A note about environment

Mainly for me, just to remenber the correct settings

I have to create an environment using conda (with python 3.6.2, pip and other packages) and then install the required torch packages using pip (installing torch 1.9.1, torchvision 0.10.1 and torchaudio 0.9.1).
The commands was

```
 conda create --prefix PATH_TO_ENVIRONMENT python=3.6.2 pip numpy matplotlib pandas jupyterlab tqdm opencv scikit-image  pip3 install torch==1.10.0+cu102 torchvision==0.11.1+cu102 torchaudio===0.10.0+cu102 -f https://download.pytorch.org/whl/cu102/torch_stable.html
```

The `--prefix` in the creation command was used to create the environment in the user folder. The version of torchvision installed is 0.10.1 so I can use the `torchvision.io` module.

Convolutional Neural Networks

Project: Write an Algorithm for a Dog Identification App

In this notebook, some template code has already been provided for you, and you will need to implement additional functionality to successfully complete this project. You will not need to modify the included code beyond what is requested. Sections that begin with **'(IMPLEMENTATION)'** in the header indicate that the following block of code will require additional functionality which you must provide. Instructions will be provided for each section, and the specifics of the implementation are marked in the code block with a 'TODO' statement. Please be sure to read the instructions carefully!

> **Note**: Once you have completed all of the code implementations, you need to finalize your work by exporting the Jupyter Notebook as an HTML document. Before exporting the notebook to html, all of the code cells need to have been run so that reviewers can see the final implementation and output. You can then export the notebook by using the menu above and navigating to **File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation. Each section where you will answer a question is preceded by a **'Question X'** header. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**. Your project submission will be evaluated based on your answers to each of the questions and the implementation you provide.

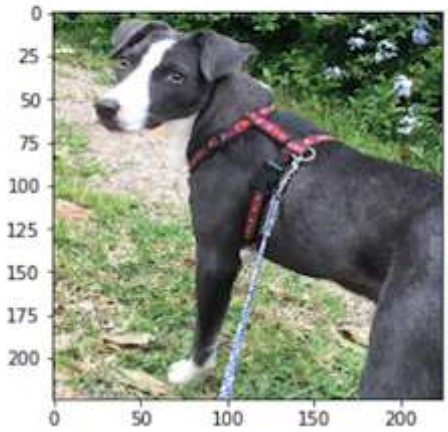
> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. Markdown cells can be edited by double-clicking the cell to enter edit mode.

The rubric contains *optional* "Stand Out Suggestions" for enhancing the project beyond the minimum requirements. If you decide to pursue the "Stand Out Suggestions", you should include the code in this Jupyter notebook.

Why We're Here

In this notebook, you will make the first steps towards developing an algorithm that could be used as part of a mobile or web app. At the end of this project, your code will accept any user-supplied image as input. If a dog is detected in the image, it will provide an estimate of the dog's breed. If a human is detected, it will provide an estimate of the dog breed that is most resembling. The image below displays potential sample output of your finished project (... but we expect that each student's algorithm will behave differently!).

```
hello, dog!
your predicted breed is ...
American Staffordshire terrier
```



In this real-world setting, you will need to piece together a series of models to perform different tasks; for instance, the algorithm that detects humans in an image will be different from the CNN that infers dog breed. There are many points of possible failure, and no perfect algorithm exists. Your imperfect solution will nonetheless create a fun user experience!

The Road Ahead

We break the notebook into separate steps. Feel free to use the links below to navigate the notebook.

Step 0: Import Datasets

Make sure that you've downloaded the required human and dog datasets:

Note: if you are using the Udacity workspace, you *DO NOT* need to re-download these - they can be found in the `/data` folder as noted in the cell below.

- Download the dog dataset. Unzip the folder and place it in this project's home directory, at the location `/dog_images`.
- Download the human dataset. Unzip the folder and place it in the home directory, at location `/lfw`.

Note: If you are using a Windows machine, you are encouraged to use 7zip to extract the folder.

In the code cell below, we save the file paths for both the human (LFW) dataset and dog dataset in the numpy arrays `human_files` and `dog_files`.

In [ ]:
```
import numpy as np
from glob import glob

# load filenames for human and dog images
human_files = np.array(glob("/data/lfw/*/*"))
if len(human_files) == 0: # Maybe it's running local, so we'll try another path
        human_files = np.array(glob("data/lfw/*/*"))

dog_files = np.array(glob("/data/dog_images/*/*/*"))
if len(dog_files) == 0: # Maybe it's running local, so we'll try another path
        dog_files = np.array(glob("data/dog_images/*/*/*"))

# print number of images in each dataset
print('There are %d total human images.' % len(human_files))
print('There are %d total dog images.' % len(dog_files))
```

```
There are 1556 total human images.
There are 8350 total dog images.
```

Step 1: Detect Humans

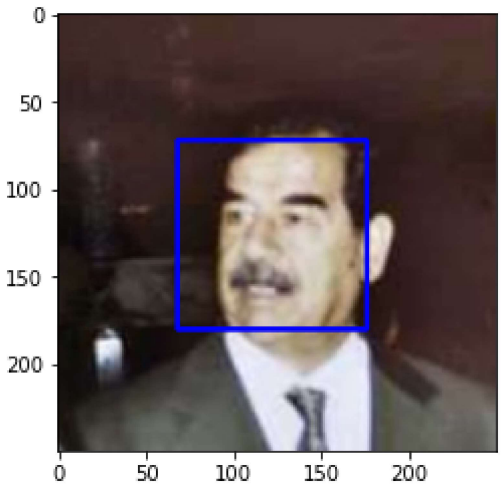
In this section, we use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.

OpenCV provides many pre-trained face detectors, stored as XML files on github. We have downloaded one of these detectors and stored it in the `haarcascades` directory. In the next code cell, we demonstrate how to use this detector to find human faces in a sample image.

In [ ]:
```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

# extract pre-trained face detector
face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml')

# load color (BGR) image
img = cv2.imread(human_files[0])
# convert BGR image to grayscale
```

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# find faces in image
faces = face_cascade.detectMultiScale(gray)

# print number of faces detected in the image
print('Number of faces detected:', len(faces))

# get bounding box for each detected face
for (x,y,w,h) in faces:
    # add bounding box to color image
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

# convert BGR image to RGB for plotting
cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# display the image, along with bounding box
plt.imshow(cv_rgb)
plt.show()
```

```
Number of faces detected: 1
```



Before using any of the face detectors, it is standard procedure to convert the images to grayscale. The `detectMultiScale` function executes the classifier stored in `face_cascade` and takes the grayscale image as a parameter.

In the above code, `faces` is a numpy array of detected faces, where each row corresponds to a detected face. Each detected face is a 1D array with four entries that specifies the bounding box of the detected face. The first two entries in the array (extracted in the above code as `x` and `y`) specify the horizontal and vertical positions of the top left corner of the bounding box. The last two entries in the array (extracted here as `w` and `h`) specify the width and height of the box.

Write a Human Face Detector

We can use this procedure to write a function that returns `True` if a human face is detected in an image and `False` otherwise. This function, aptly named `face_detector`, takes a string-valued file path to an image as input and appears in the code block below.

In [ ]:

```
# returns "True" if face is detected in image stored at img_path
def face_detector(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
    faces = face_cascade.detectMultiScale(gray)
    return len(faces) > 0
```

(IMPLEMENTATION) Assess the Human Face Detector

Question 1: Use the code cell below to test the performance of the `face_detector` function.

- What percentage of the first 100 images in `human_files` have a detected human face?
- What percentage of the first 100 images in `dog_files` have a detected human face?

Ideally, we would like 100% of human images with a detected face and 0% of dog images with a detected face. You will see that our algorithm falls short of this goal, but still gives acceptable performance. We extract the file paths for the first 100 images from each of the datasets and store them in the numpy arrays `human_files_short` and `dog_files_short`.

Answer: (You can print out your results and/or write your percentages in this cell)

I've obtainined a 100% percentage of detecting faces in the Humans dataset and a 18% percentage of detecting faces in the Dogs dataset

In [ ]:
```
from tqdm import tqdm

human_files_short = human_files[:100]
dog_files_short = dog_files[:100]

#-#-# Do NOT modify the code above this line. #-#-#

## TODO: Test the performance of the face_detector algorithm
## on the images in human_files_short and dog_files_short.
def detect_faces(images_paths):
        face_detector_results = [face_detector(image_path) for image_path in tqdm(im
        images_with_faces = [1 for result in face_detector_results if result == True
        return len(images_with_faces) / len(images_paths)

faces_in_humans = detect_faces(human_files_short)
faces_in_dogs = detect_faces(dog_files_short)

print(f"Images with a face in the Humans dataset: {faces_in_humans * 100} % - Images
```

```
100%|██████████| 100/100 [00:01<00:00, 93.79it/s]
100%|██████████| 100/100 [00:05<00:00, 17.85it/s]
Images with a face in the Humans dataset: 100.0 % - Images with a face in the Dogs d
ataset: 18.0 %
```

We suggest the face detector from OpenCV as a potential way to detect human images in your algorithm, but you are free to explore other approaches, especially approaches that make use of deep learning :). Please use the code cell below to design and test your own face detection algorithm. If you decide to pursue this *optional* task, report performance on `human_files_short` and `dog_files_short`.

In [ ]:
```
### (Optional)
### TODO: Test performance of another face detection algorithm.
### Feel free to use as many code cells as needed.

from skimage import data
from skimage.feature import Cascade
```

```
def face_detector_with_scikit(img_path):
        image = cv2.imread(img_path)

        model_data = data.lbp_frontal_face_cascade_filename()
        model_scikit = Cascade(model_data)

        faces_detected = model_scikit.detect_multi_scale(image, scale_factor=1.2, st
        return len(faces_detected) >= 1

def detect_faces_with_scikit(images_paths):
        face_detector_results = [face_detector_with_scikit(image_path) for image_pat
        images_with_faces = [1 for result in face_detector_results if result == True
        return len(images_with_faces) / len(images_paths)

faces_in_humans_with_scikit = detect_faces_with_scikit(human_files_short)
faces_in_dogs_with_scikit = detect_faces_with_scikit(dog_files_short)

print(f"Detection with Scikit Learn: Images with a face in the Humans dataset: {face
```

```
100%|██████████| 100/100 [00:00<00:00, 100.78it/s]
100%|██████████| 100/100 [00:07<00:00, 13.91it/s]
Detection with Scikit Learn: Images with a face in the Humans dataset: 78.0 % - Imag
es with a face in the Dogs dataset: 8.0 %
```

Step 2: Detect Dogs

In this section, we use a pre-trained model to detect dogs in images.

Obtain Pre-trained VGG-16 Model

The code cell below downloads the VGG-16 model, along with weights that have been trained on ImageNet, a very large, very popular dataset used for image classification and other vision tasks. ImageNet contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories.

In [ ]:

```
import torch
import torchvision.models as models

# define VGG16 model
VGG16 = models.vgg16(pretrained=True)

# check if CUDA is available
use_cuda = torch.cuda.is_available()

# move model to GPU if CUDA is available
if use_cuda:
    VGG16 = VGG16.cuda()
```

Given an image, this pre-trained VGG-16 model returns a prediction (derived from the 1000 possible categories in ImageNet) for the object that is contained in the image.

(IMPLEMENTATION) Making Predictions with a Pre-trained Model

In the next code cell, you will write a function that accepts a path to an image (such as `'dogImages/train/001.Affenpinscher/Affenpinscher_00001.jpg'`) as input and returns the index corresponding to the ImageNet class that is predicted by the pre-trained VGG-16 model. The output should always be an integer between 0 and 999, inclusive.

Before writing the function, make sure that you take the time to learn how to appropriately pre-process tensors for pre-trained models in the PyTorch documentation.

In [ ]:

