

dog_app

January 6, 2021

1 Convolutional Neural Networks

1.1 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.

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 [1]: import numpy as np
        from glob import glob

        # load filenames for human and dog images
        human_files = np.array(glob("/data/lfw/*/"))
        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 13233 total human images.

There are 8351 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 [2]: 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.

1.1.1 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 [3]: # 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
```

1.1.2 (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: - Faces in `human_files_short`: 98% - Faces in `dog_files_short`: 17%

```
In [4]: 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.
with tqdm(total=100, desc="Human faces in human_files") as human_perc:
    for i in human_files_short:
        face_found = face_detector(i)
        if face_found:
            human_perc.update()
with tqdm(total=100, desc="Human faces in dog_files") as dog_perc:
    for i in dog_files_short:
        face_found = face_detector(i)
        if face_found:
            dog_perc.update()
```

```
Human faces in human_files:  98%| 98/100 [00:02<00:00, 33.39it/s]
```

```
Human faces in dog_files:  17%| 17/100 [00:29<01:30, 1.09s/it]
```

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

Step 2: Detect Dogs

In this section, we use a [pre-trained model](#) to detect dogs in images.

1.1.3 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 [6]: 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()
```

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.torch/models/vgg16-397923af.pth  
100%|| 553433881/553433881 [00:43<00:00, 12655902.36it/s]
```

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.

1.1.4 (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 [7]: 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
    """

    transform = transforms.Compose([transforms.Resize((224,224)),
                                    transforms.ToTensor(),
                                    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

    #dataset = datasets.ImageFolder(data_dir, transform=transform)
    image = Image.open(img_path)
    img = transform(image)[None, ...]
    output = VGG16(img.cuda() if use_cuda else img)
    _, predictions = torch.max(output,1)
    class_index = predictions.item()
    ## Return the *index* of the predicted class for that image
    return class_index # predicted class index

VGG16_predict(dog_files[0])

Out[7]: 243

```

1.1.5 (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 [8]: ### returns "True" if a dog is detected in the image stored at img_path
def dog_detector(img_path):
    ## TODO: Complete the function.
    predict = VGG16_predict(img_path)
    return predict >= 151 and predict <= 268 # true/false

```

```
dog_detector(dog_files[0])
```

```
Out[8]: True
```

1.1.6 (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: - Dogs in `human_files_short`: 1% - Dogs in `dog_files_short`: 100%

```
In [9]: ### TODO: Test the performance of the dog_detector function  
### on the images in human_files_short and dog_files_short.  
with tqdm(total=100, desc="Dogs in human_files_short") as human_perc:  
    for i in human_files_short:  
        dog_found = dog_detector(i)  
        if dog_found:  
            human_perc.update()  
with tqdm(total=100, desc="Dogs in dog_files_short") as dog_perc:  
    for i in dog_files_short:  
        dog_found = dog_detector(i)  
        if dog_found:  
            dog_perc.update()
```

```
Dogs in human_files_short: 1%|          | 1/100 [00:02<04:41, 2.84s/it]  
Dogs in dog_files_short: 100%|| 100/100 [00:04<00:00, 22.51it/s]
```

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

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
-----------------	--------------------

We also mention that random chance presents an exceptionally low bar: setting aside the fact that the classes are slightly imbalanced, 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!

1.1.7 (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](#)!

```
In [11]: 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
         data_dir = '/data/dog_images'
         batch_size = 64
         num_workers = 0

         train_transforms = transforms.Compose([
             transforms.RandomRotation(40),
             transforms.RandomResizedCrop(224),
             transforms.RandomHorizontalFlip(),
             transforms.ToTensor(),
```



```

        transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
    ])

    test_transforms = transforms.Compose([
        transforms.Resize(255),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
    ])

    data_scratch = {'train': datasets.ImageFolder(data_dir + '/train', transform=train_transforms),
                    'valid': datasets.ImageFolder(data_dir + '/valid', transform=train_transforms),
                    'test': datasets.ImageFolder(data_dir + '/test', transform=test_transforms)}

    loaders_scratch = {'train': DataLoader(data_scratch['train'], batch_size=batch_size, num_workers=num_workers),
                       'valid': DataLoader(data_scratch['valid'], batch_size=batch_size, num_workers=num_workers),
                       'test': DataLoader(data_scratch['test'], batch_size=batch_size, num_workers=num_workers)}

```

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 did Resized-cropping to 224 pixels, since it most architectures trained on ImageNet use this size factor and seems to provide enough details that will be useful for correctly extract patterns for dog breeds classification. Along with resizing and randomly cropping I also added random rotations of 40 degrees both left and right, and random horizontal flips to some images to augment the data and reduce overfitting.

1.1.8 (IMPLEMENTATION) Model Architecture

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

