MultiLayerPerceptron: 3-8              [1, 197, 512]             1,051,136
│    └─EncoderBlock: 2-6                           [1, 197, 512]             --
│    │    └─AttentionBlock: 3-9                    [1, 197, 512]             1,051,648
│    │    └─MultiLayerPerceptron: 3-10             [1, 197, 512]             1,051,136
│    └─EncoderBlock: 2-7                           [1, 197, 512]             --
│    │    └─AttentionBlock: 3-11                   [1, 197, 512]             1,051,648
│    │    └─MultiLayerPerceptron: 3-12             [1, 197, 512]             1,051,136
│    └─EncoderBlock: 2-8                           [1, 197, 512]             --
│    │    └─AttentionBlock: 3-13                   [1, 197, 512]             1,051,648
│    │    └─MultiLayerPerceptron: 3-14             [1, 197, 512]             1,051,136
├─LayerNorm: 1-3                                   [1, 197, 512]             1,024
├─Linear: 1-4                                      [1, 37]                   18,981
==========================================================================================
Total params: 13,030,437
Trainable params: 13,030,437
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 83.42
==========================================================================================
Input size (MB): 0.60
Forward/backward pass size (MB): 35.50
Params size (MB): 26.90
Estimated Total Size (MB): 63.01
==========================================================================================
```

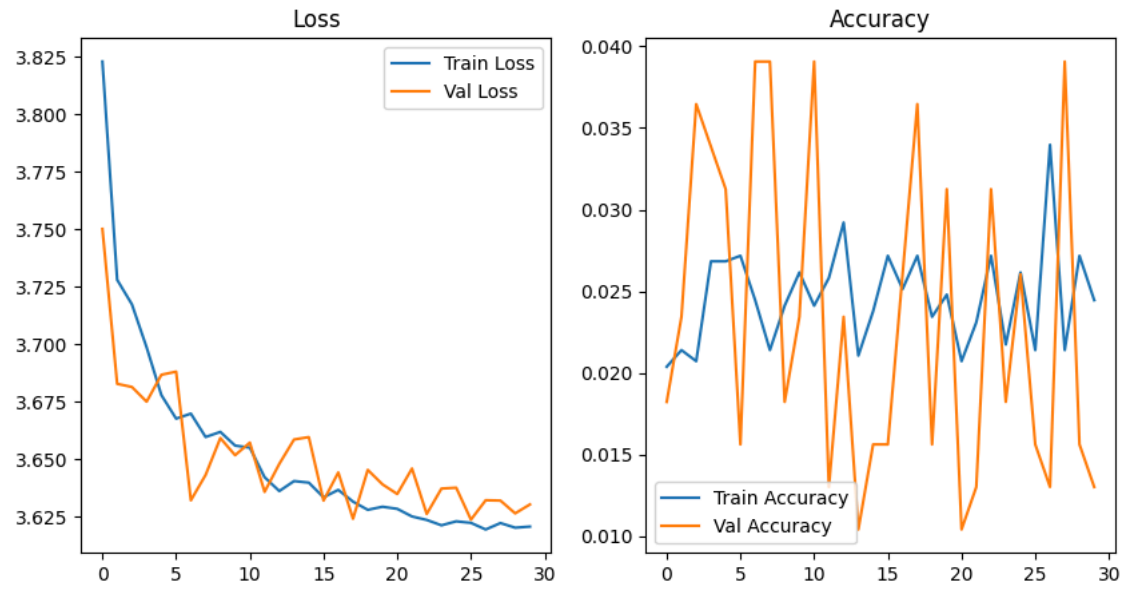
```
from tqdm.auto import tqdm
from sklearn.metrics import accuracy_score
def train(model,train,val,epoch,loss_fn,optimizer,device):
  train_loss = []
  val_loss = []
  train_accuray = []
  val_accuray = []
  for i in tqdm(range(epoch)):
    model.train()
    batch_train_loss = 0
    batch_val_loss = 0
    batch_train_accuray = 0
    batch_val_accuray = 0
    for x,y in train:
      x,y = x.to(device),y.to(device)
      optimizer.zero_grad()
      y_pred = model(x)
      loss = loss_fn(y_pred,y)
      loss.backward()
      optimizer.step()
      batch_train_loss += loss.item()
      batch_train_accuray += accuracy_score(y.cpu().detach().numpy(),y_pred.argmax(dim=-1).cpu().detach().numpy())