```
from PIL import Image
import torchvision.transforms as transforms

def VGG16_predict(img_path):
    '''
    Use pre-trained VGG-16 model to obtain index corresponding to
    predicted ImageNet class for image at specified path

    Args:
        img_path: path to an image

    Returns:
        Index corresponding to VGG-16 model's prediction
    '''

    ## TODO: Complete the function.
    ## Load and pre-process an image from the given img_path
    ## Return the *index* of the predicted class for that image
    image = Image.open(img_path)

    transformations = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])]

    image = transformations(image) # apply transformations to the image

    image = torch.unsqueeze(image, 0) # add a dimension because the model expect a b

    if use_cuda:
        image = image.cuda()

    VGG16.eval() # change to evaluation mode, just to be sure (I think that VGG does

    predictions = VGG16(image)

    predictions = torch.squeeze(predictions, 0) # remove the artificial dimension ad

    index = torch.argmax(predictions)

    if use_cuda:
        index = index.cpu()

    return index.item() # predicted class index
```

Imagenet classes Load

I'll load Imagenet classes just to know the meaning of the class (can be useful for other project too).

In [ ]:
```
import pandas as pd
classes = pd.read_csv("imagenet_classes.txt")
classes.head()
```

Out[ ]:

	class_id	name	description
0	0	tench	tench, Tinca tinca
1	1	goldfish	goldfish, Carassius auratus
2	2	great_white_shark	great white shark, white shark, man-eater, man...
3	3	tiger_shark	tiger shark, Galeocerdo cuvieri
4	4	hammerhead	hammerhead, hammerhead shark

(IMPLEMENTATION) Write a Dog Detector

While looking at the dictionary, you will notice that the categories corresponding to dogs appear in an uninterrupted sequence and correspond to dictionary keys 151-268, inclusive, to include all categories from `'Chihuahua'` to `'Mexican hairless'`. Thus, in order to check to see if an image is predicted to contain a dog by the pre-trained VGG-16 model, we need only check if the pre-trained model predicts an index between 151 and 268 (inclusive).

Use these ideas to complete the `dog_detector` function below, which returns `True` if a dog is detected in an image (and `False` if not).

In [ ]:
```
MIN_DOG_CLASS_INDEX = 151
MAX_DOG_CLASS_INDEX = 268

### returns "True" if a dog is detected in the image stored at img_path
def dog_detector(img_path):
    ## TODO: Complete the function.
    class_index = VGG16_predict(img_path)

    return (class_index >= MIN_DOG_CLASS_INDEX) and (class_index <= MAX_DOG_CLASS_IN
```

(IMPLEMENTATION) Assess the Dog Detector

Question 2: Use the code cell below to test the performance of your `dog_detector` function.

- What percentage of the images in `human_files_short` have a detected dog?
- What percentage of the images in `dog_files_short` have a detected dog?

Answer:

I've achieved a 1% percentage detecting dogs in the Humans dataset and a 94% percentage of detecting dogs in the Dogs dataset

In [ ]:
```
### TODO: Test the performance of the dog_detector function
### on the images in human_files_short and dog_files_short.
def detect_dogs(images_paths):
        dog_detector_results = [dog_detector(image_path) for image_path in tqdm(imag
        images_with_dogs = [1 for result in dog_detector_results if result == True]
        return len(images_with_dogs) / len(images_paths)

dogs_in_humans = detect_dogs(human_files_short)
dogs_in_dogs = detect_dogs(dog_files_short)
```

```
print(f"Images with a dog in the Humans dataset: {dogs_in_humans * 100} % - Images w
```

```
100%|██████████| 100/100 [00:06<00:00, 14.73it/s]
100%|██████████| 100/100 [00:01<00:00, 80.76it/s]
Images with a dog in the Humans dataset: 1.0 % - Images with a dog in the Dogs datas
et: 94.0 %
```

We suggest VGG-16 as a potential network to detect dog images in your algorithm, but you are free to explore other pre-trained networks (such as Inception-v3, ResNet-50, etc). Please use the code cell below to test other pre-trained PyTorch models. If you decide to pursue this *optional* task, report performance on `human_files_short` and `dog_files_short`.

In [ ]:

```
### (Optional)
### TODO: Report the performance of another pre-trained network.
### Feel free to use as many code cells as needed.

RESNET18 = models.resnet18(pretrained=True)

if use_cuda:
    RESNET18 = RESNET18.cuda()

def RESNET18_predict(img_path):
    image = Image.open(img_path)

    transformations = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])]

    image = transformations(image) # apply transformations to the image

    image = torch.unsqueeze(image, 0) # add a dimension because the model expect a b

    if use_cuda:
        image = image.cuda()

    RESNET18.eval() # change to evaluation mode, just to be sure (I think that VGG d

    predictions = RESNET18(image)

    predictions = torch.squeeze(predictions, 0) # remove the artificial dimension ad

    index = torch.argmax(predictions)

    if use_cuda:
        index = index.cpu()

    return index.item() # predicted class index

def dog_detector_with_RESNET18(img_path):
    ## TODO: Complete the function.
    class_index = RESNET18_predict(img_path)

    return (class_index >= MIN_DOG_CLASS_INDEX) and (class_index <= MAX_DOG_CLASS_IN
    ## TODO: Complete the function.
    class_index = RESNET18_predict(img_path)

    return (class_index >= MIN_DOG_CLASS_INDEX) and (class_index <= MAX_DOG_CLASS_IN

def detect_dogs_with_RESNET18(images_paths):
```

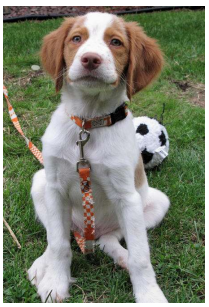
```
    dog_detector_results = [dog_detector_with_RESNET18(image_path) for image_path in
    images_with_dogs = [1 for result in dog_detector_results if result == True]
    return len(images_with_dogs) / len(images_paths)

dogs_in_humans_with_RESNET18 = detect_dogs_with_RESNET18(human_files_short)
dogs_in_dogs_with_RESNET18 = detect_dogs_with_RESNET18(dog_files_short)

print(f"Detection with RESNET18: Images with a dog in the Humans dataset: {dogs_in_h
```

```
100%|██████████| 100/100 [00:00<00:00, 190.22it/s]
100%|██████████| 100/100 [00:00<00:00, 100.57it/s]
Detection with RESNET18: Images with a dog in the Humans dataset: 3.0 % - Images wit
h a dog in the Dogs dataset: 93.0 %
```

Step 3: Create a CNN to Classify Dog Breeds (from Scratch)

Now that we have functions for detecting humans and dogs in images, we need a way to predict breed from images. In this step, you will create a CNN that classifies dog breeds. You must create your CNN *from scratch* (so, you can't use transfer learning *yet*!), and you must attain a test accuracy of at least 10%. In Step 4 of this notebook, you will have the opportunity to use transfer learning to create a CNN that attains greatly improved accuracy.

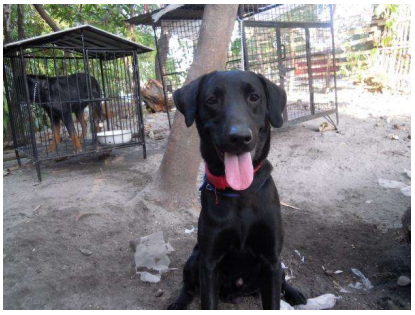
We mention that the task of assigning breed to dogs from images is considered exceptionally challenging. To see why, consider that *even a human* would have trouble distinguishing between a Brittany and a Welsh Springer Spaniel.

Brittany	Welsh Springer Spaniel
	

It is not difficult to find other dog breed pairs with minimal inter-class variation (for instance, Curly-Coated Retrievers and American Water Spaniels).

Curly-Coated Retriever	American Water Spaniel
	

Likewise, recall that labradors come in yellow, chocolate, and black. Your vision-based algorithm will have to conquer this high intra-class variation to determine how to classify all of these different shades as the same breed.

Yellow Labrador	Chocolate Labrador	Black Labrador
		

We also mention that random chance presents an exceptionally low bar: setting aside the fact that the classes are slightly imabalanced, a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%.

Remember that the practice is far ahead of the theory in deep learning. Experiment with many different architectures, and trust your intuition. And, of course, have fun!

(IMPLEMENTATION) Specify Data Loaders for the Dog Dataset

Use the code cell below to write three separate data loaders for the training, validation, and test datasets of dog images (located at `dog_images/train`, `dog_images/valid`, and `dog_images/test`, respectively). You may find this documentation on custom datasets to be a useful resource. If you are interested in augmenting your training and/or validation data, check out the wide variety of transforms!

Warning
I've found that this image cannot be loaded:
`data/dog_images/train\098.Leonberger\Leonberger_06571.jpg`
When the loader calls this method `img.convert('RGB')` I get this error: `image file is truncated (150 bytes not processed)`
So I deleted the image (there are similar images in the same folder).

Some statistics about the Dog Datasets:

- Total: 8,350 images (was 8,351 originally), 1,147,293,151 bytes (was 1,147,674,093 bytes originally)
- Train: 6,679 images (was 6,680 originally), 931,503,820 bytes (was 931,884,762 bytes originally)
- Validation: 835 images, 109,623,302 bytes
- Test: 836 images, 106,166,029 bytes
- Average size of the images: 134 KB

Each folder (train, valid, test) has 133 subfolders, corresponding to the class of the images, so there are 133 classes in each datasets.

In [ ]:
```
import os
from torchvision import datasets
```

```
from torch.utils.data import DataLoader

### TODO: Write data loaders for training, validation, and test sets
## Specify appropriate transforms, and batch_sizes

train_transformations = transforms.Compose([
        transforms.Resize(256),
        transforms.ColorJitter(brightness=0.1, contrast=0.1),
        transforms.GaussianBlur(kernel_size=3),
        transforms.RandomRotation(5),
        transforms.RandomCrop(250),
        transforms.RandomHorizontalFlip(0.5),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])]

test_valid_transformations = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])]

batch_size = 32

train_data = datasets.ImageFolder("data/dog_images/train", transform=train_transform
validation_data = datasets.ImageFolder("data/dog_images/valid", transform=test_valid
test_data = datasets.ImageFolder("data/dog_images/test", transform=test_valid_transf

train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
validation_loader = DataLoader(validation_data, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=True)
loaders_scratch = {"train": train_loader, "valid": validation_loader, "test": test_l
```

Just show some loaded and transformed images to see what happens

In [ ]:

```
from torchvision.utils import make_grid

_, axis = plt.subplots(figsize=(20,20))

batch = next(iter(train_loader)) # returns a list with the list of images as first i
images = batch[0][:16] # Get first 16 images to show in a 4x4 grid

axis.imshow(make_grid(images, 4, 4).permute(1, 2, 0))
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats o
r [0..255] for integers).
```

Out[ ]: <matplotlib.image.AxesImage at 0x21c9617eba8>

Question 3: Describe your chosen procedure for preprocessing the data.

- How does your code resize the images (by cropping, stretching, etc)? What size did you pick for the input tensor, and why?
- Did you decide to augment the dataset? If so, how (through translations, flips, rotations, etc)? If not, why not?

Answer:

I'm using a loader for training and a loader for both test and validation.
The final dimension of the two images will be 224 x 224 because Pytorch models are often trained with images with this size. The images are normalized as suggested in the Pytorch documentation.
The Train dataset does these steps:

- Resize images to 256 x 256 because the images will be slightly rotated and I want to avoid black zones in the images
- Dataset is augmented with these random transformations: ColorJitter, GaussianBlur, RandomRotation, RandomCrop, RandomHorizontalFlip
- Finally images are cropped to the final size, transformed to Pytorch Tensors and normalized

The Test and Validation datasets do these steps:

- Resize images to 256 x 256
- Crop images to the final size, transform to Pytorch Tensors and normalize them

(IMPLEMENTATION) Model Architecture

Create a CNN to classify dog breed. Use the template in the code cell below.

In [ ]:
```
# Print VGG16 structure to use as a base for the model from scratch
print(VGG16)
```

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

In [ ]:
```
# Print RESNET18 structure to use as a base for the model from scratch
print(RESNET18)
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=Fal
se)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
```

