model_multipleVideos-class_wise_Implementation-Vision_Transformer_base

April 6, 2022

[1]:
```
import torch
import torchvision
import torchvision.transforms as transforms
import torchvision.models as models
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import time
from itertools import count
import natsort
import datetime
import numpy as np
import os
import math
```

[2]:
```
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler
import albumentations as A
from albumentations.pytorch import ToTensorV2
import cv2
import glob
import numpy
import random
import pandas as pd
import tqdm
torch.manual_seed(10)
```

[2]:
```
<torch._C.Generator at 0x27911ebf130>
```

[3]:
```
print(f"Is CUDA supported by this system? {torch.cuda.is_available()}")
print(f"CUDA version: {torch.version.cuda}")
# Storing ID of current CUDA device
cuda_id = torch.cuda.current_device()
print(f"ID of current CUDA device: {torch.cuda.current_device()}")
print(f"Name of current CUDA device: {torch.cuda.get_device_name(cuda_id)}")

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

```
print(device)
```

```
Is CUDA supported by this system? True
CUDA version: 11.3
ID of current CUDA device: 0
Name of current CUDA device: NVIDIA GeForce RTX 2070 Super
cuda:0
```

1 Building the dataset

[4]:
```
class SurgicalDataset(Dataset):
    def __init__(self, image_paths, labels, transform=False):
        super(SurgicalDataset, self).__init__()
        self.image_paths = image_paths
        self.labels = labels    #.astype(dtype='int')
        self.transform = transform

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

    def __getitem__(self, idx):
        image_filepath = self.image_paths[idx]
        image = cv2.imread(image_filepath)
        label = self.labels[idx]
        if self.transform is not None:
            image = self.transform(image=image)["image"]

        return image, label
```

[5]:
```
def get_transform(model_name):

    if model_name == 'alexnet':
        transform = A.Compose([
            A.Resize(227, 227),
            A.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
            ToTensorV2(),
        ])

    elif model_name == 'effinet':
        transform = A.Compose([
            A.Resize(224, 224),
            A.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
            ToTensorV2(),
        ])

    elif model_name == 'TransferViT':
```

```
        transform = A.Compose([
            A.Resize(224, 224),
            A.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
            ToTensorV2(),
        ])

    return transform
```

[6]:
```
# Preparing the datasets
# Get images
train_image_paths = []
train_data_path = r"C:
 ↪\Users\panji\EECS6691_Advanced_DL\Assignment2\training_data_images"
train_image_paths.append(glob.glob(train_data_path + '/*'))
# unpack the listed list
train_image_paths1 = [item for sublist in train_image_paths for item in sublist]
train_image_paths1 = natsort.natsorted(train_image_paths1)
print('len(train_image_paths1)', len(train_image_paths1))

# Get labels
df = pd.read_csv("Processed_data.csv")
df1 = df.loc[:,"Phases"].to_numpy()
df2 = df1.tolist()
print('len(df2)', len(df2))

# Preparing the datasets (images and labels)
dataset_train = pd.DataFrame(
    {'Link': train_image_paths1,
     'Label': df2,
    })
dataset_train1 = dataset_train.sample(frac=1, random_state=1)
train_image_paths = dataset_train1.loc[:,"Link"].to_numpy().tolist()
labels = dataset_train1.loc[:,"Label"].to_numpy().tolist()

# manually split the dataset
train_image_paths, valid_image_paths = train_image_paths[:int(0.
 ↪8*len(train_image_paths))], train_image_paths[int(0.
 ↪8*len(train_image_paths)):]
train_labels, valid_labels = labels[:int(0.8*len(labels))], labels[int(0.
 ↪8*len(labels)):]
print('train_labels', len(train_labels))
print('train_image_paths', len(train_image_paths))
print('label distribution in the training data', np.bincount(train_labels))
```

```
len(train_image_paths1) 215057
len(df2) 215057
train_labels 172045
```

```
train_image_paths 172045
label distribution in the training data [  243  8681 22901 41140   952 22305
666 10930   896  2308 44928 12987
  1789  1246    73]
```

2 Weighted Data Sampler

[7]:
```python
# from torch.utils.data import WeightedRandomSampler

# Get labels
df = pd.read_csv("Processed_data.csv")
df1 = df.loc[:,"Phases"].to_numpy()
df2 = df1.tolist()
print('len(df2)', len(df2))

# Preparing the datasets (images and labels)
dataset_train = pd.DataFrame(
    {'Link': train_image_paths1,
     'Label': df2,
    })
dataset_train1 = dataset_train.sample(frac=1, random_state=1)
train_image_paths = dataset_train1.loc[:,"Link"].to_numpy().tolist()
labels = dataset_train1.loc[:,"Label"].to_numpy().tolist()

summary = {i:0 for i in range(15)}
num_classes = 15
total_samples = 0
for i in train_labels:
    total_samples += 1
    summary[i] += 1
print(summary)
print(total_samples)

class_weights = [total_samples/summary[i] for i in range(num_classes)]
weights = [class_weights[train_labels[i]] for i in range(total_samples)]
sampler = WeightedRandomSampler(torch.DoubleTensor(weights), len(weights))
print(len(class_weights))
print(len(weights))
print(len(list(sampler)))
```

```
len(df2) 215057
{0: 243, 1: 8681, 2: 22901, 3: 41140, 4: 952, 5: 22305, 6: 666, 7: 10930, 8:
896, 9: 2308, 10: 44928, 11: 12987, 12: 1789, 13: 1246, 14: 73}
172045
15
172045
172045
```

3 Building the classifier class

[8]:
```
class Classifier():

    def __init__(self, name, model, dataloaders, parameter, use_cuda=False):

        '''
        @name: Experiment name. Will define stored results etc.
        @model: Any models
        @dataloaders: Dictionary with keys train, val and test and␣
↪corresponding dataloaders
        @class_names: list of classes, where the idx of class name corresponds␣
↪to the label used for it in the data
        @use_cuda: whether or not to use cuda
        '''

        self.name = name
        if use_cuda and not torch.cuda.is_available():
            raise Exception("Asked for CUDA but GPU not found")

        self.use_cuda = use_cuda
        self.epoch = parameter['epochs']
        self.lr = parameter['lr']
        self.batch_size = parameter['batch_size']

        self.model = model.to('cuda' if use_cuda else 'cpu') # model.to('cpu')
        self.criterion = nn.CrossEntropyLoss()
        self.optimizer = optim.Adam(self.model.parameters(), lr=self.lr)
        self.train_loader, self.valid_loader = self.
↪get_dataloaders(dataloaders['train_image_paths'],
                                                                    ␣
↪dataloaders['train_labels'],
                                                                    ␣
↪dataloaders['valid_image_paths'],
                                                                    ␣
↪dataloaders['valid_labels'],
                                                                    ␣
↪train_transforms=dataloaders['transforms'],
                                                                     batch_size␣
↪= self.batch_size,
                                                                    ␣
↪shuffle=parameter['shuffle'],
                                                                    sampler =␣
↪dataloaders['sampler'])
        self.class_names = parameter['class_names']
```