```

In [12]: 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
    def __init__(self):
        super(Net, self).__init__()
        ## Define layers of a CNN
        #Conv layers
        self.firstConv = nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1) #input im
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
        self.lastConv = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1) #input
        #Pool Layer

```

```

        self.pool = nn.MaxPool2d(kernel_size=2, stride=2) #input size / 2
        #Fully Connected layers
        self.input_nodes = (self.lastConv.out_channels*7*7) #8x8 image x 256 channels c
        self.fc1 = nn.Linear(self.input_nodes,4096)
        self.fc2 = nn.Linear(4096,512)
        self.fc3 = nn.Linear(512,133)
        #Dropout
        self.dropout = nn.Dropout(0.5)
    def forward(self, x):
        ## Define forward behavior
        #features
        x = self.pool(F.relu(self.firstConv(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.lastConv(x)))
        #classifier
        #flatten image
        x = self.dropout(x.view(-1,self.input_nodes))
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

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

# instantiate the CNN
model_scratch = Net()
print(model_scratch)
#print(model_scratch.conv5.out_channels)

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

Net(
  (firstConv): Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (lastConv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=12544, out_features=4096, bias=True)
  (fc2): Linear(in_features=4096, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=133, bias=True)
  (dropout): Dropout(p=0.5)
)

```

Question 4: Outline the steps you took to get to your final CNN architecture and your reason-

ing at each step.

Answer: Steps to get the final CNN architecture for scratch. 1. Print out the AlexNet architecture model, read the paper and try a features layer similar to AlexNet with my own classifier layer. 2. Repeat the same previous process with VGG16, but with fewer layers (6 convs and 4 pooling, since the 13 conv layers from VGG16 almost freeze my computer) 3. Created an excel sheet to calculate the output at the end of each layer (row), by entering some parameters like Input Size, Input Channels, Output Channels, Kernel, Padding and Stride. Also calculated the final total parameters at the end of the last Conv layer. 4. Tested with 5 Conv layers and a pooling layer at the end of each one. Also incremented the learning rate to 0.15 and setting dropout to 30. Reached a test accuracy of 43% (by doing some extra training, 50 more epochs, possible overfitting) 5. Checked some thoughts about the Cats vs Dogs classification in the book Deep Learning with Python, by François Chollet. Removed one conv and pooling layer (4 total conv layers), but the last was set to output the same number of inputs channels (256). The dropout was set to 0.5, the learning rate to 0.1 and the epochs back to 100. Also changed the RandomRotation (for data augmentation) from 30 to 40 degrees to increase the possibilities.

The final test accuracy was lower (21%) than previous (43%), but still higher than the requirement (10%)

Since I didn't want to take longer to complete this task I left the last result as it is now (21%). So, to increase it I could increase the epochs, or reduce the learning rate, and (maybe) output more parameters and the of the last conv layer (512 channels and images of 7x7 instead of the current 215 channels and images of 7x7)

1.1.9 (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 [13]: import torch.optim as optim

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

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

1.1.10 (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 [16]: # the following import is required for training to be robust to truncated images
        from PIL import ImageFile
        ImageFile.LOAD_TRUNCATED_IMAGES = True

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

```

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 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_loss))
            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()
            train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.data - train_loss))

        # print training/validation statistics
        if batch_idx % 8 == 7:
            print('Epoch: {} \tBatch: {} \tTraining Loss: {:.6f}'.format(epoch, batch_idx, loss.data))

    #####
    # 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
            output = model(data)
            loss = criterion(output, target)
            valid_loss = valid_loss + ((1 / (batch_idx + 1)) * (loss.data - valid_loss))

        # print training/validation statistics
        if (batch_idx+1) % 8 == 0:
            print('Epoch: {} \tBatch: {} \tValidation Loss: {:.6f}'.format(epoch, batch_idx+1, loss.data))

    ## TODO: save the model if validation loss has decreased
    if valid_loss <= valid_loss_min:
        print('Validation loss decreased {:.6f} --> {:.6f}). Saving model ...'.format(valid_loss_min, valid_loss))
        torch.save(model.state_dict(), save_path)

```