```
rue)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=Fal
se)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bia
s=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bia
s=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bia
s=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bia
s=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), b
ias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (downsample): Sequential(
```

```
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), b
ias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

In [ ]:

```
import torch.nn as nn
import torch.nn.functional as F

# define the CNN architecture
class Net(nn.Module):
    ### TODO: choose an architecture, and complete the class

    class ResNetBlock(nn.Module):
        def __init__(self, channels, size) -> None:
            super().__init__()

            self.sequential = nn.Sequential(*self._create_block(channels))
            self.pooling = nn.AdaptiveAvgPool2d(size)

        def forward(self, data):
            current = data.clone()
            current = self.sequential(current)
```

```
            data += current
            data = self.pooling(data)
            return data

        def _create_block(self, channels):
            batch_norm_01 = nn.BatchNorm2d(channels)
            relu_01 = nn.ReLU(inplace=True)
            conv_01 = nn.Conv2d(channels, channels, kernel_size=3, padding='same')
            batch_norm_02 = nn.BatchNorm2d(channels)
            relu_02 = nn.ReLU(inplace=True)
            conv_02 = nn.Conv2d(channels, channels, kernel_size=3, padding='same')
            return [batch_norm_01, relu_01, conv_01, batch_norm_02, relu_02, conv_02

    def __init__(self):
        super(Net, self).__init__()

        ## Define layers of a CNN

        self.layers = []

        self.conv01 = nn.Conv2d(3, 32, kernel_size=(3, 3), padding='same')

        self.resnet_block_01 = self.ResNetBlock(32, 224)
        self.resnet_block_02 = self.ResNetBlock(32, 112)
        self.resnet_block_03 = self.ResNetBlock(32, 56)
        self.resnet_block_04 = self.ResNetBlock(32, 28)

        self.flatten = nn.Flatten()
        self.dropout = nn.Dropout(p=0.5, inplace=True)
        self.linear = nn.Linear(in_features=28*28*32, out_features=133)

    def forward(self, data):
        ## Define forward behavior

        data = self.conv01(data)

        data = self.resnet_block_01(data)
        data = self.resnet_block_02(data)
        data = self.resnet_block_03(data)
        data = self.resnet_block_04(data)

        data = self.flatten(data)
        data = self.dropout(data)
        data = self.linear(data)
        return data

    def _create_residual_block(self, channels):
        batch_norm_01 = nn.BatchNorm2d(channels)
        relu_01 = nn.ReLU(inplace=True)
        conv_01 = nn.Conv2d(channels, channels, kernel_size=3, padding='same')
        batch_norm_02 = nn.BatchNorm2d(channels)
        relu_02 = nn.ReLU(inplace=True)
        conv_02 = nn.Conv2d(channels, channels, kernel_size=3, padding='same')
        return [batch_norm_01, relu_01, conv_01, batch_norm_02, relu_02, conv_02]

#-#-# You so NOT have to modify the code below this line. #-#-#

# instantiate the CNN
model_scratch = Net()

print(model_scratch)

# move tensors to GPU if CUDA is available
if use_cuda:
    model_scratch.cuda()
```

```
Net(
  (conv01): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=same)
  (resnet_block_01): ResNetBlock(
    (sequential): Sequential(
      (0): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (1): ReLU(inplace=True)
      (2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=same)
      (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (4): ReLU(inplace=True)
      (5): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=same)
    )
    (pooling): AdaptiveAvgPool2d(output_size=224)
  )
  (resnet_block_02): ResNetBlock(
    (sequential): Sequential(
      (0): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (1): ReLU(inplace=True)
      (2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=same)
      (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (4): ReLU(inplace=True)
      (5): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=same)
    )
    (pooling): AdaptiveAvgPool2d(output_size=112)
  )
  (resnet_block_03): ResNetBlock(
    (sequential): Sequential(
      (0): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (1): ReLU(inplace=True)
      (2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=same)
      (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (4): ReLU(inplace=True)
      (5): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=same)
    )
    (pooling): AdaptiveAvgPool2d(output_size=56)
  )
  (resnet_block_04): ResNetBlock(
    (sequential): Sequential(
      (0): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (1): ReLU(inplace=True)
      (2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=same)
      (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (4): ReLU(inplace=True)
      (5): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=same)
    )
    (pooling): AdaptiveAvgPool2d(output_size=28)
  )
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (dropout): Dropout(p=0.5, inplace=True)
  (linear): Linear(in_features=25088, out_features=133, bias=True)
)
```

Question 4: Outline the steps you took to get to your final CNN architecture and your reasoning at each step.

Answer:

VGG16 achieves a good accuracy, so I'll use it as inspiration.
Inputs are often resized as batches of 32/64 RGB images with dimensions of 224 x 224, but with batches of 64 images I get Out of Memory errors, so I'm using batches of 32 RGB images of 224

x 224 (batch size is 32 x 3 x 224 x 224).
VGG16 uses 13 convolutional layers and 3 linear layers, so I'll use a similar complexity.
I'm using Resnet blocks (blocks of two convolutional layers that add inputs to outputs) and Batch Normalization to reduce the vanishing gradient problems.
I've found an explanation of Residual Block in this paper: Residual blocks — Building blocks of ResNet. I'm using the structure marked as **(e)** in the figure Types of Residual Block found in the cited article.
The last layer of the model has 133 nodes because the datasets has only 133 classes (there are 133 dog breeds in the Imagenet dataset).
I prefer to use layers instead of functional for operation without parameters (like ReLU), so printing the model I can get the entire structure, including the activation functions.

(IMPLEMENTATION) Specify Loss Function and Optimizer

Use the next code cell to specify a loss function and optimizer. Save the chosen loss function as `criterion_scratch` , and the optimizer as `optimizer_scratch` below.

In [ ]:

```
import torch.optim as optim
from torch.optim.lr_scheduler import StepLR # Importing Learning Rate Scheduler

### TODO: select loss function
criterion_scratch = nn.CrossEntropyLoss()

### TODO: select optimizer
optimizer_scratch = optim.Adam(model_scratch.parameters(), lr = 0.01)

scheduler_scratch = StepLR(optimizer_scratch, step_size=20, gamma=0.5) # Configuring
```

(IMPLEMENTATION) Train and Validate the Model

Train and validate your model in the code cell below. Save the final model parameters at filepath `'model_scratch.pt'` .