```
        self.activations_path = os.path.join('activations', self.name)
        self.kernel_path = os.path.join('kernel_viz', self.name)
        save_path = os.path.join(os.getcwd(), 'models', self.name)
        if not os.path.exists(save_path):
            os.makedirs(save_path)

        if not os.path.exists(self.activations_path):
            os.makedirs(self.activations_path)

        if not os.path.exists(self.kernel_path):
            os.makedirs(self.kernel_path)

        self.save_path = save_path

    def train(self, save=True):
        '''
        @epochs: number of epochs to train
        @save: whether or not to save the checkpoints
        '''
        best_val_accuracy = - math.inf

        for epoch in range(self.epoch):  # loop over the dataset multiple times
            self.model.train()
            t = time.time()
            running_loss = 0.0
            train_acc = 0
            val_accuracy = 0
            correct = 0
            total = 0
            count = 0
            loop = tqdm.tqdm(self.train_loader, total = len(self.train_loader),␣
↪leave = True)

            for img, label in loop:
                # get the inputs; data is a list of [inputs, labels]
                inputs, labels = img.to(device), label.to(device) #img.
↪to(device), label.to(device)

                # zero the parameter gradients
                self.optimizer.zero_grad()

                # forward + backward + optimize
                outputs = self.model(inputs)
                _, predictions = torch.max(outputs, 1)
                loss = self.criterion(outputs, labels)
                loss.backward()
                self.optimizer.step()
```

```
                # print statistics
                running_loss += loss.item()
                total += labels.shape[0]
                correct += (predictions == labels).sum().item()

                count += 1
                if count % 2000 == 1999:    # print every 2000 mini-batches
                    print(f'[{epoch + 1}, {count + 1:5d}] loss: {running_loss /␣
↪2000:.3f}')
                    running_loss = 0.0

            train_acc = 100 * correct / total
            print(f'Epoch:', epoch + 1, f'Training Epoch Accuracy:{train_acc}')

            # evaluate the validation dataset
            self.model.eval()
            correct_pred = {classname: 0 for classname in self.class_names}
            total_pred = {classname: 0 for classname in self.class_names}

            # again no gradients needed
            correct = 0
            total = 0
            with torch.no_grad():
                for data in self.valid_loader:
                    images, labels = data[0].to(device), data[1].to(device)␣
↪#data[0], data[1]
                    outputs = self.model(images)
                    _, predictions = torch.max(outputs, 1)
                    # collect the correct predictions for each class
                    total += labels.shape[0]
                    correct += (predictions == labels).sum().item()

                    for label, prediction in zip(labels, predictions):
                        if label == prediction:
                            correct_pred[classes[label]] += 1
                        total_pred[classes[label]] += 1

            val_accuracy = 100 * correct / total
            print(f'Epoch:', epoch + 1, f'Validation Epoch Accuracy:
↪{val_accuracy}')

            # print the summary for each class
            print('Epoch:', epoch + 1, 'Correct predictions', correct_pred)
            print('Epoch:', epoch + 1, 'Total predictions', total_pred)
            print('Epoch:', epoch + 1, 'Correct predictions', correct_pred)
            print('Epoch:', epoch + 1, 'Total predictions', total_pred)
```

```
            # inspect the time taken to train one epoch
            d = time.time()-t
            print('Fininsh Trainig Epoch', epoch, '!', 'Time used:', d)

            if save:
                torch.save(self.model.state_dict(), os.path.join(self.
↪save_path, f'epoch_{epoch}.pt'))
                if val_accuracy > best_val_accuracy:
                    torch.save(self.model.state_dict(), os.path.join(self.
↪save_path, 'best.pt'))
                    best_val_accuracy = val_accuracy

        print('Done training!')


    def evaluate(self):
        # for evaluating the test dataset if there were any.
        try:
            assert os.path.exists(os.path.join(self.save_path, 'best.pt'))

        except:
            print('Please train first')
            return

        self.model.load_state_dict(torch.load(os.path.join(self.save_path,␣
↪'best.pt')))
        self.model.eval()


    def get_dataloaders(self, train_image_paths, train_labels,␣
↪valid_image_paths, valid_labels, train_transforms=False, batch_size=32,␣
↪shuffle=True, sampler = None):
        train_dataset = SurgicalDataset(train_image_paths,train_labels,␣
↪train_transforms)
        val_dataset = SurgicalDataset(valid_image_paths,valid_labels,␣
↪train_transforms)
        train_loader = DataLoader(train_dataset, batch_size, shuffle, sampler)
        valid_loader = DataLoader(val_dataset, batch_size, shuffle = True)

        return train_loader, valid_loader


    def grad_cam_on_input(self, img):

        try:
```

```
            assert os.path.exists(os.path.join(self.save_path, 'best.pt'))

        except:
            print('It appears you are testing the model without training. Please train first')
            return

        self.model.load_state_dict(torch.load(os.path.join(self.save_path, 'best.pt')))


        self.model.eval()
        img = img.to('cuda' if self.use_cuda else 'cpu')


        out = self.model(img)

        _, pred = torch.max(out, 1)

        predicted_class = self.class_names[int(pred)]
        print(f'Predicted class was {predicted_class}')

        out[:, pred].backward()
        gradients = self.model.get_gradient_activations()

        print('Gradients shape: ', f'{gradients.shape}')

        mean_gradients = torch.mean(gradients, [0, 2, 3]).cpu()
        activations = self.model.get_final_conv_layer(img).detach().cpu()

        print('Activations shape: ', f'{activations.shape}')

        for idx in range(activations.shape[1]):
            activations[:, idx, :, :] *= mean_gradients[idx]

        final_heatmap = np.maximum(torch.mean(activations, dim=1).squeeze(), 0)

        final_heatmap /= torch.max(final_heatmap)

        return final_heatmap

    def trained_kernel_viz(self):

        all_layers = [0, 3]
        all_filters = []
        for layer in all_layers:
```

```
            filters = self.model.conv_model[layer].weight
            all_filters.append(filters.detach().cpu().clone()[:8, :8, :, :])

        for filter_idx in range(len(all_filters)):

            filter = all_filters[filter_idx]
            print(filter.shape)
            filter = filter.contiguous().view(-1, 1, filter.shape[2], filter.
↪shape[3])
            image = show_img(make_grid(filter))
            image = 255 * image
            cv2.imwrite(os.path.join(self.kernel_path,␣
↪f'filter_layer{all_layers[filter_idx]}.jpg'), image)


    def activations_on_input(self, img):

        img = img.to('cuda' if self.use_cuda else 'cpu')

        all_layers = [0,3,6,8,10]
        all_viz = []

        # looking at the outputs of the relu
        for each in all_layers:

            current_model = self.model.conv_model[:each+1]
            current_out = current_model(img)
            all_viz.append(current_out.detach().cpu().clone()[:, :64, :, :])

        for viz_idx in range(len(all_viz)):

            viz = all_viz[viz_idx]
            viz = viz.view(-1, 1, viz.shape[2], viz.shape[3])
            image = show_img(make_grid(viz))
            image = 255 * image
            cv2.imwrite(os.path.join(self.activations_path,␣
↪f'sample_layer{all_layers[viz_idx]}.jpg'), image)
```

4 Build and train models

[9]:
```
from prettytable import PrettyTable

def count_parameters(model):
    table = PrettyTable(["Modules", "Parameters"])
    total_params = 0
```