```

        valid_loss_min = valid_loss
    # return trained model
    return model

In [17]: # train the model
        if use_cuda:
            print('Training using GPU ...')
        else:
            print('Training using CPU ...')

        model_scratch = train(100, loaders_scratch, model_scratch, optimizer_scratch,
                               criterion_scratch, use_cuda, 'model_scratch.pt')

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

```

```

Training using GPU ...
Epoch: 1      Batch: 8      Training Loss: 4.950738
Epoch: 1      Batch: 16     Training Loss: 4.920071
Epoch: 1      Batch: 24     Training Loss: 4.910960
Epoch: 1      Batch: 32     Training Loss: 4.905625
Epoch: 1      Batch: 40     Training Loss: 4.900572
Epoch: 1      Batch: 48     Training Loss: 4.898306
Epoch: 1      Batch: 56     Training Loss: 4.894001
Epoch: 1      Batch: 64     Training Loss: 4.892242
Epoch: 1      Batch: 72     Training Loss: 4.891177
Epoch: 1      Batch: 80     Training Loss: 4.889294
Epoch: 1      Batch: 88     Training Loss: 4.887718
Epoch: 1      Batch: 96     Training Loss: 4.886543
Epoch: 1      Batch: 104    Training Loss: 4.885600
Validation loss decreased (inf --> 4.859114). Saving model ...
Validation loss decreased (4.859114 --> 4.846916). Saving model ...
Epoch: 1      Batch: 8      Validation Loss: 4.860590
Epoch: 2      Batch: 8      Training Loss: 4.864447
Epoch: 2      Batch: 16     Training Loss: 4.855685
Epoch: 2      Batch: 24     Training Loss: 4.854825
Epoch: 2      Batch: 32     Training Loss: 4.850051
Epoch: 2      Batch: 40     Training Loss: 4.848224
Epoch: 2      Batch: 48     Training Loss: 4.841012
Epoch: 2      Batch: 56     Training Loss: 4.841877
Epoch: 2      Batch: 64     Training Loss: 4.839192
Epoch: 2      Batch: 72     Training Loss: 4.838304
Epoch: 2      Batch: 80     Training Loss: 4.835614
Epoch: 2      Batch: 88     Training Loss: 4.828882
Epoch: 2      Batch: 96     Training Loss: 4.827730
Epoch: 2      Batch: 104    Training Loss: 4.824921
Validation loss decreased (4.846916 --> 4.802505). Saving model ...
Validation loss decreased (4.802505 --> 4.783944). Saving model ...

```