In [ ]:

```
def train(n_epochs, loaders, model, optimizer, criterion, use_cuda, save_path, sched
    """returns trained model"""
    # initialize tracker for minimum validation loss
    valid_loss_min = np.Inf

    train_losses = []
    valid_losses = []
    train_accuracies = []
    valid_accuracies = []

    _, (loss_plot, accuracy_plot) = plt.subplots(1, 2, gridspec_kw={"wspace": 0.4})
    loss_plot.set_xlabel = "Epoch"
    loss_plot.set_ylabel = "Loss"
    accuracy_plot.set_xlabel = "Epoch"
    accuracy_plot.set_ylabel = "Accuracy"

    model_number = 0
    model_file = ""

    for epoch in range(1, n_epochs + 1):
        # initialize variables to monitor training and validation loss
        train_loss = 0.0
        valid_loss = 0.0
```

```
        train_correct = 0
        train_total = 0
        valid_correct = 0
        valid_total = 0

        ###################
        # train the model #
        ###################
        model.train()
        for batch_idx, (data, target) in enumerate(loaders['train']):
            # move to GPU
            if use_cuda:
                data, target = data.cuda(), target.cuda()
            ## find the loss and update the model parameters accordingly
            ## record the average training loss, using something like
            ## train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.data - train

            train_total += len(data)

            if ((batch_idx + 1) == 1) or \
                    ((batch_idx + 1) % 100 == 0) or \
                        ((batch_idx + 1) == len(loaders['train'])):
                print(f"TRAIN: Epoch = {epoch} - Batch Number = {batch_idx + 1} - Ba

            optimizer.zero_grad() # Reset gradients
            train_output = model(data) # Calculate the results
            actual_loss = criterion(train_output, target) # Calculate the error
            actual_loss.backward() # Calculate the gradients
            optimizer.step() # Update model parameters
            train_loss += ((1 / (batch_idx + 1)) * (actual_loss.data - train_loss))

            train_predictions = train_output.data.max(1, keepdim=True)[1]
            train_correct += np.sum(np.squeeze(train_predictions.eq(target.data.view

        train_accuracy = 100.0 * train_correct / train_total

        ######################
        # validate the model #
        ######################
        model.eval()
        for batch_idx, (data, target) in enumerate(loaders['valid']):
            # move to GPU
            if use_cuda:
                data, target = data.cuda(), target.cuda()
            ## update the average validation loss

            valid_total += len(data)

            if ((batch_idx + 1) == 1) or \
                    ((batch_idx + 1) % 25 == 0) or \
                        ((batch_idx + 1) == len(loaders['valid'])):
                print(f"VALIDATION: Epoch = {epoch} - Batch Number = {batch_idx + 1}

            valid_output = model(data) # Calculate the results
            actual_loss = criterion(valid_output, target) # Calculate the error
            valid_loss += ((1 / (batch_idx + 1)) * (actual_loss.data - valid_loss))

            valid_predictions = valid_output.data.max(1, keepdim=True)[1]
            valid_correct += np.sum(np.squeeze(valid_predictions.eq(target.data.view

        valid_accuracy = 100.0 * valid_correct / valid_total

        if scheduler is None:
            # print training/validation statistics
```

```
            message = "Epoch: {}" + \
                "\r\n\tTraining Loss: {:.6f} " + \
                    "\tValidation Loss: {:.6f} "+ \
                        "\r\n\tTraining Accuracy: {:.2f}% ({}/{})" + \
                            "\tValidation Accuracy: {:.2f}% ({}/{})"
            print(message.format(
                epoch,
                train_loss,
                valid_loss,
                train_accuracy,
                train_correct,
                train_total,
                valid_accuracy,
                valid_correct,
                valid_total
            ))
        else:
            current_learning_rate = scheduler.get_last_lr()[-1] # Saving current Lea

            scheduler.step() # Updating Learning Rate

            # print training/validation statistics
            message = "Epoch: {} \tLearning Rate: {:.6f}" + \
                "\r\n\tTraining Loss: {:.6f} " + \
                    "\tValidation Loss: {:.6f} "+ \
                        "\r\n\tTraining Accuracy: {:.2f}% ({}/{})" + \
                            "\tValidation Accuracy: {:.2f}% ({}/{})"
            print(message.format(
                epoch,
                current_learning_rate,
                train_loss,
                valid_loss,
                train_accuracy,
                train_correct,
                train_total,
                valid_accuracy,
                valid_correct,
                valid_total
            ))

        train_losses.append(train_loss)
        valid_losses.append(valid_loss)
        train_accuracies.append(train_accuracy)
        valid_accuracies.append(valid_accuracy)

        ## TODO: save the model if validation loss has decreased
        if valid_loss < valid_loss_min:
            print(f"Saving model: Training Loss: {train_loss:.6f} - Validation Loss:
                f"Training Accuracy: {train_accuracy:.2f}% ({train_correct}/{train_t
                f"Validation Accuracy: {valid_accuracy:.2f}% ({valid_correct}/{valid
            valid_loss_min = valid_loss

            # It seems that when saving large file, the file it's not close properly
            # I've tried several workarounds (open and closing explicitly, deleting
            # but nothing worked, so I'll save every model in a new file and finally
            # with the correct name
            # torch.save(model.state_dict(), save_path)
            if not use_model_versioning:
                torch.save(model.state_dict(), save_path)
            else:
                model_number += 1
                model_file = f"{save_path.replace('.pt', '')}_{model_number}.pt"
                torch.save(model.state_dict(), model_file)

loss_plot.plot(range(1, n_epochs + 1), train_losses, label="Train")
```

```
    loss_plot.plot(range(1, n_epochs + 1), valid_losses, label="Valid")
    loss_plot.legend()
    loss_plot.text(1, train_losses[0], "Train")
    loss_plot.text(1, valid_losses[0], "Valid")
    accuracy_plot.plot(range(1, n_epochs + 1), train_accuracies, label="Train")
    accuracy_plot.plot(range(1, n_epochs + 1), valid_accuracies, label="Valid")
    accuracy_plot.legend()
    accuracy_plot.text(1, train_accuracies[0], "Train")
    accuracy_plot.text(1, valid_accuracies[0], "Valid")

    if use_model_versioning:
        if os.path.exists(save_path):
            os.remove(save_path)
        os.rename(model_file, save_path)

    # return trained model
    return model

# train the model
model_scratch = train(100, loaders_scratch, model_scratch, optimizer_scratch,
                      criterion_scratch, use_cuda, 'model_scratch.pt', scheduler_scr

# load the model that got the best validation accuracy
model_scratch.load_state_dict(torch.load('model_scratch.pt'))
```

```
TRAIN: Epoch = 1 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 1 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 1 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 1 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 1 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 1 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 1 - Batch Number = 27 - Batch Length = 3
Epoch: 1        Learning Rate: 0.010000
        Training Loss: 111.481018       Validation Loss: 20.092075
        Training Accuracy: 1.26% (84/6679)      Validation Accuracy: 1.44% (12/835)
Saving model: Training Loss: 111.481018 - Validation Loss: 20.092075 - Training Accu
racy: 1.26% (84/6679) - Validation Accuracy: 1.44% (12/835)
TRAIN: Epoch = 2 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 2 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 2 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 2 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 2 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 2 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 2 - Batch Number = 27 - Batch Length = 3
Epoch: 2        Learning Rate: 0.010000
        Training Loss: 15.344349        Validation Loss: 10.319086
        Training Accuracy: 1.89% (126/6679)     Validation Accuracy: 1.80% (15/835)
Saving model: Training Loss: 15.344349 - Validation Loss: 10.319086 - Training Accur
acy: 1.89% (126/6679) - Validation Accuracy: 1.80% (15/835)
TRAIN: Epoch = 3 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 3 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 3 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 3 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 3 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 3 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 3 - Batch Number = 27 - Batch Length = 3
Epoch: 3        Learning Rate: 0.010000
        Training Loss: 8.104103         Validation Loss: 6.656757
        Training Accuracy: 3.07% (205/6679)     Validation Accuracy: 1.92% (16/835)
Saving model: Training Loss: 8.104103 - Validation Loss: 6.656757 - Training Accurac
y: 3.07% (205/6679) - Validation Accuracy: 1.92% (16/835)
TRAIN: Epoch = 4 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 4 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 4 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 4 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 4 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 4 - Batch Number = 25 - Batch Length = 32
```

```
VALIDATION: Epoch = 4 - Batch Number = 27 - Batch Length = 3
Epoch: 4        Learning Rate: 0.010000
        Training Loss: 6.596448         Validation Loss: 5.868458
        Training Accuracy: 3.59% (240/6679)     Validation Accuracy: 2.40% (20/835)
Saving model: Training Loss: 6.596448 - Validation Loss: 5.868458 - Training Accurac
y: 3.59% (240/6679) - Validation Accuracy: 2.40% (20/835)
TRAIN: Epoch = 5 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 5 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 5 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 5 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 5 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 5 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 5 - Batch Number = 27 - Batch Length = 3
Epoch: 5        Learning Rate: 0.010000
        Training Loss: 6.542132         Validation Loss: 5.943712
        Training Accuracy: 3.73% (249/6679)     Validation Accuracy: 2.87% (24/835)
TRAIN: Epoch = 6 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 6 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 6 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 6 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 6 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 6 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 6 - Batch Number = 27 - Batch Length = 3
Epoch: 6        Learning Rate: 0.010000
        Training Loss: 6.501535         Validation Loss: 5.979832
        Training Accuracy: 3.98% (266/6679)     Validation Accuracy: 3.23% (27/835)
TRAIN: Epoch = 7 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 7 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 7 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 7 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 7 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 7 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 7 - Batch Number = 27 - Batch Length = 3
Epoch: 7        Learning Rate: 0.010000
        Training Loss: 6.337212         Validation Loss: 6.217942
        Training Accuracy: 4.18% (279/6679)     Validation Accuracy: 4.07% (34/835)
TRAIN: Epoch = 8 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 8 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 8 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 8 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 8 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 8 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 8 - Batch Number = 27 - Batch Length = 3
Epoch: 8        Learning Rate: 0.010000
        Training Loss: 6.617059         Validation Loss: 6.038437
        Training Accuracy: 4.49% (300/6679)     Validation Accuracy: 4.07% (34/835)
TRAIN: Epoch = 9 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 9 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 9 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 9 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 9 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 9 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 9 - Batch Number = 27 - Batch Length = 3
Epoch: 9        Learning Rate: 0.010000
        Training Loss: 7.237194         Validation Loss: 8.308772
        Training Accuracy: 4.22% (282/6679)     Validation Accuracy: 3.11% (26/835)
TRAIN: Epoch = 10 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 10 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 10 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 10 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 10 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 10 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 10 - Batch Number = 27 - Batch Length = 3
Epoch: 10       Learning Rate: 0.010000
        Training Loss: 8.330458         Validation Loss: 10.861801
        Training Accuracy: 4.03% (269/6679)     Validation Accuracy: 2.63% (22/835)
TRAIN: Epoch = 11 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 11 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 11 - Batch Number = 200 - Batch Length = 32
```

```
TRAIN: Epoch = 11 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 11 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 11 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 11 - Batch Number = 27 - Batch Length = 3
Epoch: 11       Learning Rate: 0.010000
        Training Loss: 10.840325        Validation Loss: 11.750802
        Training Accuracy: 3.86% (258/6679)     Validation Accuracy: 1.20% (10/835)
TRAIN: Epoch = 12 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 12 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 12 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 12 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 12 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 12 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 12 - Batch Number = 27 - Batch Length = 3
Epoch: 12       Learning Rate: 0.010000
        Training Loss: 14.930326        Validation Loss: 22.687115
        Training Accuracy: 3.41% (228/6679)     Validation Accuracy: 1.68% (14/835)
TRAIN: Epoch = 13 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 13 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 13 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 13 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 13 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 13 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 13 - Batch Number = 27 - Batch Length = 3
Epoch: 13       Learning Rate: 0.010000
        Training Loss: 18.467701        Validation Loss: 16.301119
        Training Accuracy: 2.71% (181/6679)     Validation Accuracy: 2.51% (21/835)
TRAIN: Epoch = 14 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 14 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 14 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 14 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 14 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 14 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 14 - Batch Number = 27 - Batch Length = 3
Epoch: 14       Learning Rate: 0.010000
        Training Loss: 18.450289        Validation Loss: 18.375101
        Training Accuracy: 2.80% (187/6679)     Validation Accuracy: 2.75% (23/835)
TRAIN: Epoch = 15 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 15 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 15 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 15 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 15 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 15 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 15 - Batch Number = 27 - Batch Length = 3
Epoch: 15       Learning Rate: 0.010000
        Training Loss: 16.644255        Validation Loss: 14.934290
        Training Accuracy: 3.31% (221/6679)     Validation Accuracy: 2.16% (18/835)
TRAIN: Epoch = 16 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 16 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 16 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 16 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 16 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 16 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 16 - Batch Number = 27 - Batch Length = 3
Epoch: 16       Learning Rate: 0.010000
        Training Loss: 16.746264        Validation Loss: 17.701054
        Training Accuracy: 3.20% (214/6679)     Validation Accuracy: 2.87% (24/835)
TRAIN: Epoch = 17 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 17 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 17 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 17 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 17 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 17 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 17 - Batch Number = 27 - Batch Length = 3
Epoch: 17       Learning Rate: 0.010000
        Training Loss: 13.779526        Validation Loss: 13.802267
        Training Accuracy: 3.55% (237/6679)     Validation Accuracy: 2.28% (19/835)
TRAIN: Epoch = 18 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 18 - Batch Number = 100 - Batch Length = 32
```

```
TRAIN: Epoch = 18 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 18 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 18 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 18 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 18 - Batch Number = 27 - Batch Length = 3
Epoch: 18       Learning Rate: 0.010000
        Training Loss: 15.671740        Validation Loss: 22.580532
        Training Accuracy: 3.41% (228/6679)     Validation Accuracy: 1.44% (12/835)
TRAIN: Epoch = 19 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 19 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 19 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 19 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 19 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 19 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 19 - Batch Number = 27 - Batch Length = 3
Epoch: 19       Learning Rate: 0.010000
        Training Loss: 16.166063        Validation Loss: 16.869678
        Training Accuracy: 3.74% (250/6679)     Validation Accuracy: 2.99% (25/835)
TRAIN: Epoch = 20 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 20 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 20 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 20 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 20 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 20 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 20 - Batch Number = 27 - Batch Length = 3
Epoch: 20       Learning Rate: 0.010000
        Training Loss: 15.871133        Validation Loss: 16.624937
        Training Accuracy: 3.79% (253/6679)     Validation Accuracy: 2.51% (21/835)
TRAIN: Epoch = 21 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 21 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 21 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 21 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 21 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 21 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 21 - Batch Number = 27 - Batch Length = 3
Epoch: 21       Learning Rate: 0.005000
        Training Loss: 8.435807         Validation Loss: 8.351117
        Training Accuracy: 5.75% (384/6679)     Validation Accuracy: 3.11% (26/835)
TRAIN: Epoch = 22 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 22 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 22 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 22 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 22 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 22 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 22 - Batch Number = 27 - Batch Length = 3
Epoch: 22       Learning Rate: 0.005000
        Training Loss: 7.025702         Validation Loss: 6.942466
        Training Accuracy: 6.11% (408/6679)     Validation Accuracy: 4.43% (37/835)
TRAIN: Epoch = 23 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 23 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 23 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 23 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 23 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 23 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 23 - Batch Number = 27 - Batch Length = 3
Epoch: 23       Learning Rate: 0.005000
        Training Loss: 6.606545         Validation Loss: 6.520589
        Training Accuracy: 6.89% (460/6679)     Validation Accuracy: 4.55% (38/835)
TRAIN: Epoch = 24 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 24 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 24 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 24 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 24 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 24 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 24 - Batch Number = 27 - Batch Length = 3
Epoch: 24       Learning Rate: 0.005000
        Training Loss: 6.654833         Validation Loss: 6.133405
        Training Accuracy: 6.59% (440/6679)     Validation Accuracy: 5.03% (42/835)
TRAIN: Epoch = 25 - Batch Number = 1 - Batch Length = 32
```

```
TRAIN: Epoch = 25 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 25 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 25 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 25 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 25 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 25 - Batch Number = 27 - Batch Length = 3
Epoch: 25       Learning Rate: 0.005000
        Training Loss: 6.310643         Validation Loss: 5.946960
        Training Accuracy: 7.44% (497/6679)     Validation Accuracy: 5.03% (42/835)
TRAIN: Epoch = 26 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 26 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 26 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 26 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 26 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 26 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 26 - Batch Number = 27 - Batch Length = 3
Epoch: 26       Learning Rate: 0.005000
        Training Loss: 6.452153         Validation Loss: 7.022847
        Training Accuracy: 7.55% (504/6679)     Validation Accuracy: 3.83% (32/835)
TRAIN: Epoch = 27 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 27 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 27 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 27 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 27 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 27 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 27 - Batch Number = 27 - Batch Length = 3
Epoch: 27       Learning Rate: 0.005000
        Training Loss: 6.672282         Validation Loss: 6.862758
        Training Accuracy: 6.62% (442/6679)     Validation Accuracy: 3.83% (32/835)
TRAIN: Epoch = 28 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 28 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 28 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 28 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 28 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 28 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 28 - Batch Number = 27 - Batch Length = 3
Epoch: 28       Learning Rate: 0.005000
        Training Loss: 6.470906         Validation Loss: 7.004987
        Training Accuracy: 7.28% (486/6679)     Validation Accuracy: 4.67% (39/835)
TRAIN: Epoch = 29 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 29 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 29 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 29 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 29 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 29 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 29 - Batch Number = 27 - Batch Length = 3
Epoch: 29       Learning Rate: 0.005000
        Training Loss: 6.418140         Validation Loss: 5.879237
        Training Accuracy: 7.74% (517/6679)     Validation Accuracy: 5.51% (46/835)
TRAIN: Epoch = 30 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 30 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 30 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 30 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 30 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 30 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 30 - Batch Number = 27 - Batch Length = 3
Epoch: 30       Learning Rate: 0.005000
        Training Loss: 6.407489         Validation Loss: 6.086012
        Training Accuracy: 8.28% (553/6679)     Validation Accuracy: 4.79% (40/835)
TRAIN: Epoch = 31 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 31 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 31 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 31 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 31 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 31 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 31 - Batch Number = 27 - Batch Length = 3
Epoch: 31       Learning Rate: 0.005000
        Training Loss: 7.116241         Validation Loss: 8.726353
        Training Accuracy: 7.20% (481/6679)     Validation Accuracy: 3.23% (27/835)
```

```
TRAIN: Epoch = 32 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 32 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 32 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 32 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 32 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 32 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 32 - Batch Number = 27 - Batch Length = 3
Epoch: 32       Learning Rate: 0.005000
        Training Loss: 6.609919         Validation Loss: 7.039008
        Training Accuracy: 8.19% (547/6679)     Validation Accuracy: 5.63% (47/835)
TRAIN: Epoch = 33 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 33 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 33 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 33 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 33 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 33 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 33 - Batch Number = 27 - Batch Length = 3
Epoch: 33       Learning Rate: 0.005000
        Training Loss: 6.908591         Validation Loss: 7.189306
        Training Accuracy: 8.31% (555/6679)     Validation Accuracy: 3.83% (32/835)