```
    for name, parameter in model.named_parameters():
        if not parameter.requires_grad: continue
        params = parameter.numel()
        table.add_row([name, params])
        total_params+=params
    print(table)
    print(f"Total Trainable Params: {total_params}")
    return total_params
```

[10]:
```
# example_model = models.vit_l_32(pretrained=True)
# count_parameters(example_model)
```

[11]:
```
# vit_l_16 = models.vit_l_16(pretrained=True)
# count_parameters(vit_l_16)
```

[12]:
```
example_model =  models.vit_b_32(pretrained=True) #vit_b_32 = models.
↪vit_b_32(pretrained=True)
count_parameters(example_model)
```

```
+-----------------------------------------------------------------+------------+
|                             Modules                             | Parameters |
+-----------------------------------------------------------------+------------+
|                           class_token                           |    768     |
|                         conv_proj.weight                        |  2359296   |
|                          conv_proj.bias                         |    768     |
|                      encoder.pos_embedding                      |   38400    |
|           encoder.layers.encoder_layer_0.ln_1.weight            |    768     |
|            encoder.layers.encoder_layer_0.ln_1.bias             |    768     |
|  encoder.layers.encoder_layer_0.self_attention.in_proj_weight   |  1769472   |
|   encoder.layers.encoder_layer_0.self_attention.in_proj_bias    |    2304    |
| encoder.layers.encoder_layer_0.self_attention.out_proj.weight   |   589824   |
|  encoder.layers.encoder_layer_0.self_attention.out_proj.bias    |    768     |
|           encoder.layers.encoder_layer_0.ln_2.weight            |    768     |
|            encoder.layers.encoder_layer_0.ln_2.bias             |    768     |
|       encoder.layers.encoder_layer_0.mlp.linear_1.weight        |  2359296   |
|        encoder.layers.encoder_layer_0.mlp.linear_1.bias         |    3072    |
|       encoder.layers.encoder_layer_0.mlp.linear_2.weight        |  2359296   |
|        encoder.layers.encoder_layer_0.mlp.linear_2.bias         |    768     |
|           encoder.layers.encoder_layer_1.ln_1.weight            |    768     |
|            encoder.layers.encoder_layer_1.ln_1.bias             |    768     |
|  encoder.layers.encoder_layer_1.self_attention.in_proj_weight   |  1769472   |
|   encoder.layers.encoder_layer_1.self_attention.in_proj_bias    |    2304    |
| encoder.layers.encoder_layer_1.self_attention.out_proj.weight   |   589824   |
|  encoder.layers.encoder_layer_1.self_attention.out_proj.bias    |    768     |
|           encoder.layers.encoder_layer_1.ln_2.weight            |    768     |
|            encoder.layers.encoder_layer_1.ln_2.bias             |    768     |
|       encoder.layers.encoder_layer_1.mlp.linear_1.weight        |  2359296   |
|        encoder.layers.encoder_layer_1.mlp.linear_1.bias         |    3072    |
```

encoder.layers.encoder_layer_1.mlp.linear_2.weight	2359296
encoder.layers.encoder_layer_1.mlp.linear_2.bias	768
encoder.layers.encoder_layer_2.ln_1.weight	768
encoder.layers.encoder_layer_2.ln_1.bias	768
encoder.layers.encoder_layer_2.self_attention.in_proj_weight	1769472
encoder.layers.encoder_layer_2.self_attention.in_proj_bias	2304
encoder.layers.encoder_layer_2.self_attention.out_proj.weight	589824
encoder.layers.encoder_layer_2.self_attention.out_proj.bias	768
encoder.layers.encoder_layer_2.ln_2.weight	768
encoder.layers.encoder_layer_2.ln_2.bias	768
encoder.layers.encoder_layer_2.mlp.linear_1.weight	2359296
encoder.layers.encoder_layer_2.mlp.linear_1.bias	3072
encoder.layers.encoder_layer_2.mlp.linear_2.weight	2359296
encoder.layers.encoder_layer_2.mlp.linear_2.bias	768
encoder.layers.encoder_layer_3.ln_1.weight	768
encoder.layers.encoder_layer_3.ln_1.bias	768
encoder.layers.encoder_layer_3.self_attention.in_proj_weight	1769472
encoder.layers.encoder_layer_3.self_attention.in_proj_bias	2304
encoder.layers.encoder_layer_3.self_attention.out_proj.weight	589824
encoder.layers.encoder_layer_3.self_attention.out_proj.bias	768
encoder.layers.encoder_layer_3.ln_2.weight	768
encoder.layers.encoder_layer_3.ln_2.bias	768
encoder.layers.encoder_layer_3.mlp.linear_1.weight	2359296
encoder.layers.encoder_layer_3.mlp.linear_1.bias	3072
encoder.layers.encoder_layer_3.mlp.linear_2.weight	2359296
encoder.layers.encoder_layer_3.mlp.linear_2.bias	768
encoder.layers.encoder_layer_4.ln_1.weight	768
encoder.layers.encoder_layer_4.ln_1.bias	768
encoder.layers.encoder_layer_4.self_attention.in_proj_weight	1769472
encoder.layers.encoder_layer_4.self_attention.in_proj_bias	2304
encoder.layers.encoder_layer_4.self_attention.out_proj.weight	589824
encoder.layers.encoder_layer_4.self_attention.out_proj.bias	768
encoder.layers.encoder_layer_4.ln_2.weight	768
encoder.layers.encoder_layer_4.ln_2.bias	768
encoder.layers.encoder_layer_4.mlp.linear_1.weight	2359296
encoder.layers.encoder_layer_4.mlp.linear_1.bias	3072
encoder.layers.encoder_layer_4.mlp.linear_2.weight	2359296
encoder.layers.encoder_layer_4.mlp.linear_2.bias	768
encoder.layers.encoder_layer_5.ln_1.weight	768
encoder.layers.encoder_layer_5.ln_1.bias	768
encoder.layers.encoder_layer_5.self_attention.in_proj_weight	1769472
encoder.layers.encoder_layer_5.self_attention.in_proj_bias	2304
encoder.layers.encoder_layer_5.self_attention.out_proj.weight	589824
encoder.layers.encoder_layer_5.self_attention.out_proj.bias	768
encoder.layers.encoder_layer_5.ln_2.weight	768
encoder.layers.encoder_layer_5.ln_2.bias	768
encoder.layers.encoder_layer_5.mlp.linear_1.weight	2359296
encoder.layers.encoder_layer_5.mlp.linear_1.bias	3072

encoder.layers.encoder_layer_5.mlp.linear_2.weight	2359296
encoder.layers.encoder_layer_5.mlp.linear_2.bias	768
encoder.layers.encoder_layer_6.ln_1.weight	768
encoder.layers.encoder_layer_6.ln_1.bias	768
encoder.layers.encoder_layer_6.self_attention.in_proj_weight	1769472
encoder.layers.encoder_layer_6.self_attention.in_proj_bias	2304
encoder.layers.encoder_layer_6.self_attention.out_proj.weight	589824
encoder.layers.encoder_layer_6.self_attention.out_proj.bias	768
encoder.layers.encoder_layer_6.ln_2.weight	768
encoder.layers.encoder_layer_6.ln_2.bias	768
encoder.layers.encoder_layer_6.mlp.linear_1.weight	2359296
encoder.layers.encoder_layer_6.mlp.linear_1.bias	3072
encoder.layers.encoder_layer_6.mlp.linear_2.weight	2359296
encoder.layers.encoder_layer_6.mlp.linear_2.bias	768
encoder.layers.encoder_layer_7.ln_1.weight	768
encoder.layers.encoder_layer_7.ln_1.bias	768
encoder.layers.encoder_layer_7.self_attention.in_proj_weight	1769472
encoder.layers.encoder_layer_7.self_attention.in_proj_bias	2304
encoder.layers.encoder_layer_7.self_attention.out_proj.weight	589824
encoder.layers.encoder_layer_7.self_attention.out_proj.bias	768
encoder.layers.encoder_layer_7.ln_2.weight	768
encoder.layers.encoder_layer_7.ln_2.bias	768
encoder.layers.encoder_layer_7.mlp.linear_1.weight	2359296
encoder.layers.encoder_layer_7.mlp.linear_1.bias	3072
encoder.layers.encoder_layer_7.mlp.linear_2.weight	2359296
encoder.layers.encoder_layer_7.mlp.linear_2.bias	768
encoder.layers.encoder_layer_8.ln_1.weight	768
encoder.layers.encoder_layer_8.ln_1.bias	768
encoder.layers.encoder_layer_8.self_attention.in_proj_weight	1769472
encoder.layers.encoder_layer_8.self_attention.in_proj_bias	2304
encoder.layers.encoder_layer_8.self_attention.out_proj.weight	589824
encoder.layers.encoder_layer_8.self_attention.out_proj.bias	768
encoder.layers.encoder_layer_8.ln_2.weight	768
encoder.layers.encoder_layer_8.ln_2.bias	768
encoder.layers.encoder_layer_8.mlp.linear_1.weight	2359296
encoder.layers.encoder_layer_8.mlp.linear_1.bias	3072
encoder.layers.encoder_layer_8.mlp.linear_2.weight	2359296
encoder.layers.encoder_layer_8.mlp.linear_2.bias	768
encoder.layers.encoder_layer_9.ln_1.weight	768
encoder.layers.encoder_layer_9.ln_1.bias	768
encoder.layers.encoder_layer_9.self_attention.in_proj_weight	1769472
encoder.layers.encoder_layer_9.self_attention.in_proj_bias	2304
encoder.layers.encoder_layer_9.self_attention.out_proj.weight	589824
encoder.layers.encoder_layer_9.self_attention.out_proj.bias	768
encoder.layers.encoder_layer_9.ln_2.weight	768
encoder.layers.encoder_layer_9.ln_2.bias	768
encoder.layers.encoder_layer_9.mlp.linear_1.weight	2359296
encoder.layers.encoder_layer_9.mlp.linear_1.bias	3072