    train_loss.append(batch_train_loss/len(train))
    train_accuray.append(batch_train_accuray/len(train))

    model.eval()

    with torch.inference_mode():
      for x,y in val:
```

```
            x,y = x.to(device),y.to(device)
            y_pred = model(x)
            loss = loss_fn(y_pred,y)
            batch_val_loss += loss.item()
            batch_val_accuray += accuracy_score(y.cpu().detach().numpy(),y_pred.argmax(dim=-1).cpu().detach().numpy())
          val_loss.append(batch_val_loss/len(val))
          val_accuray.append(batch_val_accuray/len(val))


        if i%2 == 0:
          print(f"Epoch: {i+1}")
          print(f"Train Loss: {train_loss[-1]:.4f} | Train Accuracy: {train_accuray[-1]:.4f}")
          print(f"Val Loss: {val_loss[-1]:.4f} | Val Accuracy: {val_accuray[-1]:.4f}")
          print("-"*50)

      return train_loss,val_loss,train_accuray,val_accuray
```

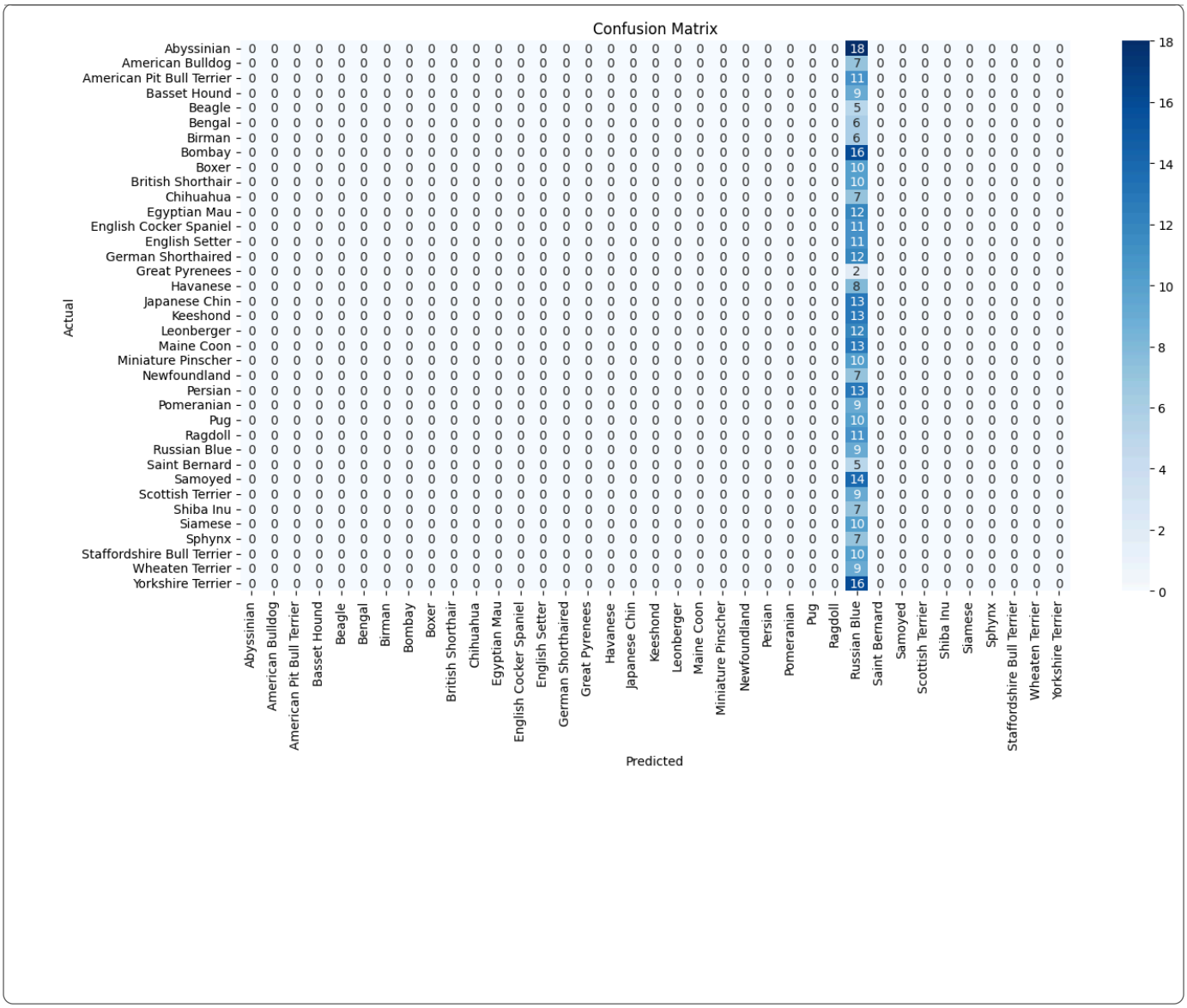
```
loss = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(),lr=0.001)
device = 'cuda' if torch.cuda.is_available() else 'cpu'
model.to(device)