TRAIN: Epoch = 34 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 34 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 34 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 34 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 34 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 34 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 34 - Batch Number = 27 - Batch Length = 3
Epoch: 34       Learning Rate: 0.005000
        Training Loss: 7.233864         Validation Loss: 7.335357
        Training Accuracy: 8.56% (572/6679)     Validation Accuracy: 3.83% (32/835)
TRAIN: Epoch = 35 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 35 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 35 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 35 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 35 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 35 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 35 - Batch Number = 27 - Batch Length = 3
Epoch: 35       Learning Rate: 0.005000
        Training Loss: 7.441982         Validation Loss: 7.782876
        Training Accuracy: 7.89% (527/6679)     Validation Accuracy: 4.07% (34/835)
TRAIN: Epoch = 36 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 36 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 36 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 36 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 36 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 36 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 36 - Batch Number = 27 - Batch Length = 3
Epoch: 36       Learning Rate: 0.005000
        Training Loss: 7.364164         Validation Loss: 7.645422
        Training Accuracy: 7.70% (514/6679)     Validation Accuracy: 4.43% (37/835)
TRAIN: Epoch = 37 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 37 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 37 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 37 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 37 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 37 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 37 - Batch Number = 27 - Batch Length = 3
Epoch: 37       Learning Rate: 0.005000
        Training Loss: 7.057166         Validation Loss: 7.494228
        Training Accuracy: 8.67% (579/6679)     Validation Accuracy: 7.43% (62/835)
TRAIN: Epoch = 38 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 38 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 38 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 38 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 38 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 38 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 38 - Batch Number = 27 - Batch Length = 3
Epoch: 38       Learning Rate: 0.005000
        Training Loss: 6.892736         Validation Loss: 8.165794
```

```
        Training Accuracy: 9.01% (602/6679)     Validation Accuracy: 5.87% (49/835)
TRAIN: Epoch = 39 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 39 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 39 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 39 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 39 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 39 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 39 - Batch Number = 27 - Batch Length = 3
Epoch: 39       Learning Rate: 0.005000
        Training Loss: 7.685175         Validation Loss: 7.849782
        Training Accuracy: 8.23% (550/6679)     Validation Accuracy: 3.95% (33/835)
TRAIN: Epoch = 40 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 40 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 40 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 40 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 40 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 40 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 40 - Batch Number = 27 - Batch Length = 3
Epoch: 40       Learning Rate: 0.005000
        Training Loss: 7.368264         Validation Loss: 8.261791
        Training Accuracy: 8.88% (593/6679)     Validation Accuracy: 5.03% (42/835)
TRAIN: Epoch = 41 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 41 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 41 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 41 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 41 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 41 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 41 - Batch Number = 27 - Batch Length = 3
Epoch: 41       Learning Rate: 0.002500
        Training Loss: 5.140141         Validation Loss: 5.872838
        Training Accuracy: 13.42% (896/6679)    Validation Accuracy: 6.83% (57/835)
TRAIN: Epoch = 42 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 42 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 42 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 42 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 42 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 42 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 42 - Batch Number = 27 - Batch Length = 3
Epoch: 42       Learning Rate: 0.002500
        Training Loss: 4.801005         Validation Loss: 5.594129
        Training Accuracy: 14.43% (964/6679)    Validation Accuracy: 8.02% (67/835)
Saving model: Training Loss: 4.801005 - Validation Loss: 5.594129 - Training Accurac
y: 14.43% (964/6679) - Validation Accuracy: 8.02% (67/835)
TRAIN: Epoch = 43 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 43 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 43 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 43 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 43 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 43 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 43 - Batch Number = 27 - Batch Length = 3
Epoch: 43       Learning Rate: 0.002500
        Training Loss: 4.641853         Validation Loss: 5.211202
        Training Accuracy: 15.71% (1049/6679)   Validation Accuracy: 6.95% (58/835)
Saving model: Training Loss: 4.641853 - Validation Loss: 5.211202 - Training Accurac
y: 15.71% (1049/6679) - Validation Accuracy: 6.95% (58/835)
TRAIN: Epoch = 44 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 44 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 44 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 44 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 44 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 44 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 44 - Batch Number = 27 - Batch Length = 3
Epoch: 44       Learning Rate: 0.002500
        Training Loss: 4.570570         Validation Loss: 5.156029
        Training Accuracy: 15.41% (1029/6679)   Validation Accuracy: 8.62% (72/835)
Saving model: Training Loss: 4.570570 - Validation Loss: 5.156029 - Training Accurac
y: 15.41% (1029/6679) - Validation Accuracy: 8.62% (72/835)
TRAIN: Epoch = 45 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 45 - Batch Number = 100 - Batch Length = 32
```

```
TRAIN: Epoch = 45 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 45 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 45 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 45 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 45 - Batch Number = 27 - Batch Length = 3
Epoch: 45       Learning Rate: 0.002500
        Training Loss: 4.544639         Validation Loss: 5.174686
        Training Accuracy: 16.23% (1084/6679)   Validation Accuracy: 8.50% (71/835)
TRAIN: Epoch = 46 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 46 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 46 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 46 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 46 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 46 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 46 - Batch Number = 27 - Batch Length = 3
Epoch: 46       Learning Rate: 0.002500
        Training Loss: 4.607342         Validation Loss: 5.416032
        Training Accuracy: 15.66% (1046/6679)   Validation Accuracy: 8.86% (74/835)
TRAIN: Epoch = 47 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 47 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 47 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 47 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 47 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 47 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 47 - Batch Number = 27 - Batch Length = 3
Epoch: 47       Learning Rate: 0.002500
        Training Loss: 4.631073         Validation Loss: 5.904429
        Training Accuracy: 15.80% (1055/6679)   Validation Accuracy: 7.66% (64/835)
TRAIN: Epoch = 48 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 48 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 48 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 48 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 48 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 48 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 48 - Batch Number = 27 - Batch Length = 3
Epoch: 48       Learning Rate: 0.002500
        Training Loss: 4.700077         Validation Loss: 5.563506
        Training Accuracy: 15.56% (1039/6679)   Validation Accuracy: 6.71% (56/835)
TRAIN: Epoch = 49 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 49 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 49 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 49 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 49 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 49 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 49 - Batch Number = 27 - Batch Length = 3
Epoch: 49       Learning Rate: 0.002500
        Training Loss: 4.750462         Validation Loss: 5.195859
        Training Accuracy: 15.65% (1045/6679)   Validation Accuracy: 8.74% (73/835)
TRAIN: Epoch = 50 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 50 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 50 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 50 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 50 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 50 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 50 - Batch Number = 27 - Batch Length = 3
Epoch: 50       Learning Rate: 0.002500
        Training Loss: 4.482036         Validation Loss: 5.543723
        Training Accuracy: 16.92% (1130/6679)   Validation Accuracy: 5.75% (48/835)
TRAIN: Epoch = 51 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 51 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 51 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 51 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 51 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 51 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 51 - Batch Number = 27 - Batch Length = 3
Epoch: 51       Learning Rate: 0.002500
        Training Loss: 4.634370         Validation Loss: 5.481880
        Training Accuracy: 15.36% (1026/6679)   Validation Accuracy: 8.50% (71/835)
TRAIN: Epoch = 52 - Batch Number = 1 - Batch Length = 32
```

```
TRAIN: Epoch = 52 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 52 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 52 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 52 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 52 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 52 - Batch Number = 27 - Batch Length = 3
Epoch: 52       Learning Rate: 0.002500
        Training Loss: 4.586291         Validation Loss: 5.582438
        Training Accuracy: 16.75% (1119/6679)   Validation Accuracy: 8.26% (69/835)
TRAIN: Epoch = 53 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 53 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 53 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 53 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 53 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 53 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 53 - Batch Number = 27 - Batch Length = 3
Epoch: 53       Learning Rate: 0.002500
        Training Loss: 4.545233         Validation Loss: 5.349155
        Training Accuracy: 16.68% (1114/6679)   Validation Accuracy: 7.66% (64/835)
TRAIN: Epoch = 54 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 54 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 54 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 54 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 54 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 54 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 54 - Batch Number = 27 - Batch Length = 3
Epoch: 54       Learning Rate: 0.002500
        Training Loss: 4.610042         Validation Loss: 5.528902
        Training Accuracy: 16.35% (1092/6679)   Validation Accuracy: 7.31% (61/835)
TRAIN: Epoch = 55 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 55 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 55 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 55 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 55 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 55 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 55 - Batch Number = 27 - Batch Length = 3
Epoch: 55       Learning Rate: 0.002500
        Training Loss: 4.646403         Validation Loss: 5.221113
        Training Accuracy: 16.48% (1101/6679)   Validation Accuracy: 9.34% (78/835)
TRAIN: Epoch = 56 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 56 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 56 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 56 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 56 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 56 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 56 - Batch Number = 27 - Batch Length = 3
Epoch: 56       Learning Rate: 0.002500
        Training Loss: 4.446397         Validation Loss: 5.399811
        Training Accuracy: 18.31% (1223/6679)   Validation Accuracy: 7.07% (59/835)
TRAIN: Epoch = 57 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 57 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 57 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 57 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 57 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 57 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 57 - Batch Number = 27 - Batch Length = 3
Epoch: 57       Learning Rate: 0.002500
        Training Loss: 4.579528         Validation Loss: 5.214633
        Training Accuracy: 18.04% (1205/6679)   Validation Accuracy: 9.46% (79/835)
TRAIN: Epoch = 58 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 58 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 58 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 58 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 58 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 58 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 58 - Batch Number = 27 - Batch Length = 3
Epoch: 58       Learning Rate: 0.002500
        Training Loss: 4.726053         Validation Loss: 5.271165
        Training Accuracy: 18.00% (1202/6679)   Validation Accuracy: 6.83% (57/835)
```

```
TRAIN: Epoch = 59 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 59 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 59 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 59 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 59 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 59 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 59 - Batch Number = 27 - Batch Length = 3
Epoch: 59       Learning Rate: 0.002500
        Training Loss: 4.482984         Validation Loss: 5.542975
        Training Accuracy: 18.57% (1240/6679)   Validation Accuracy: 10.54% (88/835)
TRAIN: Epoch = 60 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 60 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 60 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 60 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 60 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 60 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 60 - Batch Number = 27 - Batch Length = 3
Epoch: 60       Learning Rate: 0.002500
        Training Loss: 4.616062         Validation Loss: 5.502017
        Training Accuracy: 18.72% (1250/6679)   Validation Accuracy: 9.70% (81/835)
TRAIN: Epoch = 61 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 61 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 61 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 61 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 61 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 61 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 61 - Batch Number = 27 - Batch Length = 3
Epoch: 61       Learning Rate: 0.001250
        Training Loss: 3.669930         Validation Loss: 5.279048
        Training Accuracy: 24.34% (1626/6679)   Validation Accuracy: 8.74% (73/835)
TRAIN: Epoch = 62 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 62 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 62 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 62 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 62 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 62 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 62 - Batch Number = 27 - Batch Length = 3
Epoch: 62       Learning Rate: 0.001250
        Training Loss: 3.579484         Validation Loss: 4.838084
        Training Accuracy: 24.79% (1656/6679)   Validation Accuracy: 10.90% (91/835)
Saving model: Training Loss: 3.579484 - Validation Loss: 4.838084 - Training Accurac
y: 24.79% (1656/6679) - Validation Accuracy: 10.90% (91/835)
TRAIN: Epoch = 63 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 63 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 63 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 63 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 63 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 63 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 63 - Batch Number = 27 - Batch Length = 3
Epoch: 63       Learning Rate: 0.001250
        Training Loss: 3.565651         Validation Loss: 4.934968
        Training Accuracy: 24.97% (1668/6679)   Validation Accuracy: 10.90% (91/835)
TRAIN: Epoch = 64 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 64 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 64 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 64 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 64 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 64 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 64 - Batch Number = 27 - Batch Length = 3
Epoch: 64       Learning Rate: 0.001250
        Training Loss: 3.567521         Validation Loss: 4.935832
        Training Accuracy: 24.93% (1665/6679)   Validation Accuracy: 11.02% (92/835)
TRAIN: Epoch = 65 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 65 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 65 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 65 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 65 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 65 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 65 - Batch Number = 27 - Batch Length = 3
```

```
Epoch: 65       Learning Rate: 0.001250
        Training Loss: 3.485925         Validation Loss: 5.118797
        Training Accuracy: 25.77% (1721/6679)   Validation Accuracy: 8.98% (75/835)
TRAIN: Epoch = 66 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 66 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 66 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 66 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 66 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 66 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 66 - Batch Number = 27 - Batch Length = 3
Epoch: 66       Learning Rate: 0.001250
        Training Loss: 3.528677         Validation Loss: 4.932831
        Training Accuracy: 25.80% (1723/6679)   Validation Accuracy: 10.66% (89/835)
TRAIN: Epoch = 67 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 67 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 67 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 67 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 67 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 67 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 67 - Batch Number = 27 - Batch Length = 3
Epoch: 67       Learning Rate: 0.001250
        Training Loss: 3.459615         Validation Loss: 4.919689
        Training Accuracy: 26.76% (1787/6679)   Validation Accuracy: 10.18% (85/835)
TRAIN: Epoch = 68 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 68 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 68 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 68 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 68 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 68 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 68 - Batch Number = 27 - Batch Length = 3
Epoch: 68       Learning Rate: 0.001250
        Training Loss: 3.479914         Validation Loss: 4.871528
        Training Accuracy: 26.53% (1772/6679)   Validation Accuracy: 9.82% (82/835)
TRAIN: Epoch = 69 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 69 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 69 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 69 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 69 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 69 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 69 - Batch Number = 27 - Batch Length = 3
Epoch: 69       Learning Rate: 0.001250
        Training Loss: 3.488640         Validation Loss: 5.045734
        Training Accuracy: 26.20% (1750/6679)   Validation Accuracy: 10.54% (88/835)
TRAIN: Epoch = 70 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 70 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 70 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 70 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 70 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 70 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 70 - Batch Number = 27 - Batch Length = 3
Epoch: 70       Learning Rate: 0.001250
        Training Loss: 3.480793         Validation Loss: 4.799309
        Training Accuracy: 26.44% (1766/6679)   Validation Accuracy: 10.78% (90/835)
Saving model: Training Loss: 3.480793 - Validation Loss: 4.799309 - Training Accurac
y: 26.44% (1766/6679) - Validation Accuracy: 10.78% (90/835)
TRAIN: Epoch = 71 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 71 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 71 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 71 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 71 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 71 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 71 - Batch Number = 27 - Batch Length = 3
Epoch: 71       Learning Rate: 0.001250
        Training Loss: 3.449620         Validation Loss: 5.309474
        Training Accuracy: 27.05% (1807/6679)   Validation Accuracy: 11.50% (96/835)
TRAIN: Epoch = 72 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 72 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 72 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 72 - Batch Number = 209 - Batch Length = 23
```

```
VALIDATION: Epoch = 72 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 72 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 72 - Batch Number = 27 - Batch Length = 3
Epoch: 72       Learning Rate: 0.001250
        Training Loss: 3.451427         Validation Loss: 4.823253
        Training Accuracy: 27.64% (1846/6679)   Validation Accuracy: 10.06% (84/835)
TRAIN: Epoch = 73 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 73 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 73 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 73 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 73 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 73 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 73 - Batch Number = 27 - Batch Length = 3
Epoch: 73       Learning Rate: 0.001250
        Training Loss: 3.430787         Validation Loss: 4.852455
        Training Accuracy: 27.70% (1850/6679)   Validation Accuracy: 9.94% (83/835)
TRAIN: Epoch = 74 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 74 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 74 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 74 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 74 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 74 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 74 - Batch Number = 27 - Batch Length = 3
Epoch: 74       Learning Rate: 0.001250
        Training Loss: 3.379121         Validation Loss: 4.783593
        Training Accuracy: 28.37% (1895/6679)   Validation Accuracy: 9.94% (83/835)
Saving model: Training Loss: 3.379121 - Validation Loss: 4.783593 - Training Accurac
y: 28.37% (1895/6679) - Validation Accuracy: 9.94% (83/835)
TRAIN: Epoch = 75 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 75 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 75 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 75 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 75 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 75 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 75 - Batch Number = 27 - Batch Length = 3
Epoch: 75       Learning Rate: 0.001250
        Training Loss: 3.432745         Validation Loss: 5.009666
        Training Accuracy: 27.53% (1839/6679)   Validation Accuracy: 10.66% (89/835)
TRAIN: Epoch = 76 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 76 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 76 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 76 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 76 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 76 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 76 - Batch Number = 27 - Batch Length = 3
Epoch: 76       Learning Rate: 0.001250
        Training Loss: 3.370217         Validation Loss: 5.001001
        Training Accuracy: 28.91% (1931/6679)   Validation Accuracy: 11.50% (96/835)
TRAIN: Epoch = 77 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 77 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 77 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 77 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 77 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 77 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 77 - Batch Number = 27 - Batch Length = 3
Epoch: 77       Learning Rate: 0.001250
        Training Loss: 3.412632         Validation Loss: 5.209480
        Training Accuracy: 28.72% (1918/6679)   Validation Accuracy: 9.58% (80/835)
TRAIN: Epoch = 78 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 78 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 78 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 78 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 78 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 78 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 78 - Batch Number = 27 - Batch Length = 3
Epoch: 78       Learning Rate: 0.001250
        Training Loss: 3.313253         Validation Loss: 5.153429
        Training Accuracy: 28.99% (1936/6679)   Validation Accuracy: 10.78% (90/835)
TRAIN: Epoch = 79 - Batch Number = 1 - Batch Length = 32
```