```

Validation loss decreased (4.783944 --> 4.763907). Saving model ...
Validation loss decreased (4.763907 --> 4.763682). Saving model ...
Epoch: 2      Batch: 8      Validation Loss: 4.782955
Epoch: 3      Batch: 8      Training Loss: 4.784704
Epoch: 3      Batch: 16     Training Loss: 4.777868
Epoch: 3      Batch: 24     Training Loss: 4.783395
Epoch: 3      Batch: 32     Training Loss: 4.784312
Epoch: 3      Batch: 40     Training Loss: 4.787673
Epoch: 3      Batch: 48     Training Loss: 4.790274
Epoch: 3      Batch: 56     Training Loss: 4.783354
Epoch: 3      Batch: 64     Training Loss: 4.779815
Epoch: 3      Batch: 72     Training Loss: 4.780708
Epoch: 3      Batch: 80     Training Loss: 4.780187
Epoch: 3      Batch: 88     Training Loss: 4.775567
Epoch: 3      Batch: 96     Training Loss: 4.772824
Epoch: 3      Batch: 104    Training Loss: 4.774716
Epoch: 3      Batch: 8      Validation Loss: 4.780576
Validation loss decreased (4.763682 --> 4.754713). Saving model ...
Validation loss decreased (4.754713 --> 4.741617). Saving model ...
Epoch: 4      Batch: 8      Training Loss: 4.721614
Epoch: 4      Batch: 16     Training Loss: 4.744046
Epoch: 4      Batch: 24     Training Loss: 4.743595
Epoch: 4      Batch: 32     Training Loss: 4.746141
Epoch: 4      Batch: 40     Training Loss: 4.745605
Epoch: 4      Batch: 48     Training Loss: 4.743405
Epoch: 4      Batch: 56     Training Loss: 4.740699
Epoch: 4      Batch: 64     Training Loss: 4.737442
Epoch: 4      Batch: 72     Training Loss: 4.740972
Epoch: 4      Batch: 80     Training Loss: 4.739629
Epoch: 4      Batch: 88     Training Loss: 4.737436
Epoch: 4      Batch: 96     Training Loss: 4.734359
Epoch: 4      Batch: 104    Training Loss: 4.733608
Validation loss decreased (4.741617 --> 4.689143). Saving model ...
Validation loss decreased (4.689143 --> 4.684460). Saving model ...
Validation loss decreased (4.684460 --> 4.676361). Saving model ...
Validation loss decreased (4.676361 --> 4.673043). Saving model ...
Epoch: 4      Batch: 8      Validation Loss: 4.691146
Epoch: 5      Batch: 8      Training Loss: 4.651179
Epoch: 5      Batch: 16     Training Loss: 4.679965
Epoch: 5      Batch: 24     Training Loss: 4.680078
Epoch: 5      Batch: 32     Training Loss: 4.686786
Epoch: 5      Batch: 40     Training Loss: 4.683903
Epoch: 5      Batch: 48     Training Loss: 4.681348
Epoch: 5      Batch: 56     Training Loss: 4.679383
Epoch: 5      Batch: 64     Training Loss: 4.669789
Epoch: 5      Batch: 72     Training Loss: 4.672075
Epoch: 5      Batch: 80     Training Loss: 4.666189
Epoch: 5      Batch: 88     Training Loss: 4.663914

```

Epoch: 5	Batch: 96	Training Loss: 4.665305
Epoch: 5	Batch: 104	Training Loss: 4.661838
Validation loss decreased (4.673043 --> 4.603216). Saving model ...		
Validation loss decreased (4.603216 --> 4.557189). Saving model ...		
Epoch: 5	Batch: 8	Validation Loss: 4.642361
Epoch: 6	Batch: 8	Training Loss: 4.654899
Epoch: 6	Batch: 16	Training Loss: 4.650558
Epoch: 6	Batch: 24	Training Loss: 4.643580
Epoch: 6	Batch: 32	Training Loss: 4.635230
Epoch: 6	Batch: 40	Training Loss: 4.624289
Epoch: 6	Batch: 48	Training Loss: 4.618072
Epoch: 6	Batch: 56	Training Loss: 4.613575
Epoch: 6	Batch: 64	Training Loss: 4.611708
Epoch: 6	Batch: 72	Training Loss: 4.609276
Epoch: 6	Batch: 80	Training Loss: 4.604810
Epoch: 6	Batch: 88	Training Loss: 4.595790
Epoch: 6	Batch: 96	Training Loss: 4.595573
Epoch: 6	Batch: 104	Training Loss: 4.595631
Epoch: 6	Batch: 8	Validation Loss: 4.603721
Epoch: 7	Batch: 8	Training Loss: 4.522449
Epoch: 7	Batch: 16	Training Loss: 4.521842
Epoch: 7	Batch: 24	Training Loss: 4.541240
Epoch: 7	Batch: 32	Training Loss: 4.546033
Epoch: 7	Batch: 40	Training Loss: 4.528154
Epoch: 7	Batch: 48	Training Loss: 4.525290
Epoch: 7	Batch: 56	Training Loss: 4.519054
Epoch: 7	Batch: 64	Training Loss: 4.519035
Epoch: 7	Batch: 72	Training Loss: 4.517717
Epoch: 7	Batch: 80	Training Loss: 4.522759
Epoch: 7	Batch: 88	Training Loss: 4.519713
Epoch: 7	Batch: 96	Training Loss: 4.521208
Epoch: 7	Batch: 104	Training Loss: 4.523515
Validation loss decreased (4.557189 --> 4.503601). Saving model ...		
Epoch: 7	Batch: 8	Validation Loss: 4.567509
Epoch: 8	Batch: 8	Training Loss: 4.497904
Epoch: 8	Batch: 16	Training Loss: 4.490761
Epoch: 8	Batch: 24	Training Loss: 4.483775
Epoch: 8	Batch: 32	Training Loss: 4.482175
Epoch: 8	Batch: 40	Training Loss: 4.481651
Epoch: 8	Batch: 48	Training Loss: 4.481449
Epoch: 8	Batch: 56	Training Loss: 4.482252
Epoch: 8	Batch: 64	Training Loss: 4.478927
Epoch: 8	Batch: 72	Training Loss: 4.479158
Epoch: 8	Batch: 80	Training Loss: 4.482637
Epoch: 8	Batch: 88	Training Loss: 4.483826
Epoch: 8	Batch: 96	Training Loss: 4.488967
Epoch: 8	Batch: 104	Training Loss: 4.485515
Epoch: 8	Batch: 8	Validation Loss: 4.567316