train_loss,val_loss,train_accuracy,val_accuracy=train(model,train_dataloader,val_dataloader,30,loss,optimizer,device
```

```
100%                                                30/30 [15:44<00:00, 31.42s/it]
Epoch: 1
Train Loss: 3.8229 | Train Accuracy: 0.0204
Val Loss: 3.7501 | Val Accuracy: 0.0182
--------------------------------------------------
Epoch: 3
Train Loss: 3.7173 | Train Accuracy: 0.0207
Val Loss: 3.6813 | Val Accuracy: 0.0365
--------------------------------------------------
Epoch: 5
Train Loss: 3.6778 | Train Accuracy: 0.0268
Val Loss: 3.6866 | Val Accuracy: 0.0312
--------------------------------------------------
Epoch: 7
Train Loss: 3.6697 | Train Accuracy: 0.0245
Val Loss: 3.6320 | Val Accuracy: 0.0391
--------------------------------------------------
Epoch: 9
Train Loss: 3.6618 | Train Accuracy: 0.0241
Val Loss: 3.6591 | Val Accuracy: 0.0182
--------------------------------------------------
Epoch: 11
Train Loss: 3.6549 | Train Accuracy: 0.0241
Val Loss: 3.6571 | Val Accuracy: 0.0391
--------------------------------------------------
Epoch: 13
Train Loss: 3.6360 | Train Accuracy: 0.0292
Val Loss: 3.6479 | Val Accuracy: 0.0234
--------------------------------------------------
Epoch: 15
Train Loss: 3.6397 | Train Accuracy: 0.0238
Val Loss: 3.6595 | Val Accuracy: 0.0156
--------------------------------------------------
Epoch: 17
Train Loss: 3.6365 | Train Accuracy: 0.0251
Val Loss: 3.6441 | Val Accuracy: 0.0260
--------------------------------------------------
Epoch: 19
Train Loss: 3.6279 | Train Accuracy: 0.0234
Val Loss: 3.6452 | Val Accuracy: 0.0156
--------------------------------------------------
Epoch: 21
Train Loss: 3.6283 | Train Accuracy: 0.0207
Val Loss: 3.6348 | Val Accuracy: 0.0104
--------------------------------------------------
Epoch: 23
Train Loss: 3.6235 | Train Accuracy: 0.0272
Val Loss: 3.6261 | Val Accuracy: 0.0312
--------------------------------------------------
Epoch: 25
Train Loss: 3.6229 | Train Accuracy: 0.0262
Val Loss: 3.6375 | Val Accuracy: 0.0260
--------------------------------------------------
Epoch: 27
Train Loss: 3.6193 | Train Accuracy: 0.0340
Val Loss: 3.6320 | Val Accuracy: 0.0130
--------------------------------------------------
Epoch: 29
Train Loss: 3.6201 | Train Accuracy: 0.0272
Val Loss: 3.6263 | Val Accuracy: 0.0156
--------------------------------------------------
```