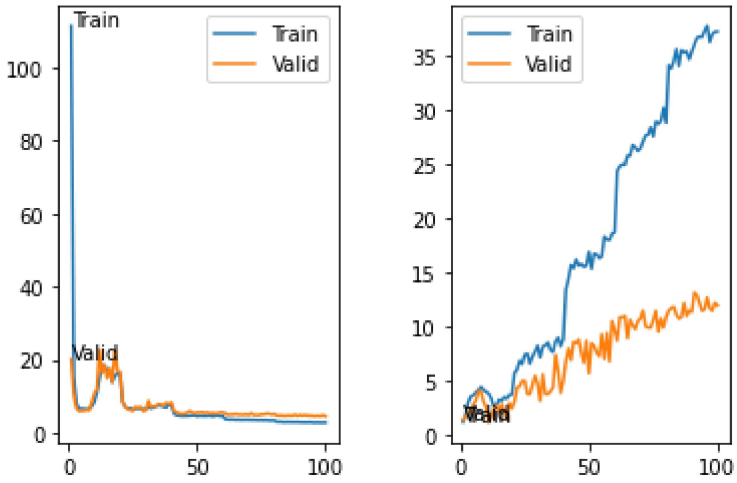
```
TRAIN: Epoch = 79 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 79 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 79 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 79 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 79 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 79 - Batch Number = 27 - Batch Length = 3
Epoch: 79       Learning Rate: 0.001250
        Training Loss: 3.301022         Validation Loss: 4.969206
        Training Accuracy: 30.17% (2015/6679)   Validation Accuracy: 9.94% (83/835)
TRAIN: Epoch = 80 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 80 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 80 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 80 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 80 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 80 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 80 - Batch Number = 27 - Batch Length = 3
Epoch: 80       Learning Rate: 0.001250
        Training Loss: 3.329554         Validation Loss: 5.088140
        Training Accuracy: 28.81% (1924/6679)   Validation Accuracy: 10.90% (91/835)
TRAIN: Epoch = 81 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 81 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 81 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 81 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 81 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 81 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 81 - Batch Number = 27 - Batch Length = 3
Epoch: 81       Learning Rate: 0.000625
        Training Loss: 2.949201         Validation Loss: 4.680870
        Training Accuracy: 34.09% (2277/6679)   Validation Accuracy: 11.50% (96/835)
Saving model: Training Loss: 2.949201 - Validation Loss: 4.680870 - Training Accurac
y: 34.09% (2277/6679) - Validation Accuracy: 11.50% (96/835)
TRAIN: Epoch = 82 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 82 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 82 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 82 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 82 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 82 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 82 - Batch Number = 27 - Batch Length = 3
Epoch: 82       Learning Rate: 0.000625
        Training Loss: 2.961213         Validation Loss: 4.640644
        Training Accuracy: 33.79% (2257/6679)   Validation Accuracy: 11.74% (98/835)
Saving model: Training Loss: 2.961213 - Validation Loss: 4.640644 - Training Accurac
y: 33.79% (2257/6679) - Validation Accuracy: 11.74% (98/835)
TRAIN: Epoch = 83 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 83 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 83 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 83 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 83 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 83 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 83 - Batch Number = 27 - Batch Length = 3
Epoch: 83       Learning Rate: 0.000625
        Training Loss: 2.916459         Validation Loss: 4.694673
        Training Accuracy: 34.48% (2303/6679)   Validation Accuracy: 11.86% (99/835)
TRAIN: Epoch = 84 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 84 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 84 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 84 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 84 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 84 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 84 - Batch Number = 27 - Batch Length = 3
Epoch: 84       Learning Rate: 0.000625
        Training Loss: 2.847469         Validation Loss: 4.528801
        Training Accuracy: 35.57% (2376/6679)   Validation Accuracy: 11.14% (93/835)
Saving model: Training Loss: 2.847469 - Validation Loss: 4.528801 - Training Accurac
y: 35.57% (2376/6679) - Validation Accuracy: 11.14% (93/835)
TRAIN: Epoch = 85 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 85 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 85 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 85 - Batch Number = 209 - Batch Length = 23
```

```
VALIDATION: Epoch = 85 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 85 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 85 - Batch Number = 27 - Batch Length = 3
Epoch: 85       Learning Rate: 0.000625
        Training Loss: 2.915909         Validation Loss: 4.659973
        Training Accuracy: 34.03% (2273/6679)   Validation Accuracy: 10.78% (90/835)
TRAIN: Epoch = 86 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 86 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 86 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 86 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 86 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 86 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 86 - Batch Number = 27 - Batch Length = 3
Epoch: 86       Learning Rate: 0.000625
        Training Loss: 2.872762         Validation Loss: 4.658040
        Training Accuracy: 35.48% (2370/6679)   Validation Accuracy: 10.90% (91/835)
TRAIN: Epoch = 87 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 87 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 87 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 87 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 87 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 87 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 87 - Batch Number = 27 - Batch Length = 3
Epoch: 87       Learning Rate: 0.000625
        Training Loss: 2.865053         Validation Loss: 4.581346
        Training Accuracy: 35.29% (2357/6679)   Validation Accuracy: 12.22% (102/83
5)
TRAIN: Epoch = 88 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 88 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 88 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 88 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 88 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 88 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 88 - Batch Number = 27 - Batch Length = 3
Epoch: 88       Learning Rate: 0.000625
        Training Loss: 2.853018         Validation Loss: 4.889217
        Training Accuracy: 35.36% (2362/6679)   Validation Accuracy: 11.02% (92/835)
TRAIN: Epoch = 89 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 89 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 89 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 89 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 89 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 89 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 89 - Batch Number = 27 - Batch Length = 3
Epoch: 89       Learning Rate: 0.000625
        Training Loss: 2.872388         Validation Loss: 4.559721
        Training Accuracy: 34.65% (2314/6679)   Validation Accuracy: 11.50% (96/835)
TRAIN: Epoch = 90 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 90 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 90 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 90 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 90 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 90 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 90 - Batch Number = 27 - Batch Length = 3
Epoch: 90       Learning Rate: 0.000625
        Training Loss: 2.843107         Validation Loss: 4.819802
        Training Accuracy: 35.35% (2361/6679)   Validation Accuracy: 11.38% (95/835)
TRAIN: Epoch = 91 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 91 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 91 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 91 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 91 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 91 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 91 - Batch Number = 27 - Batch Length = 3
Epoch: 91       Learning Rate: 0.000625
        Training Loss: 2.811568         Validation Loss: 4.621395
        Training Accuracy: 35.98% (2403/6679)   Validation Accuracy: 13.17% (110/83
5)
TRAIN: Epoch = 92 - Batch Number = 1 - Batch Length = 32
```

```
TRAIN: Epoch = 92 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 92 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 92 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 92 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 92 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 92 - Batch Number = 27 - Batch Length = 3
Epoch: 92       Learning Rate: 0.000625
        Training Loss: 2.812950         Validation Loss: 4.771688
        Training Accuracy: 36.62% (2446/6679)   Validation Accuracy: 12.93% (108/83
5)
TRAIN: Epoch = 93 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 93 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 93 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 93 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 93 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 93 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 93 - Batch Number = 27 - Batch Length = 3
Epoch: 93       Learning Rate: 0.000625
        Training Loss: 2.814196         Validation Loss: 4.625530
        Training Accuracy: 36.73% (2453/6679)   Validation Accuracy: 12.22% (102/83
5)
TRAIN: Epoch = 94 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 94 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 94 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 94 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 94 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 94 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 94 - Batch Number = 27 - Batch Length = 3
Epoch: 94       Learning Rate: 0.000625
        Training Loss: 2.853769         Validation Loss: 4.649419
        Training Accuracy: 36.71% (2452/6679)   Validation Accuracy: 11.50% (96/835)
TRAIN: Epoch = 95 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 95 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 95 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 95 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 95 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 95 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 95 - Batch Number = 27 - Batch Length = 3
Epoch: 95       Learning Rate: 0.000625
        Training Loss: 2.754766         Validation Loss: 4.565948
        Training Accuracy: 37.27% (2489/6679)   Validation Accuracy: 11.62% (97/835)
TRAIN: Epoch = 96 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 96 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 96 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 96 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 96 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 96 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 96 - Batch Number = 27 - Batch Length = 3
Epoch: 96       Learning Rate: 0.000625
        Training Loss: 2.737137         Validation Loss: 4.492325
        Training Accuracy: 37.72% (2519/6679)   Validation Accuracy: 12.69% (106/83
5)
Saving model: Training Loss: 2.737137 - Validation Loss: 4.492325 - Training Accurac
y: 37.72% (2519/6679) - Validation Accuracy: 12.69% (106/835)
TRAIN: Epoch = 97 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 97 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 97 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 97 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 97 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 97 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 97 - Batch Number = 27 - Batch Length = 3
Epoch: 97       Learning Rate: 0.000625
        Training Loss: 2.801125         Validation Loss: 4.621341
        Training Accuracy: 36.23% (2420/6679)   Validation Accuracy: 11.74% (98/835)
TRAIN: Epoch = 98 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 98 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 98 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 98 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 98 - Batch Number = 1 - Batch Length = 32
```

```
VALIDATION: Epoch = 98 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 98 - Batch Number = 27 - Batch Length = 3
Epoch: 98       Learning Rate: 0.000625
        Training Loss: 2.723269         Validation Loss: 4.478930
        Training Accuracy: 36.95% (2468/6679)   Validation Accuracy: 11.50% (96/835)
Saving model: Training Loss: 2.723269 - Validation Loss: 4.478930 - Training Accurac
y: 36.95% (2468/6679) - Validation Accuracy: 11.50% (96/835)
TRAIN: Epoch = 99 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 99 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 99 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 99 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 99 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 99 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 99 - Batch Number = 27 - Batch Length = 3
Epoch: 99       Learning Rate: 0.000625
        Training Loss: 2.758056         Validation Loss: 4.769645
        Training Accuracy: 37.18% (2483/6679)   Validation Accuracy: 12.22% (102/83
5)
TRAIN: Epoch = 100 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 100 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 100 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 100 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 100 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 100 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 100 - Batch Number = 27 - Batch Length = 3
Epoch: 100      Learning Rate: 0.000625
        Training Loss: 2.746540         Validation Loss: 4.451214
        Training Accuracy: 37.21% (2485/6679)   Validation Accuracy: 11.98% (100/83
5)
Saving model: Training Loss: 2.746540 - Validation Loss: 4.451214 - Training Accurac
y: 37.21% (2485/6679) - Validation Accuracy: 11.98% (100/835)
```