```

Validation loss decreased (4.503601 --> 4.488793). Saving model ...
Epoch: 9      Batch: 8      Training Loss: 4.484379
Epoch: 9      Batch: 16     Training Loss: 4.477128
Epoch: 9      Batch: 24     Training Loss: 4.436920
Epoch: 9      Batch: 32     Training Loss: 4.442531
Epoch: 9      Batch: 40     Training Loss: 4.456889
Epoch: 9      Batch: 48     Training Loss: 4.462322
Epoch: 9      Batch: 56     Training Loss: 4.455269
Epoch: 9      Batch: 64     Training Loss: 4.458827
Epoch: 9      Batch: 72     Training Loss: 4.452344
Epoch: 9      Batch: 80     Training Loss: 4.448497
Epoch: 9      Batch: 88     Training Loss: 4.445886
Epoch: 9      Batch: 96     Training Loss: 4.446889
Epoch: 9      Batch: 104    Training Loss: 4.444340
Validation loss decreased (4.488793 --> 4.372999). Saving model ...
Epoch: 9      Batch: 8      Validation Loss: 4.498720
Epoch: 10     Batch: 8      Training Loss: 4.439479
Epoch: 10     Batch: 16     Training Loss: 4.418725
Epoch: 10     Batch: 24     Training Loss: 4.416855
Epoch: 10     Batch: 32     Training Loss: 4.408002
Epoch: 10     Batch: 40     Training Loss: 4.402724
Epoch: 10     Batch: 48     Training Loss: 4.393609
Epoch: 10     Batch: 56     Training Loss: 4.391093
Epoch: 10     Batch: 64     Training Loss: 4.390497
Epoch: 10     Batch: 72     Training Loss: 4.389392
Epoch: 10     Batch: 80     Training Loss: 4.387359
Epoch: 10     Batch: 88     Training Loss: 4.392704
Epoch: 10     Batch: 96     Training Loss: 4.392684
Epoch: 10     Batch: 104    Training Loss: 4.395888
Validation loss decreased (4.372999 --> 4.361280). Saving model ...
Epoch: 10     Batch: 8      Validation Loss: 4.408828
Epoch: 11     Batch: 8      Training Loss: 4.375345
Epoch: 11     Batch: 16     Training Loss: 4.393040
Epoch: 11     Batch: 24     Training Loss: 4.397876
Epoch: 11     Batch: 32     Training Loss: 4.394171
Epoch: 11     Batch: 40     Training Loss: 4.381237
Epoch: 11     Batch: 48     Training Loss: 4.380563
Epoch: 11     Batch: 56     Training Loss: 4.375506
Epoch: 11     Batch: 64     Training Loss: 4.371209
Epoch: 11     Batch: 72     Training Loss: 4.372840
Epoch: 11     Batch: 80     Training Loss: 4.369634
Epoch: 11     Batch: 88     Training Loss: 4.368632
Epoch: 11     Batch: 96     Training Loss: 4.370170
Epoch: 11     Batch: 104    Training Loss: 4.370428
Validation loss decreased (4.361280 --> 4.226694). Saving model ...
Epoch: 11     Batch: 8      Validation Loss: 4.351790
Epoch: 12     Batch: 8      Training Loss: 4.322908
Epoch: 12     Batch: 16     Training Loss: 4.336759

```


Epoch: 12	Batch: 24	Training Loss: 4.352860
Epoch: 12	Batch: 32	Training Loss: 4.352385
Epoch: 12	Batch: 40	Training Loss: 4.355853
Epoch: 12	Batch: 48	Training Loss: 4.349832
Epoch: 12	Batch: 56	Training Loss: 4.345829
Epoch: 12	Batch: 64	Training Loss: 4.341095
Epoch: 12	Batch: 72	Training Loss: 4.336979
Epoch: 12	Batch: 80	Training Loss: 4.329981