```
import matplotlib.pyplot as plt
def plot(train_loss,val_loss,train_accuracy,val_accuracy):
  fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(10,5))

  ax[0].plot(train_loss,label='Train Loss')
  ax[0].plot(val_loss,label='Val Loss')
  ax[0].set_title('Loss')
  ax[0].legend()

  ax[1].plot(train_accuracy,label='Train Accuracy')
  ax[1].plot(val_accuracy,label='Val Accuracy')
  ax[1].set_title('Accuracy')
  ax[1].legend()

  plt.show()

plot(train_loss,val_loss,train_accuracy,val_accuracy)
```




```
from sklearn.metrics import confusion_matrix
import seaborn as sns

model.eval()
all_preds = []
all_labels = []
with torch.no_grad():
  for inputs, labels in test_dataloader:
    inputs, labels = inputs.to(device), labels.to(device)
    outputs = model(inputs)
    _, preds = torch.max(outputs, 1)
    all_preds.extend(preds.cpu().numpy())
    all_labels.extend(labels.cpu().numpy())

cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(15, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=dataset.classes, yticklabels=dataset.classes)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



Using Pretrained

```
import torch
import torchvision.models as models



weights = models.ViT_B_16_Weights.DEFAULT


model = models.vit_b_16(weights=weights)

summary(model,input_size=(1,3,224,224))
```

```
Downloading: "https://download.pytorch.org/models/vit_b_16-c867db91.pth" to /root/.cache/torch/hub/checkpoints/vit_b_
100%|██████████| 330M/330M [00:02<00:00, 142MB/s]
==========================================================================================
Layer (type:depth-idx)                             Output Shape              Param #
==========================================================================================
VisionTransformer                                  [1, 1000]                 768
├─Conv2d: 1-1                                      [1, 768, 14, 14]          590,592
├─Encoder: 1-2                                     [1, 197, 768]             151,296
│    └─Dropout: 2-1                                [1, 197, 768]             --
│    └─Sequential: 2-2                             [1, 197, 768]             --
│    │    └─EncoderBlock: 3-1                      [1, 197, 768]             7,087,872
│    │    └─EncoderBlock: 3-2                      [1, 197, 768]             7,087,872
│    │    └─EncoderBlock: 3-3                      [1, 197, 768]             7,087,872
│    │    └─EncoderBlock: 3-4                      [1, 197, 768]             7,087,872
│    │    └─EncoderBlock: 3-5                      [1, 197, 768]             7,087,872
│    │    └─EncoderBlock: 3-6                      [1, 197, 768]             7,087,872
│    │    └─EncoderBlock: 3-7                      [1, 197, 768]             7,087,872
│    │    └─EncoderBlock: 3-8                      [1, 197, 768]             7,087,872
│    │    └─EncoderBlock: 3-9                      [1, 197, 768]             7,087,872
│    │    └─EncoderBlock: 3-10                     [1, 197, 768]             7,087,872
```

```
│    │    └─EncoderBlock: 3-11                [1, 197, 768]             7,087,872
│    │    └─EncoderBlock: 3-12                [1, 197, 768]             7,087,872
│    └─LayerNorm: 2-3                         [1, 197, 768]             1,536
├─Sequential: 1-3                              [1, 1000]                 --
│    └─Linear: 2-4                            [1, 1000]                 769,000
==========================================================================================
Total params: 86,567,656
Trainable params: 86,567,656
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 173.23
==========================================================================================
Input size (MB): 0.60
Forward/backward pass size (MB): 104.09
Params size (MB): 232.27
Estimated Total Size (MB): 336.96
==========================================================================================
```

```
Start coding or generate with AI.
```

```
from huggingface_hub import PyTorchModelHubMixin

for param in model.parameters():
  param.requires_grad = False

model.heads = torch.nn.Linear(768,len(dataset.classes))

class HuggingFaceModel(torch.nn.Module, PyTorchModelHubMixin):
    def __init__(self, model):
        super().__init__()
        self.model = model
        self.config = {
            "name":"Vistion Transformer",
            "num_classes": len(dataset.classes),
            "image_size": 224,
            "patch_size": 16,
            "idx2label": dataset.classes,
            "label2idx": {label:i for i,label in enumerate(dataset.classes)}
        }