Out[ ]:
```
<All keys matched successfully>
```



(IMPLEMENTATION) Test the Model

Try out your model on the test dataset of dog images. Use the code cell below to calculate and print the test loss and accuracy. Ensure that your test accuracy is greater than 10%.

In [ ]:
```
def test(loaders, model, criterion, use_cuda):

    # monitor test loss and accuracy
    test_loss = 0.
    correct = 0.
    total = 0.

    model.eval()
    for batch_idx, (data, target) in enumerate(loaders['test']):
        # move to GPU
        if use_cuda:
```

```
            data, target = data.cuda(), target.cuda()
        # forward pass: compute predicted outputs by passing inputs to the model
        output = model(data)
        # calculate the loss
        loss = criterion(output, target)
        # update average test loss
        test_loss = test_loss + ((1 / (batch_idx + 1)) * (loss.data - test_loss))
        # convert output probabilities to predicted class
        pred = output.data.max(1, keepdim=True)[1]
        # compare predictions to true label
        correct += np.sum(np.squeeze(pred.eq(target.data.view_as(pred)).cpu().numpy(
        total += data.size(0)

    print('Test Loss: {:.6f}\n'.format(test_loss))

    print('\nTest Accuracy: %2d%% (%2d/%2d)' % (
        100. * correct / total, correct, total))

# call test function
test(loaders_scratch, model_scratch, criterion_scratch, use_cuda)
```

```
Test Loss: 4.356790


Test Accuracy: 14% (123/836)
```

Step 4: Create a CNN to Classify Dog Breeds (using Transfer Learning)

You will now use transfer learning to create a CNN that can identify dog breed from images. Your CNN must attain at least 60% accuracy on the test set.

(IMPLEMENTATION) Specify Data Loaders for the Dog Dataset

Use the code cell below to write three separate data loaders for the training, validation, and test datasets of dog images (located at `dogImages/train`, `dogImages/valid`, and `dogImages/test`, respectively).

If you like, **you are welcome to use the same data loaders from the previous step**, when you created a CNN from scratch.

In [ ]:

```
## TODO: Specify data loaders
train_transformations = transforms.Compose([
        transforms.Resize(256),
        transforms.ColorJitter(brightness=0.1, contrast=0.1),
        transforms.GaussianBlur(kernel_size=3),
        transforms.RandomRotation(5),
        transforms.RandomCrop(250),
        transforms.RandomHorizontalFlip(0.5),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])]

test_valid_transformations = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])]
```

```
batch_size = 32

train_data = datasets.ImageFolder("data/dog_images/train", transform=train_transform
validation_data = datasets.ImageFolder("data/dog_images/valid", transform=test_valid
test_data = datasets.ImageFolder("data/dog_images/test", transform=test_valid_transf

train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
validation_loader = DataLoader(validation_data, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=True)
loaders_transfer = {"train": train_loader, "valid": validation_loader, "test": test_
```

(IMPLEMENTATION) Model Architecture

Use transfer learning to create a CNN to classify dog breed. Use the code cell below, and save your initialized model as the variable `model_transfer` .

In [ ]:

```
import torchvision.models as models
import torch.nn as nn

## TODO: Specify model architecture
model_transfer = models.vgg16(pretrained=True)

# Freeze all layers
for parameter in model_transfer.parameters():
    parameter.requires_grad = False

last_layer = model_transfer.classifier[-1]
classification_layer = nn.Linear(in_features=last_layer.in_features, out_features=13
model_transfer.classifier[-1] = classification_layer

if use_cuda:
    model_transfer = model_transfer.cuda()

print(model_transfer)
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
```

```
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=133, bias=True)
  )
)
```

Question 5: Outline the steps you took to get to your final CNN architecture and your reasoning at each step. Describe why you think the architecture is suitable for the current problem.

Answer:

I'm using the VGG16 network used earlier as a starting point because achieved good performance on a similar task.
I'll prepare the new network with this steps:

- freeze of all the weights of the model, to avoid changing trained parameters
- replace the last layer with a new trainable layer which outputs 133 classes (parameters of the new layer are not freezed by default)

(IMPLEMENTATION) Specify Loss Function and Optimizer

Use the next code cell to specify a loss function and optimizer. Save the chosen loss function as `criterion_transfer`, and the optimizer as `optimizer_transfer` below.