```
|       encoder.layers.encoder_layer_9.mlp.linear_2.weight       |  2359296   |
|        encoder.layers.encoder_layer_9.mlp.linear_2.bias        |    768     |
|          encoder.layers.encoder_layer_10.ln_1.weight           |    768     |
|           encoder.layers.encoder_layer_10.ln_1.bias            |    768     |
| encoder.layers.encoder_layer_10.self_attention.in_proj_weight  |  1769472   |
|  encoder.layers.encoder_layer_10.self_attention.in_proj_bias   |    2304    |
| encoder.layers.encoder_layer_10.self_attention.out_proj.weight |   589824   |
|  encoder.layers.encoder_layer_10.self_attention.out_proj.bias  |    768     |
|          encoder.layers.encoder_layer_10.ln_2.weight           |    768     |
|           encoder.layers.encoder_layer_10.ln_2.bias            |    768     |
|      encoder.layers.encoder_layer_10.mlp.linear_1.weight       |  2359296   |
|       encoder.layers.encoder_layer_10.mlp.linear_1.bias        |    3072    |
|      encoder.layers.encoder_layer_10.mlp.linear_2.weight       |  2359296   |
|       encoder.layers.encoder_layer_10.mlp.linear_2.bias        |    768     |
|          encoder.layers.encoder_layer_11.ln_1.weight           |    768     |
|           encoder.layers.encoder_layer_11.ln_1.bias            |    768     |
| encoder.layers.encoder_layer_11.self_attention.in_proj_weight  |  1769472   |
|  encoder.layers.encoder_layer_11.self_attention.in_proj_bias   |    2304    |
| encoder.layers.encoder_layer_11.self_attention.out_proj.weight |   589824   |
|  encoder.layers.encoder_layer_11.self_attention.out_proj.bias  |    768     |
|          encoder.layers.encoder_layer_11.ln_2.weight           |    768     |
|           encoder.layers.encoder_layer_11.ln_2.bias            |    768     |
|      encoder.layers.encoder_layer_11.mlp.linear_1.weight       |  2359296   |
|       encoder.layers.encoder_layer_11.mlp.linear_1.bias        |    3072    |
|      encoder.layers.encoder_layer_11.mlp.linear_2.weight       |  2359296   |
|       encoder.layers.encoder_layer_11.mlp.linear_2.bias        |    768     |
|                      encoder.ln.weight                         |    768     |
|                       encoder.ln.bias                          |    768     |
|                      heads.head.weight                         |   768000   |
|                       heads.head.bias                          |    1000    |
+---------------------------------------------------------------+------------+
Total Trainable Params: 88224232
```

[12]:
```
88224232
```

[13]:
```
print(example_model)
```

```
VisionTransformer(
  (conv_proj): Conv2d(3, 768, kernel_size=(32, 32), stride=(32, 32))
  (encoder): Encoder(
    (dropout): Dropout(p=0.0, inplace=False)
    (layers): Sequential(
      (encoder_layer_0): EncoderBlock(
        (ln_1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (self_attention): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=768,
out_features=768, bias=True)
        )
```

```
        (dropout): Dropout(p=0.0, inplace=False)
        (ln_2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (mlp): MLPBlock(
          (linear_1): Linear(in_features=768, out_features=3072, bias=True)
          (act): GELU()
          (dropout_1): Dropout(p=0.0, inplace=False)
          (linear_2): Linear(in_features=3072, out_features=768, bias=True)
          (dropout_2): Dropout(p=0.0, inplace=False)
        )
      )
      (encoder_layer_1): EncoderBlock(
        (ln_1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (self_attention): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=768,
out_features=768, bias=True)
        )
        (dropout): Dropout(p=0.0, inplace=False)
        (ln_2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (mlp): MLPBlock(
          (linear_1): Linear(in_features=768, out_features=3072, bias=True)
          (act): GELU()
          (dropout_1): Dropout(p=0.0, inplace=False)
          (linear_2): Linear(in_features=3072, out_features=768, bias=True)
          (dropout_2): Dropout(p=0.0, inplace=False)
        )
      )
      (encoder_layer_2): EncoderBlock(
        (ln_1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (self_attention): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=768,
out_features=768, bias=True)
        )
        (dropout): Dropout(p=0.0, inplace=False)
        (ln_2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (mlp): MLPBlock(
          (linear_1): Linear(in_features=768, out_features=3072, bias=True)
          (act): GELU()
          (dropout_1): Dropout(p=0.0, inplace=False)
          (linear_2): Linear(in_features=3072, out_features=768, bias=True)
          (dropout_2): Dropout(p=0.0, inplace=False)
        )
      )
      (encoder_layer_3): EncoderBlock(
        (ln_1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (self_attention): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=768,
out_features=768, bias=True)
        )
```