Epoch: 12	Batch: 88	Training Loss: 4.337068
Epoch: 12	Batch: 96	Training Loss: 4.339734
Epoch: 12	Batch: 104	Training Loss: 4.337749
Epoch: 12	Batch: 8	Validation Loss: 4.533208
Epoch: 13	Batch: 8	Training Loss: 4.362005
Epoch: 13	Batch: 16	Training Loss: 4.295546
Epoch: 13	Batch: 24	Training Loss: 4.303682
Epoch: 13	Batch: 32	Training Loss: 4.307255
Epoch: 13	Batch: 40	Training Loss: 4.301492
Epoch: 13	Batch: 48	Training Loss: 4.311739
Epoch: 13	Batch: 56	Training Loss: 4.314414
Epoch: 13	Batch: 64	Training Loss: 4.315077
Epoch: 13	Batch: 72	Training Loss: 4.317226
Epoch: 13	Batch: 80	Training Loss: 4.316068
Epoch: 13	Batch: 88	Training Loss: 4.314120
Epoch: 13	Batch: 96	Training Loss: 4.310042
Epoch: 13	Batch: 104	Training Loss: 4.307047
Epoch: 13	Batch: 8	Validation Loss: 4.365065
Epoch: 14	Batch: 8	Training Loss: 4.367738
Epoch: 14	Batch: 16	Training Loss: 4.320339
Epoch: 14	Batch: 24	Training Loss: 4.284220
Epoch: 14	Batch: 32	Training Loss: 4.287804
Epoch: 14	Batch: 40	Training Loss: 4.279252
Epoch: 14	Batch: 48	Training Loss: 4.281742
Epoch: 14	Batch: 56	Training Loss: 4.275989
Epoch: 14	Batch: 64	Training Loss: 4.276533
Epoch: 14	Batch: 72	Training Loss: 4.269255
Epoch: 14	Batch: 80	Training Loss: 4.272816
Epoch: 14	Batch: 88	Training Loss: 4.271692
Epoch: 14	Batch: 96	Training Loss: 4.270006
Epoch: 14	Batch: 104	Training Loss: 4.266318
Epoch: 14	Batch: 8	Validation Loss: 4.457117
Epoch: 15	Batch: 8	Training Loss: 4.187098
Epoch: 15	Batch: 16	Training Loss: 4.214290
Epoch: 15	Batch: 24	Training Loss: 4.262614
Epoch: 15	Batch: 32	Training Loss: 4.280627
Epoch: 15	Batch: 40	Training Loss: 4.271636
Epoch: 15	Batch: 48	Training Loss: 4.264586
Epoch: 15	Batch: 56	Training Loss: 4.269547
Epoch: 15	Batch: 64	Training Loss: 4.268991

Epoch: 15	Batch: 72	Training Loss: 4.267837
Epoch: 15	Batch: 80	Training Loss: 4.262891
Epoch: 15	Batch: 88	Training Loss: 4.251660
Epoch: 15	Batch: 96	Training Loss: 4.247557
Epoch: 15	Batch: 104	Training Loss: 4.242321
Epoch: 15	Batch: 8	Validation Loss: 4.364682
Epoch: 16	Batch: 8	Training Loss: 4.286006
Epoch: 16	Batch: 16	Training Loss: 4.268202
Epoch: 16	Batch: 24	Training Loss: 4.241388
Epoch: 16	Batch: 32	Training Loss: 4.231898
Epoch: 16	Batch: 40	Training Loss: 4.218283
Epoch: 16	Batch: 48	Training Loss: 4.201633
Epoch: 16	Batch: 56	Training Loss: 4.199611
Epoch: 16	Batch: 64	Training Loss: 4.208216
Epoch: 16	Batch: 72	Training Loss: 4.213280
Epoch: 16	Batch: 80	Training Loss: 4.217071
Epoch: 16	Batch: 88	Training Loss: 4.220840
Epoch: 16	Batch: 96	Training Loss: 4.215586
Epoch: 16	Batch: 104	Training Loss: 4.215780
Epoch: 16	Batch: 8	Validation Loss: 4.455976
Epoch: 17	Batch: 8	Training Loss: 4.171691
Epoch: 17	Batch: 16	Training Loss: 4.210182
Epoch: 17	Batch: 24	Training Loss: 4.203785
Epoch: 17	Batch: 32	Training Loss: 4.189828
Epoch: 17	Batch: 40	Training Loss: 4.189648
Epoch: 17	Batch: 48	Training Loss: 4.183677
Epoch: 17	Batch: 56	Training Loss: 4.189120