In [ ]:

```
criterion_transfer = nn.CrossEntropyLoss()
optimizer_transfer = optim.Adam(model_transfer.classifier[-1].parameters(), lr = 0.0
scheduler_transfer = StepLR(optimizer_transfer, step_size=2, gamma=0.5) # Configurin
```

(IMPLEMENTATION) Train and Validate the Model

Train and validate your model in the code cell below. Save the final model parameters at filepath `'model_transfer.pt'`.

In [ ]:

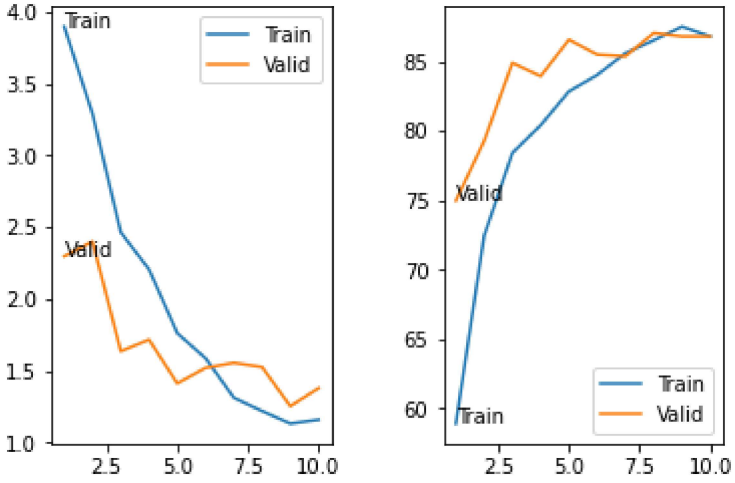
```
# train the model
model_transfer = train(10, loaders_transfer, model_transfer, optimizer_transfer, cri

# load the model that got the best validation accuracy (uncomment the line below)
model_transfer.load_state_dict(torch.load('model_transfer.pt'))
```

```
TRAIN: Epoch = 1 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 1 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 1 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 1 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 1 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 1 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 1 - Batch Number = 27 - Batch Length = 3
Epoch: 1        Learning Rate: 0.010000
        Training Loss: 3.897341         Validation Loss: 2.298716
```

```
        Training Accuracy: 58.89% (3933/6679)   Validation Accuracy: 74.97% (626/83
5)
Saving model: Training Loss: 3.897341 - Validation Loss: 2.298716 - Training Accurac
y: 58.89% (3933/6679) - Validation Accuracy: 74.97% (626/835)
TRAIN: Epoch = 2 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 2 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 2 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 2 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 2 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 2 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 2 - Batch Number = 27 - Batch Length = 3
Epoch: 2        Learning Rate: 0.010000
        Training Loss: 3.291225         Validation Loss: 2.401521
        Training Accuracy: 72.44% (4838/6679)   Validation Accuracy: 79.28% (662/83
5)
TRAIN: Epoch = 3 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 3 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 3 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 3 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 3 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 3 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 3 - Batch Number = 27 - Batch Length = 3
Epoch: 3        Learning Rate: 0.005000
        Training Loss: 2.464664         Validation Loss: 1.636994
        Training Accuracy: 78.41% (5237/6679)   Validation Accuracy: 84.91% (709/83
5)
Saving model: Training Loss: 2.464664 - Validation Loss: 1.636994 - Training Accurac
y: 78.41% (5237/6679) - Validation Accuracy: 84.91% (709/835)
TRAIN: Epoch = 4 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 4 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 4 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 4 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 4 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 4 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 4 - Batch Number = 27 - Batch Length = 3
Epoch: 4        Learning Rate: 0.005000
        Training Loss: 2.206692         Validation Loss: 1.717830
        Training Accuracy: 80.42% (5371/6679)   Validation Accuracy: 83.95% (701/83
5)
TRAIN: Epoch = 5 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 5 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 5 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 5 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 5 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 5 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 5 - Batch Number = 27 - Batch Length = 3
Epoch: 5        Learning Rate: 0.002500
        Training Loss: 1.762786         Validation Loss: 1.412583
        Training Accuracy: 82.86% (5534/6679)   Validation Accuracy: 86.59% (723/83
5)
Saving model: Training Loss: 1.762786 - Validation Loss: 1.412583 - Training Accurac
y: 82.86% (5534/6679) - Validation Accuracy: 86.59% (723/835)
TRAIN: Epoch = 6 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 6 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 6 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 6 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 6 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 6 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 6 - Batch Number = 27 - Batch Length = 3
Epoch: 6        Learning Rate: 0.002500
        Training Loss: 1.588094         Validation Loss: 1.521113
        Training Accuracy: 84.04% (5613/6679)   Validation Accuracy: 85.51% (714/83
5)
TRAIN: Epoch = 7 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 7 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 7 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 7 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 7 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 7 - Batch Number = 25 - Batch Length = 32
```

```
VALIDATION: Epoch = 7 - Batch Number = 27 - Batch Length = 3
Epoch: 7        Learning Rate: 0.001250
        Training Loss: 1.312837         Validation Loss: 1.556090
        Training Accuracy: 85.61% (5718/6679)   Validation Accuracy: 85.39% (713/83
5)
TRAIN: Epoch = 8 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 8 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 8 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 8 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 8 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 8 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 8 - Batch Number = 27 - Batch Length = 3
Epoch: 8        Learning Rate: 0.001250
        Training Loss: 1.220533         Validation Loss: 1.526852
        Training Accuracy: 86.54% (5780/6679)   Validation Accuracy: 87.07% (727/83
5)
TRAIN: Epoch = 9 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 9 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 9 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 9 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 9 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 9 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 9 - Batch Number = 27 - Batch Length = 3
Epoch: 9        Learning Rate: 0.000625
        Training Loss: 1.133112         Validation Loss: 1.253817
        Training Accuracy: 87.53% (5846/6679)   Validation Accuracy: 86.83% (725/83
5)
Saving model: Training Loss: 1.133112 - Validation Loss: 1.253817 - Training Accurac
y: 87.53% (5846/6679) - Validation Accuracy: 86.83% (725/835)
TRAIN: Epoch = 10 - Batch Number = 1 - Batch Length = 32
TRAIN: Epoch = 10 - Batch Number = 100 - Batch Length = 32
TRAIN: Epoch = 10 - Batch Number = 200 - Batch Length = 32
TRAIN: Epoch = 10 - Batch Number = 209 - Batch Length = 23
VALIDATION: Epoch = 10 - Batch Number = 1 - Batch Length = 32
VALIDATION: Epoch = 10 - Batch Number = 25 - Batch Length = 32
VALIDATION: Epoch = 10 - Batch Number = 27 - Batch Length = 3
Epoch: 10       Learning Rate: 0.000625
        Training Loss: 1.159792         Validation Loss: 1.379638
        Training Accuracy: 86.82% (5799/6679)   Validation Accuracy: 86.83% (725/83
5)
```

Out[ ]:
```
<All keys matched successfully>
```



(IMPLEMENTATION) Test the Model

Try out your model on the test dataset of dog images. Use the code cell below to calculate and print the test loss and accuracy. Ensure that your test accuracy is greater than 60%.

In [ ]:
```
test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 1.953998


Test Accuracy: 85% (715/836)
```

(IMPLEMENTATION) Predict Dog Breed with the Model

Write a function that takes an image path as input and returns the dog breed (`Affenpinscher` , `Afghan hound` , etc) that is predicted by your model.

In [ ]:

```
### TODO: Write a function that takes a path to an image as input
### and returns the dog breed that is predicted by the model.

# list of class names by index, i.e. a name can be accessed like class_names[0]
class_names = [item[4:].replace("_", " ") for item in train_data.classes]

def predict_breed_transfer(img_path):
    # load the image and return the predicted breed
    image = Image.open(img_path)

    transformations = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])]

    image = transformations(image) # apply transformations to the image

    image = torch.unsqueeze(image, 0) # add a dimension because the model expect a b

    if use_cuda:
        image = image.cuda()

    model_transfer.eval() # change to evaluation mode, just to be sure (I think that

    predictions = model_transfer(image)

    predictions = torch.squeeze(predictions, 0) # remove the artificial dimension ad

    index = torch.argmax(predictions)

    if use_cuda:
        index = index.cpu()

    return class_names[index.item()] # predicted class index
```

Step 5: Write your Algorithm

Write an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. Then,

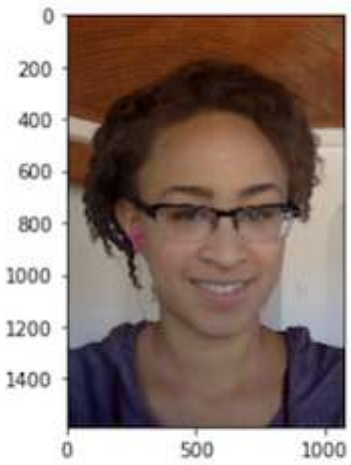
- if a **dog** is detected in the image, return the predicted breed.
- if a **human** is detected in the image, return the resembling dog breed.
- if **neither** is detected in the image, provide output that indicates an error.

You are welcome to write your own functions for detecting humans and dogs in images, but feel free to use the `face_detector` and `human_detector` functions developed above. You are

required to use your CNN from Step 4 to predict dog breed.

Some sample output for our algorithm is provided below, but feel free to design your own user experience!

```
hello, human!
```




```
You look like a ...
Chinese_shar-pei
```

(IMPLEMENTATION) Write your Algorithm

In [ ]:

```
### TODO: Write your algorithm.
### Feel free to use as many code cells as needed.

def run_app(img_path):
    ## handle cases for a human face, dog, and neither
    if dog_detector(img_path):
        dog_breed = predict_breed_transfer(img_path)
        print(f"In the image there is a DOG. It's breed is {dog_breed}")
    elif face_detector(img_path):
        resembling_breed = predict_breed_transfer(img_path)
        print(f"In the image there is an HUMAN. It's resembling breed is {resembling
    else:
        print("ERROR: neither dog neither human detected!")
```

Step 6: Test Your Algorithm

In this section, you will take your new algorithm for a spin! What kind of dog does the algorithm think that *you* look like? If you have a dog, does it predict your dog's breed accurately? If you have a cat, does it mistakenly think that your cat is a dog?

(IMPLEMENTATION) Test Your Algorithm on Sample Images!

Test your algorithm at least six images on your computer. Feel free to use any images you like. Use at least two human and two dog images.

Question 6: Is the output better than you expected :) ? Or worse :(? Provide at least three possible points of improvement for your algorithm.

Answer: (Three possible points for improvement)

1. Using more epochs: after 10 epochs validation loss seems still decreasing and validation accuracy seems still increasing, so probably more epochs will make a better model
2. Reducing the step of the Learning Rate Scheduler to 1, to reduce the learning rate more often
3. Using a different type of Learning Rate Scheduler

In [ ]:

```
## TODO: Execute your algorithm from Step 6 on
## at least 6 images on your computer.
## Feel free to use as many code cells as needed.

## suggested code, below
for file in np.hstack((human_files[:3], dog_files[:3])):
    run_app(file)

images_samples = np.array(["images/Labrador_retriever_06457.jpg", "data/lfw/Sally_Fi

for image_sample in images_samples:
    run_app(image_sample)
```

```
In the image there is an HUMAN. It's resembling breed is Irish wolfhound
In the image there is an HUMAN. It's resembling breed is Welsh springer spaniel
In the image there is an HUMAN. It's resembling breed is Doberman pinscher
In the image there is a DOG. It's breed is Affenpinscher
In the image there is a DOG. It's breed is Affenpinscher
In the image there is a DOG. It's breed is Affenpinscher
In the image there is a DOG. It's breed is Labrador retriever
In the image there is an HUMAN. It's resembling breed is Silky terrier
ERROR: neither dog neither human detected!
```