```
        (dropout): Dropout(p=0.0, inplace=False)
        (ln_2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (mlp): MLPBlock(
          (linear_1): Linear(in_features=768, out_features=3072, bias=True)
          (act): GELU()
          (dropout_1): Dropout(p=0.0, inplace=False)
          (linear_2): Linear(in_features=3072, out_features=768, bias=True)
          (dropout_2): Dropout(p=0.0, inplace=False)
        )
      )
      (encoder_layer_4): EncoderBlock(
        (ln_1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (self_attention): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=768,
out_features=768, bias=True)
        )
        (dropout): Dropout(p=0.0, inplace=False)
        (ln_2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (mlp): MLPBlock(
          (linear_1): Linear(in_features=768, out_features=3072, bias=True)
          (act): GELU()
          (dropout_1): Dropout(p=0.0, inplace=False)
          (linear_2): Linear(in_features=3072, out_features=768, bias=True)
          (dropout_2): Dropout(p=0.0, inplace=False)
        )
      )
      (encoder_layer_5): EncoderBlock(
        (ln_1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (self_attention): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=768,
out_features=768, bias=True)
        )
        (dropout): Dropout(p=0.0, inplace=False)
        (ln_2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (mlp): MLPBlock(
          (linear_1): Linear(in_features=768, out_features=3072, bias=True)
          (act): GELU()
          (dropout_1): Dropout(p=0.0, inplace=False)
          (linear_2): Linear(in_features=3072, out_features=768, bias=True)
          (dropout_2): Dropout(p=0.0, inplace=False)
        )
      )
      (encoder_layer_6): EncoderBlock(
        (ln_1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (self_attention): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=768,
out_features=768, bias=True)
        )
```

```
        (dropout): Dropout(p=0.0, inplace=False)
        (ln_2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (mlp): MLPBlock(
          (linear_1): Linear(in_features=768, out_features=3072, bias=True)
          (act): GELU()
          (dropout_1): Dropout(p=0.0, inplace=False)
          (linear_2): Linear(in_features=3072, out_features=768, bias=True)
          (dropout_2): Dropout(p=0.0, inplace=False)
        )
      )
      (encoder_layer_7): EncoderBlock(
        (ln_1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (self_attention): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=768,
out_features=768, bias=True)
        )
        (dropout): Dropout(p=0.0, inplace=False)
        (ln_2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (mlp): MLPBlock(
          (linear_1): Linear(in_features=768, out_features=3072, bias=True)
          (act): GELU()
          (dropout_1): Dropout(p=0.0, inplace=False)
          (linear_2): Linear(in_features=3072, out_features=768, bias=True)
          (dropout_2): Dropout(p=0.0, inplace=False)
        )
      )
      (encoder_layer_8): EncoderBlock(
        (ln_1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (self_attention): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=768,
out_features=768, bias=True)
        )
        (dropout): Dropout(p=0.0, inplace=False)
        (ln_2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (mlp): MLPBlock(
          (linear_1): Linear(in_features=768, out_features=3072, bias=True)
          (act): GELU()
          (dropout_1): Dropout(p=0.0, inplace=False)
          (linear_2): Linear(in_features=3072, out_features=768, bias=True)
          (dropout_2): Dropout(p=0.0, inplace=False)
        )
      )
      (encoder_layer_9): EncoderBlock(
        (ln_1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (self_attention): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=768,
out_features=768, bias=True)
        )
```

```
        (dropout): Dropout(p=0.0, inplace=False)
        (ln_2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (mlp): MLPBlock(
          (linear_1): Linear(in_features=768, out_features=3072, bias=True)
          (act): GELU()
          (dropout_1): Dropout(p=0.0, inplace=False)
          (linear_2): Linear(in_features=3072, out_features=768, bias=True)
          (dropout_2): Dropout(p=0.0, inplace=False)
        )
      )
      (encoder_layer_10): EncoderBlock(
        (ln_1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (self_attention): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=768,
out_features=768, bias=True)
        )
        (dropout): Dropout(p=0.0, inplace=False)
        (ln_2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (mlp): MLPBlock(
          (linear_1): Linear(in_features=768, out_features=3072, bias=True)
          (act): GELU()
          (dropout_1): Dropout(p=0.0, inplace=False)
          (linear_2): Linear(in_features=3072, out_features=768, bias=True)
          (dropout_2): Dropout(p=0.0, inplace=False)
        )
      )
      (encoder_layer_11): EncoderBlock(
        (ln_1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (self_attention): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=768,
out_features=768, bias=True)
        )
        (dropout): Dropout(p=0.0, inplace=False)
        (ln_2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (mlp): MLPBlock(
          (linear_1): Linear(in_features=768, out_features=3072, bias=True)
          (act): GELU()
          (dropout_1): Dropout(p=0.0, inplace=False)
          (linear_2): Linear(in_features=3072, out_features=768, bias=True)
          (dropout_2): Dropout(p=0.0, inplace=False)
        )
      )
    )
    (ln): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
  )
  (heads): Sequential(
    (head): Linear(in_features=768, out_features=1000, bias=True)
  )
```

```
)
```

[14]:
```
print(example_model.heads)
```

```
Sequential(
  (head): Linear(in_features=768, out_features=1000, bias=True)
)
```

[15]:
```
for name, param in example_model.named_parameters():
    number = name.split('.')
    print(number)
    #if number[0] == 'layers':
        #print(number[1].split('_')[2])
        #print(number[2])
```

```
['class_token']
['conv_proj', 'weight']
['conv_proj', 'bias']
['encoder', 'pos_embedding']
['encoder', 'layers', 'encoder_layer_0', 'ln_1', 'weight']
['encoder', 'layers', 'encoder_layer_0', 'ln_1', 'bias']
['encoder', 'layers', 'encoder_layer_0', 'self_attention', 'in_proj_weight']
['encoder', 'layers', 'encoder_layer_0', 'self_attention', 'in_proj_bias']
['encoder', 'layers', 'encoder_layer_0', 'self_attention', 'out_proj', 'weight']
['encoder', 'layers', 'encoder_layer_0', 'self_attention', 'out_proj', 'bias']
['encoder', 'layers', 'encoder_layer_0', 'ln_2', 'weight']
['encoder', 'layers', 'encoder_layer_0', 'ln_2', 'bias']
['encoder', 'layers', 'encoder_layer_0', 'mlp', 'linear_1', 'weight']
['encoder', 'layers', 'encoder_layer_0', 'mlp', 'linear_1', 'bias']
['encoder', 'layers', 'encoder_layer_0', 'mlp', 'linear_2', 'weight']
['encoder', 'layers', 'encoder_layer_0', 'mlp', 'linear_2', 'bias']
['encoder', 'layers', 'encoder_layer_1', 'ln_1', 'weight']
['encoder', 'layers', 'encoder_layer_1', 'ln_1', 'bias']
['encoder', 'layers', 'encoder_layer_1', 'self_attention', 'in_proj_weight']
['encoder', 'layers', 'encoder_layer_1', 'self_attention', 'in_proj_bias']
['encoder', 'layers', 'encoder_layer_1', 'self_attention', 'out_proj', 'weight']
['encoder', 'layers', 'encoder_layer_1', 'self_attention', 'out_proj', 'bias']
['encoder', 'layers', 'encoder_layer_1', 'ln_2', 'weight']
['encoder', 'layers', 'encoder_layer_1', 'ln_2', 'bias']
['encoder', 'layers', 'encoder_layer_1', 'mlp', 'linear_1', 'weight']
['encoder', 'layers', 'encoder_layer_1', 'mlp', 'linear_1', 'bias']
['encoder', 'layers', 'encoder_layer_1', 'mlp', 'linear_2', 'weight']
['encoder', 'layers', 'encoder_layer_1', 'mlp', 'linear_2', 'bias']
['encoder', 'layers', 'encoder_layer_2', 'ln_1', 'weight']
['encoder', 'layers', 'encoder_layer_2', 'ln_1', 'bias']
['encoder', 'layers', 'encoder_layer_2', 'self_attention', 'in_proj_weight']
['encoder', 'layers', 'encoder_layer_2', 'self_attention', 'in_proj_bias']
['encoder', 'layers', 'encoder_layer_2', 'self_attention', 'out_proj', 'weight']
['encoder', 'layers', 'encoder_layer_2', 'self_attention', 'out_proj', 'bias']
```