    def forward(self, x):
        return self.model(x)

model = HuggingFaceModel(model)


summary(model,input_size=(1,3,224,224))
```

```
==========================================================================================
Layer (type:depth-idx)                             Output Shape              Param #
==========================================================================================
HuggingFaceModel                                   [1, 37]                   --
├─VisionTransformer: 1-1                           [1, 37]                   768
│    └─Conv2d: 2-1                                 [1, 768, 14, 14]          (590,592)
│    └─Encoder: 2-2                                [1, 197, 768]             151,296
│    │    └─Dropout: 3-1                           [1, 197, 768]             --
│    │    └─Sequential: 3-2                        [1, 197, 768]             (85,054,464)
│    │    └─LayerNorm: 3-3                         [1, 197, 768]             (1,536)
│    └─Linear: 2-3                                 [1, 37]                   28,453
==========================================================================================
Total params: 85,827,109
Trainable params: 28,453
Non-trainable params: 85,798,656
Total mult-adds (Units.MEGABYTES): 172.49
==========================================================================================
Input size (MB): 0.60
Forward/backward pass size (MB): 104.09
Params size (MB): 229.31
Estimated Total Size (MB): 333.99
==========================================================================================
```

```
optimizer = torch.optim.Adam(model.parameters(),lr=0.001)
device = 'cuda' if torch.cuda.is_available() else 'cpu'
model.to(device)
train_loss,val_loss,train_accuracy,val_accuracy=train(model,train_dataloader,val_dataloader,30,loss,optimizer,device
```

```
100%                                                30/30 [29:35<00:00, 58.66s/it]
Epoch: 1
Train Loss: 1.4045 | Train Accuracy: 0.7456
Val Loss: 0.5587 | Val Accuracy: 0.8776
--------------------------------------------------
Epoch: 3
Train Loss: 0.2589 | Train Accuracy: 0.9457
Val Loss: 0.3408 | Val Accuracy: 0.9062
--------------------------------------------------
Epoch: 5
Train Loss: 0.1527 | Train Accuracy: 0.9721
Val Loss: 0.2689 | Val Accuracy: 0.9141
--------------------------------------------------
Epoch: 7
Train Loss: 0.1046 | Train Accuracy: 0.9851
Val Loss: 0.2446 | Val Accuracy: 0.9297
--------------------------------------------------
Epoch: 9
Train Loss: 0.0739 | Train Accuracy: 0.9912
Val Loss: 0.2461 | Val Accuracy: 0.9193
--------------------------------------------------
Epoch: 11
Train Loss: 0.0572 | Train Accuracy: 0.9956
Val Loss: 0.2364 | Val Accuracy: 0.9141
--------------------------------------------------
Epoch: 13
Train Loss: 0.0440 | Train Accuracy: 0.9983
Val Loss: 0.2277 | Val Accuracy: 0.9219
--------------------------------------------------
Epoch: 15
Train Loss: 0.0356 | Train Accuracy: 0.9990
Val Loss: 0.2204 | Val Accuracy: 0.9271
--------------------------------------------------
Epoch: 17
Train Loss: 0.0290 | Train Accuracy: 1.0000
Val Loss: 0.2240 | Val Accuracy: 0.9193
--------------------------------------------------
Epoch: 19
Train Loss: 0.0246 | Train Accuracy: 0.9993
Val Loss: 0.2112 | Val Accuracy: 0.9323
--------------------------------------------------
Epoch: 21
Train Loss: 0.0208 | Train Accuracy: 0.9997
Val Loss: 0.2144 | Val Accuracy: 0.9271
--------------------------------------------------
Epoch: 23
Train Loss: 0.0179 | Train Accuracy: 1.0000
Val Loss: 0.2205 | Val Accuracy: 0.9245
--------------------------------------------------
Epoch: 25
Train Loss: 0.0150 | Train Accuracy: 1.0000
Val Loss: 0.2129 | Val Accuracy: 0.9297
--------------------------------------------------
Epoch: 27
Train Loss: 0.0133 | Train Accuracy: 1.0000
Val Loss: 0.2121 | Val Accuracy: 0.9297
--------------------------------------------------
Epoch: 29
Train Loss: 0.0115 | Train Accuracy: 1.0000
Val Loss: 0.2193 | Val Accuracy: 0.9271
--------------------------------------------------
```

```
import matplotlib.pyplot as plt
def plot(train_loss,val_loss,train_accuracy,val_accuracy):
  fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(10,5))

  ax[0].plot(train_loss,label='Train Loss')
  ax[0].plot(val_loss,label='Val Loss')
  ax[0].set_title('Loss')
  ax[0].legend()

  ax[1].plot(train_accuracy,label='Train Accuracy')
  ax[1].plot(val_accuracy,label='Val Accuracy')
  ax[1].set_title('Accuracy')
  ax[1].legend()

  plt.show()

plot(train_loss,val_loss,train_accuracy,val_accuracy)
```