Epoch: 17	Batch: 64	Training Loss: 4.195908
Epoch: 17	Batch: 72	Training Loss: 4.188668
Epoch: 17	Batch: 80	Training Loss: 4.197115
Epoch: 17	Batch: 88	Training Loss: 4.194785
Epoch: 17	Batch: 96	Training Loss: 4.190123
Epoch: 17	Batch: 104	Training Loss: 4.190114
Epoch: 17	Batch: 8	Validation Loss: 4.327612
Epoch: 18	Batch: 8	Training Loss: 4.157388
Epoch: 18	Batch: 16	Training Loss: 4.129417
Epoch: 18	Batch: 24	Training Loss: 4.134620
Epoch: 18	Batch: 32	Training Loss: 4.137045
Epoch: 18	Batch: 40	Training Loss: 4.124399
Epoch: 18	Batch: 48	Training Loss: 4.120279
Epoch: 18	Batch: 56	Training Loss: 4.119178
Epoch: 18	Batch: 64	Training Loss: 4.120257
Epoch: 18	Batch: 72	Training Loss: 4.134624
Epoch: 18	Batch: 80	Training Loss: 4.137491
Epoch: 18	Batch: 88	Training Loss: 4.135799
Epoch: 18	Batch: 96	Training Loss: 4.138006
Epoch: 18	Batch: 104	Training Loss: 4.138586

Validation loss decreased (4.226694 --> 4.159998). Saving model ...

Epoch: 18	Batch: 8	Validation Loss: 4.341642
Epoch: 19	Batch: 8	Training Loss: 4.142940
Epoch: 19	Batch: 16	Training Loss: 4.101075
Epoch: 19	Batch: 24	Training Loss: 4.095407
Epoch: 19	Batch: 32	Training Loss: 4.101656
Epoch: 19	Batch: 40	Training Loss: 4.108685
Epoch: 19	Batch: 48	Training Loss: 4.118424
Epoch: 19	Batch: 56	Training Loss: 4.120990
Epoch: 19	Batch: 64	Training Loss: 4.123382
Epoch: 19	Batch: 72	Training Loss: 4.127234
Epoch: 19	Batch: 80	Training Loss: 4.126948
Epoch: 19	Batch: 88	Training Loss: 4.130721
Epoch: 19	Batch: 96	Training Loss: 4.134282
Epoch: 19	Batch: 104	Training Loss: 4.135950
Validation loss decreased (4.159998 --> 3.935259). Saving model ...		
Epoch: 19	Batch: 8	Validation Loss: 4.292964
Epoch: 20	Batch: 8	Training Loss: 3.991362
Epoch: 20	Batch: 16	Training Loss: 4.041692
Epoch: 20	Batch: 24	Training Loss: 4.081853
Epoch: 20	Batch: 32	Training Loss: 4.071826
Epoch: 20	Batch: 40	Training Loss: 4.091940
Epoch: 20	Batch: 48	Training Loss: 4.102663
Epoch: 20	Batch: 56	Training Loss: 4.102640
Epoch: 20	Batch: 64	Training Loss: 4.105545
Epoch: 20	Batch: 72	Training Loss: 4.116432
Epoch: 20	Batch: 80	Training Loss: 4.112274
Epoch: 20	Batch: 88	Training Loss: 4.109227
Epoch: 20	Batch: 96	Training Loss: 4.105034
Epoch: 20	Batch: 104	Training Loss: 4.112779
Epoch: 20	Batch: 8	Validation Loss: 4.139682
Epoch: 21	Batch: 8	Training Loss: 4.024659
Epoch: 21	Batch: 16	Training Loss: 4.049232
Epoch: 21	Batch: 24	Training Loss: 4.039968
Epoch: 21	Batch: 32	Training Loss: 4.068321
Epoch: 21	Batch: 40	Training Loss: 4.062661
Epoch: 21	Batch: 48	Training Loss: 4.063417
Epoch: 21	Batch: 56	Training Loss: 4.061507
Epoch: 21	Batch: 64	Training Loss: 4.065015
Epoch: 21	Batch: 72	Training Loss: 4.062055
Epoch: 21	Batch: 80	Training Loss: 4.062562
Epoch: 21	Batch: 88	Training Loss: 4.066975
Epoch: 21	Batch: 96	Training Loss: 4.068645
Epoch: 21	Batch: 104	Training Loss: 4.068302
Epoch: 21	Batch: 8	Validation Loss: 4.414538