```
['encoder', 'layers', 'encoder_layer_2', 'ln_2', 'weight']
['encoder', 'layers', 'encoder_layer_2', 'ln_2', 'bias']
['encoder', 'layers', 'encoder_layer_2', 'mlp', 'linear_1', 'weight']
['encoder', 'layers', 'encoder_layer_2', 'mlp', 'linear_1', 'bias']
['encoder', 'layers', 'encoder_layer_2', 'mlp', 'linear_2', 'weight']
['encoder', 'layers', 'encoder_layer_2', 'mlp', 'linear_2', 'bias']
['encoder', 'layers', 'encoder_layer_3', 'ln_1', 'weight']
['encoder', 'layers', 'encoder_layer_3', 'ln_1', 'bias']
['encoder', 'layers', 'encoder_layer_3', 'self_attention', 'in_proj_weight']
['encoder', 'layers', 'encoder_layer_3', 'self_attention', 'in_proj_bias']
['encoder', 'layers', 'encoder_layer_3', 'self_attention', 'out_proj', 'weight']
['encoder', 'layers', 'encoder_layer_3', 'self_attention', 'out_proj', 'bias']
['encoder', 'layers', 'encoder_layer_3', 'ln_2', 'weight']
['encoder', 'layers', 'encoder_layer_3', 'ln_2', 'bias']
['encoder', 'layers', 'encoder_layer_3', 'mlp', 'linear_1', 'weight']
['encoder', 'layers', 'encoder_layer_3', 'mlp', 'linear_1', 'bias']
['encoder', 'layers', 'encoder_layer_3', 'mlp', 'linear_2', 'weight']
['encoder', 'layers', 'encoder_layer_3', 'mlp', 'linear_2', 'bias']
['encoder', 'layers', 'encoder_layer_4', 'ln_1', 'weight']
['encoder', 'layers', 'encoder_layer_4', 'ln_1', 'bias']
['encoder', 'layers', 'encoder_layer_4', 'self_attention', 'in_proj_weight']
['encoder', 'layers', 'encoder_layer_4', 'self_attention', 'in_proj_bias']
['encoder', 'layers', 'encoder_layer_4', 'self_attention', 'out_proj', 'weight']
['encoder', 'layers', 'encoder_layer_4', 'self_attention', 'out_proj', 'bias']
['encoder', 'layers', 'encoder_layer_4', 'ln_2', 'weight']
['encoder', 'layers', 'encoder_layer_4', 'ln_2', 'bias']
['encoder', 'layers', 'encoder_layer_4', 'mlp', 'linear_1', 'weight']
['encoder', 'layers', 'encoder_layer_4', 'mlp', 'linear_1', 'bias']
['encoder', 'layers', 'encoder_layer_4', 'mlp', 'linear_2', 'weight']
['encoder', 'layers', 'encoder_layer_4', 'mlp', 'linear_2', 'bias']
['encoder', 'layers', 'encoder_layer_5', 'ln_1', 'weight']
['encoder', 'layers', 'encoder_layer_5', 'ln_1', 'bias']
['encoder', 'layers', 'encoder_layer_5', 'self_attention', 'in_proj_weight']
['encoder', 'layers', 'encoder_layer_5', 'self_attention', 'in_proj_bias']
['encoder', 'layers', 'encoder_layer_5', 'self_attention', 'out_proj', 'weight']
['encoder', 'layers', 'encoder_layer_5', 'self_attention', 'out_proj', 'bias']
['encoder', 'layers', 'encoder_layer_5', 'ln_2', 'weight']
['encoder', 'layers', 'encoder_layer_5', 'ln_2', 'bias']
['encoder', 'layers', 'encoder_layer_5', 'mlp', 'linear_1', 'weight']
['encoder', 'layers', 'encoder_layer_5', 'mlp', 'linear_1', 'bias']
['encoder', 'layers', 'encoder_layer_5', 'mlp', 'linear_2', 'weight']
['encoder', 'layers', 'encoder_layer_5', 'mlp', 'linear_2', 'bias']
['encoder', 'layers', 'encoder_layer_6', 'ln_1', 'weight']
['encoder', 'layers', 'encoder_layer_6', 'ln_1', 'bias']
['encoder', 'layers', 'encoder_layer_6', 'self_attention', 'in_proj_weight']
['encoder', 'layers', 'encoder_layer_6', 'self_attention', 'in_proj_bias']
['encoder', 'layers', 'encoder_layer_6', 'self_attention', 'out_proj', 'weight']
['encoder', 'layers', 'encoder_layer_6', 'self_attention', 'out_proj', 'bias']
```

```
['encoder', 'layers', 'encoder_layer_6', 'ln_2', 'weight']
['encoder', 'layers', 'encoder_layer_6', 'ln_2', 'bias']
['encoder', 'layers', 'encoder_layer_6', 'mlp', 'linear_1', 'weight']
['encoder', 'layers', 'encoder_layer_6', 'mlp', 'linear_1', 'bias']
['encoder', 'layers', 'encoder_layer_6', 'mlp', 'linear_2', 'weight']
['encoder', 'layers', 'encoder_layer_6', 'mlp', 'linear_2', 'bias']
['encoder', 'layers', 'encoder_layer_7', 'ln_1', 'weight']
['encoder', 'layers', 'encoder_layer_7', 'ln_1', 'bias']
['encoder', 'layers', 'encoder_layer_7', 'self_attention', 'in_proj_weight']
['encoder', 'layers', 'encoder_layer_7', 'self_attention', 'in_proj_bias']
['encoder', 'layers', 'encoder_layer_7', 'self_attention', 'out_proj', 'weight']
['encoder', 'layers', 'encoder_layer_7', 'self_attention', 'out_proj', 'bias']
['encoder', 'layers', 'encoder_layer_7', 'ln_2', 'weight']
['encoder', 'layers', 'encoder_layer_7', 'ln_2', 'bias']
['encoder', 'layers', 'encoder_layer_7', 'mlp', 'linear_1', 'weight']
['encoder', 'layers', 'encoder_layer_7', 'mlp', 'linear_1', 'bias']
['encoder', 'layers', 'encoder_layer_7', 'mlp', 'linear_2', 'weight']
['encoder', 'layers', 'encoder_layer_7', 'mlp', 'linear_2', 'bias']
['encoder', 'layers', 'encoder_layer_8', 'ln_1', 'weight']
['encoder', 'layers', 'encoder_layer_8', 'ln_1', 'bias']
['encoder', 'layers', 'encoder_layer_8', 'self_attention', 'in_proj_weight']
['encoder', 'layers', 'encoder_layer_8', 'self_attention', 'in_proj_bias']
['encoder', 'layers', 'encoder_layer_8', 'self_attention', 'out_proj', 'weight']
['encoder', 'layers', 'encoder_layer_8', 'self_attention', 'out_proj', 'bias']
['encoder', 'layers', 'encoder_layer_8', 'ln_2', 'weight']
['encoder', 'layers', 'encoder_layer_8', 'ln_2', 'bias']
['encoder', 'layers', 'encoder_layer_8', 'mlp', 'linear_1', 'weight']
['encoder', 'layers', 'encoder_layer_8', 'mlp', 'linear_1', 'bias']
['encoder', 'layers', 'encoder_layer_8', 'mlp', 'linear_2', 'weight']
['encoder', 'layers', 'encoder_layer_8', 'mlp', 'linear_2', 'bias']
['encoder', 'layers', 'encoder_layer_9', 'ln_1', 'weight']
['encoder', 'layers', 'encoder_layer_9', 'ln_1', 'bias']
['encoder', 'layers', 'encoder_layer_9', 'self_attention', 'in_proj_weight']
['encoder', 'layers', 'encoder_layer_9', 'self_attention', 'in_proj_bias']
['encoder', 'layers', 'encoder_layer_9', 'self_attention', 'out_proj', 'weight']
['encoder', 'layers', 'encoder_layer_9', 'self_attention', 'out_proj', 'bias']
['encoder', 'layers', 'encoder_layer_9', 'ln_2', 'weight']
['encoder', 'layers', 'encoder_layer_9', 'ln_2', 'bias']
['encoder', 'layers', 'encoder_layer_9', 'mlp', 'linear_1', 'weight']
['encoder', 'layers', 'encoder_layer_9', 'mlp', 'linear_1', 'bias']
['encoder', 'layers', 'encoder_layer_9', 'mlp', 'linear_2', 'weight']
['encoder', 'layers', 'encoder_layer_9', 'mlp', 'linear_2', 'bias']
['encoder', 'layers', 'encoder_layer_10', 'ln_1', 'weight']
['encoder', 'layers', 'encoder_layer_10', 'ln_1', 'bias']
['encoder', 'layers', 'encoder_layer_10', 'self_attention', 'in_proj_weight']
['encoder', 'layers', 'encoder_layer_10', 'self_attention', 'in_proj_bias']
['encoder', 'layers', 'encoder_layer_10', 'self_attention', 'out_proj',
'weight']
```

```
['encoder', 'layers', 'encoder_layer_10', 'self_attention', 'out_proj', 'bias']
['encoder', 'layers', 'encoder_layer_10', 'ln_2', 'weight']
['encoder', 'layers', 'encoder_layer_10', 'ln_2', 'bias']
['encoder', 'layers', 'encoder_layer_10', 'mlp', 'linear_1', 'weight']
['encoder', 'layers', 'encoder_layer_10', 'mlp', 'linear_1', 'bias']
['encoder', 'layers', 'encoder_layer_10', 'mlp', 'linear_2', 'weight']
['encoder', 'layers', 'encoder_layer_10', 'mlp', 'linear_2', 'bias']
['encoder', 'layers', 'encoder_layer_11', 'ln_1', 'weight']
['encoder', 'layers', 'encoder_layer_11', 'ln_1', 'bias']
['encoder', 'layers', 'encoder_layer_11', 'self_attention', 'in_proj_weight']
['encoder', 'layers', 'encoder_layer_11', 'self_attention', 'in_proj_bias']
['encoder', 'layers', 'encoder_layer_11', 'self_attention', 'out_proj',
'weight']
['encoder', 'layers', 'encoder_layer_11', 'self_attention', 'out_proj', 'bias']
['encoder', 'layers', 'encoder_layer_11', 'ln_2', 'weight']
['encoder', 'layers', 'encoder_layer_11', 'ln_2', 'bias']
['encoder', 'layers', 'encoder_layer_11', 'mlp', 'linear_1', 'weight']
['encoder', 'layers', 'encoder_layer_11', 'mlp', 'linear_1', 'bias']
['encoder', 'layers', 'encoder_layer_11', 'mlp', 'linear_2', 'weight']
['encoder', 'layers', 'encoder_layer_11', 'mlp', 'linear_2', 'bias']
['encoder', 'ln', 'weight']
['encoder', 'ln', 'bias']
['heads', 'head', 'weight']
['heads', 'head', 'bias']
```

[18]:
```python
class TransferViT(nn.Module):
    def __init__(self):
        super().__init__()
        self.vit = models.vit_b_32(pretrained=True)
        #self.conv_layer = self.get_conv_proj()
        self.vit.heads = self.get_fc_layers()
        #self.vit = self.get_ViT_encoder()
        #self.fc_model = self.get_fc_layers()
        self.activate_training_layers()

    def activate_training_layers(self):
#         for name, param in self.conv_layer.named_parameters():
#             # for all of these layers set param.requires_grad as True
#             param.requires_grad = False

        for name, param in self.vit.named_parameters():
            number = name.split('.')
            # for all layers except the last conv layer, set param.
 ↪requires_grad = False
            if number[0] == 'heads':
#                 if number[1].split('_')[2] == 11 and number[2] == 'mlp':
#                     param.requires_grad = True
```

```
#                 else:
                    param.requires_grad = True
                    print('required_grad = True', number)
                else:
                    param.requires_grad = False
                    print('required_grad = False', number)

        #for name, param in self.vit.heads.named_parameters():
            # for all of these layers set param.requires_grad as True


    def get_fc_layers(self):
        return nn.Sequential(
            nn.Dropout(p=0.5, inplace=False),
            nn.Linear(in_features=768, out_features=512, bias=True),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5, inplace=False),
            nn.Linear(in_features=512, out_features=128, bias=True),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=128, out_features=15, bias=True),
        )

    def forward(self, x):
        #x = self.conv_layer(x)
        x = self.vit(x)
        #x = torch.flatten(x, 1)
        #x = self.fc_model(x)  #call fully connected layers

        return x
```

[19]:
```
# Train a transfer learning model with Alexnet
name = 'TransferViT'
classes = [i for i in range(15)]
transforms = get_transform('TransferViT')
dataloaders = {'train_image_paths': train_image_paths, 'train_labels' :␣
↪train_labels, 'valid_image_paths': valid_image_paths, 'valid_labels':
↪valid_labels, 'transforms':transforms, 'sampler':sampler}
parameters = {'lr': 0.001, 'epochs' : 5, 'batch_size':32, 'shuffle':False,␣
↪'class_names':classes}

model = TransferViT()
classifier = Classifier(name, model, dataloaders, parameters, use_cuda=True)
classifier.train()
```

```
required_grad = False ['class_token']
required_grad = False ['conv_proj', 'weight']
required_grad = False ['conv_proj', 'bias']
required_grad = False ['encoder', 'pos_embedding']
```

```
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_0', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_1', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'self_attention',
'out_proj', 'weight']
```

```
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_2', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_3', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'mlp',
'linear_1', 'bias']
```

```
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_4', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_5', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_6', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'self_attention',
'in_proj_weight']
```

```
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_7', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_8', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'ln_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'self_attention',
'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'self_attention',
'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'self_attention',
'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'self_attention',
'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'ln_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'ln_2', 'bias']
```

```
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_9', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'mlp',
'linear_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_10', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'ln_1',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'ln_1', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11',
'self_attention', 'in_proj_weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11',
'self_attention', 'in_proj_bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11',
'self_attention', 'out_proj', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11',
'self_attention', 'out_proj', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'ln_2',
'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'ln_2', 'bias']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'mlp',
'linear_1', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'mlp',
'linear_1', 'bias']
```

```
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'mlp',
'linear_2', 'weight']
required_grad = False ['encoder', 'layers', 'encoder_layer_11', 'mlp',
'linear_2', 'bias']
required_grad = False ['encoder', 'ln', 'weight']
required_grad = False ['encoder', 'ln', 'bias']
required_grad = True ['heads', '1', 'weight']
required_grad = True ['heads', '1', 'bias']
required_grad = True ['heads', '4', 'weight']
required_grad = True ['heads', '4', 'bias']
required_grad = True ['heads', '6', 'weight']
required_grad = True ['heads', '6', 'bias']

 37%|
| 1999/5377 [09:14<14:58,  3.76it/s]

[1,  2000] loss: 0.971

 74%|
| 3999/5377 [18:01<05:54,  3.88it/s]

[1,  4000] loss: 0.746

100%|
  | 5377/5377 [24:05<00:00,  3.72it/s]

Epoch: 1 Training Epoch Accuracy:72.2659769246418
Epoch: 1 Validation Epoch Accuracy:71.09178833813819
Epoch: 1 Correct predictions {0: 52, 1: 1913, 2: 4392, 3: 7496, 4: 178, 5: 4656,
6: 131, 7: 1902, 8: 242, 9: 508, 10: 7544, 11: 867, 12: 397, 13: 283, 14: 17}
Epoch: 1 Total predictions {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470,
6: 158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
Epoch: 1 Correct predictions {0: 52, 1: 1913, 2: 4392, 3: 7496, 4: 178, 5: 4656,
6: 131, 7: 1902, 8: 242, 9: 508, 10: 7544, 11: 867, 12: 397, 13: 283, 14: 17}
Epoch: 1 Total predictions {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470,
6: 158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
Fininsh Trainig Epoch 0 ! Time used: 1817.4088490009308

 37%|
| 1999/5377 [08:36<14:13,  3.96it/s]

[2,  2000] loss: 0.665

 74%|
| 3999/5377 [17:15<06:03,  3.79it/s]

[2,  4000] loss: 0.645

100%|
  | 5377/5377 [23:30<00:00,  3.81it/s]

Epoch: 2 Training Epoch Accuracy:77.4204423261356
Epoch: 2 Validation Epoch Accuracy:75.8881242443969
Epoch: 2 Correct predictions {0: 51, 1: 2023, 2: 3578, 3: 9070, 4: 186, 5: 4650,
```

```
6: 135, 7: 1617, 8: 242, 9: 513, 10: 8801, 11: 1078, 12: 398, 13: 282, 14: 17}
Epoch: 2 Total predictions {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470,
6: 158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
Epoch: 2 Correct predictions {0: 51, 1: 2023, 2: 3578, 3: 9070, 4: 186, 5: 4650,
6: 135, 7: 1617, 8: 242, 9: 513, 10: 8801, 11: 1078, 12: 398, 13: 282, 14: 17}
Epoch: 2 Total predictions {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470,
6: 158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
Fininsh Trainig Epoch 1 ! Time used: 1789.2193973064423

 37%|
| 1999/5377 [08:57<14:58,  3.76it/s]

[3,  2000] loss: 0.618

 74%|
| 3999/5377 [17:57<06:24,  3.58it/s]

[3,  4000] loss: 0.619

100%|
  | 5377/5377 [24:06<00:00,  3.72it/s]

Epoch: 3 Training Epoch Accuracy:78.58583510128165
Epoch: 3 Validation Epoch Accuracy:75.89742397470474
Epoch: 3 Correct predictions {0: 52, 1: 1922, 2: 4254, 3: 8460, 4: 182, 5: 4762,
6: 135, 7: 1757, 8: 242, 9: 535, 10: 8705, 11: 941, 12: 398, 13: 283, 14: 17}
Epoch: 3 Total predictions {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470,
6: 158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
Epoch: 3 Correct predictions {0: 52, 1: 1922, 2: 4254, 3: 8460, 4: 182, 5: 4762,
6: 135, 7: 1757, 8: 242, 9: 535, 10: 8705, 11: 941, 12: 398, 13: 283, 14: 17}
Epoch: 3 Total predictions {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470,
6: 158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
Fininsh Trainig Epoch 2 ! Time used: 1821.3886678218842

 37%|
| 1999/5377 [08:57<15:13,  3.70it/s]

[4,  2000] loss: 0.591

 74%|
| 3999/5377 [17:51<06:06,  3.76it/s]

[4,  4000] loss: 0.583

100%|
  | 5377/5377 [23:58<00:00,  3.74it/s]

Epoch: 4 Training Epoch Accuracy:79.56232381063094
Epoch: 4 Validation Epoch Accuracy:75.3278154933507
Epoch: 4 Correct predictions {0: 51, 1: 2077, 2: 3364, 3: 9201, 4: 188, 5: 4734,
6: 127, 7: 2079, 8: 245, 9: 537, 10: 8197, 11: 900, 12: 400, 13: 283, 14: 17}
Epoch: 4 Total predictions {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470,
6: 158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
Epoch: 4 Correct predictions {0: 51, 1: 2077, 2: 3364, 3: 9201, 4: 188, 5: 4734,
```

```
6: 127, 7: 2079, 8: 245, 9: 537, 10: 8197, 11: 900, 12: 400, 13: 283, 14: 17}
Epoch: 4 Total predictions {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470,
6: 158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
Fininsh Trainig Epoch 3 ! Time used: 1815.4485664367676

 37%|
| 1999/5377 [08:55<15:01,  3.75it/s]

[5,  2000] loss: 0.576

 74%|
| 3999/5377 [17:47<06:04,  3.78it/s]

[5,  4000] loss: 0.566

100%|
  | 5377/5377 [23:55<00:00,  3.75it/s]

Epoch: 5 Training Epoch Accuracy:80.2493533668517
Epoch: 5 Validation Epoch Accuracy:77.74109550823026
Epoch: 5 Correct predictions {0: 51, 1: 2044, 2: 3659, 3: 9166, 4: 186, 5: 4695,
6: 131, 7: 1904, 8: 244, 9: 530, 10: 8993, 11: 1135, 12: 399, 13: 284, 14: 17}
Epoch: 5 Total predictions {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470,
6: 158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
Epoch: 5 Correct predictions {0: 51, 1: 2044, 2: 3659, 3: 9166, 4: 186, 5: 4695,
6: 131, 7: 1904, 8: 244, 9: 530, 10: 8993, 11: 1135, 12: 399, 13: 284, 14: 17}
Epoch: 5 Total predictions {0: 52, 1: 2177, 2: 5799, 3: 10405, 4: 213, 5: 5470,
6: 158, 7: 2843, 8: 247, 9: 587, 10: 11135, 11: 3216, 12: 404, 13: 288, 14: 18}
Fininsh Trainig Epoch 4 ! Time used: 1813.407681941986
Done training!
```

5 Data Augmentations for scarse data

[ ]:
```
# find out the ratio of different labels/data

# we decide to augment the scarser data in the following scheme: label 0, +␣
↪400, label 14 + 500, label 6 + 200
```