Epoch: 22	Batch: 8	Training Loss: 4.137849
Epoch: 22	Batch: 16	Training Loss: 4.105227
Epoch: 22	Batch: 24	Training Loss: 4.070393
Epoch: 22	Batch: 32	Training Loss: 4.101192

Epoch: 22	Batch: 40	Training Loss: 4.085562
Epoch: 22	Batch: 48	Training Loss: 4.083662
Epoch: 22	Batch: 56	Training Loss: 4.077438
Epoch: 22	Batch: 64	Training Loss: 4.079983
Epoch: 22	Batch: 72	Training Loss: 4.072296
Epoch: 22	Batch: 80	Training Loss: 4.065526
Epoch: 22	Batch: 88	Training Loss: 4.066249
Epoch: 22	Batch: 96	Training Loss: 4.058273
Epoch: 22	Batch: 104	Training Loss: 4.064249
Epoch: 22	Batch: 8	Validation Loss: 4.242671
Epoch: 23	Batch: 8	Training Loss: 4.012147
Epoch: 23	Batch: 16	Training Loss: 4.008953
Epoch: 23	Batch: 24	Training Loss: 4.029284
Epoch: 23	Batch: 32	Training Loss: 4.008623
Epoch: 23	Batch: 40	Training Loss: 4.026269
Epoch: 23	Batch: 48	Training Loss: 4.036721
Epoch: 23	Batch: 56	Training Loss: 4.037492
Epoch: 23	Batch: 64	Training Loss: 4.034243
Epoch: 23	Batch: 72	Training Loss: 4.031055
Epoch: 23	Batch: 80	Training Loss: 4.030395
Epoch: 23	Batch: 88	Training Loss: 4.026766
Epoch: 23	Batch: 96	Training Loss: 4.027915
Epoch: 23	Batch: 104	Training Loss: 4.031282
Epoch: 23	Batch: 8	Validation Loss: 4.256061
Epoch: 24	Batch: 8	Training Loss: 3.985459
Epoch: 24	Batch: 16	Training Loss: 3.998773
Epoch: 24	Batch: 24	Training Loss: 4.025755
Epoch: 24	Batch: 32	Training Loss: 4.012599
Epoch: 24	Batch: 40	Training Loss: 4.012181
Epoch: 24	Batch: 48	Training Loss: 4.017027
Epoch: 24	Batch: 56	Training Loss: 4.003299
Epoch: 24	Batch: 64	Training Loss: 3.992682
Epoch: 24	Batch: 72	Training Loss: 4.007679
Epoch: 24	Batch: 80	Training Loss: 4.009932
Epoch: 24	Batch: 88	Training Loss: 4.010566
Epoch: 24	Batch: 96	Training Loss: 4.007709
Epoch: 24	Batch: 104	Training Loss: 4.007520
Epoch: 24	Batch: 8	Validation Loss: 4.178884
Epoch: 25	Batch: 8	Training Loss: 3.940858
Epoch: 25	Batch: 16	Training Loss: 3.912397
Epoch: 25	Batch: 24	Training Loss: 3.927195
Epoch: 25	Batch: 32	Training Loss: 3.937899
Epoch: 25	Batch: 40	Training Loss: 3.921660
Epoch: 25	Batch: 48	Training Loss: 3.937729
Epoch: 25	Batch: 56	Training Loss: 3.941847
Epoch: 25	Batch: 64	Training Loss: 3.955219
Epoch: 25	Batch: 72	Training Loss: 3.960403
Epoch: 25	Batch: 80	Training Loss: 3.958036

Epoch: 25	Batch: 88	Training Loss: 3.958109
Epoch: 25	Batch: 96	Training Loss: 3.955986
Epoch: 25	Batch: 104	Training Loss: 3.961507
Epoch: 25	Batch: 8	Validation Loss: 4.101764
Epoch: 26	Batch: 8	Training Loss: 3.923777
Epoch: 26	Batch: 16	Training Loss: 3.945693
Epoch: 26	Batch: 24	Training Loss: 3.949188
Epoch: 26	Batch: 32	Training Loss: 3.942964
Epoch: 26	Batch: 40	Training Loss: 3.951647
Epoch: 26	Batch: 48	Training Loss: 3.950740
Epoch: 26	Batch: 56	Training Loss: 3.943997
Epoch: 26	Batch: 64	Training Loss: 3.952528
Epoch: 26	Batch: 72	Training Loss: 3.963936
Epoch: 26	Batch: 80	Training Loss: 3.956064
Epoch: 26	Batch: 88	Training Loss: 3.956804
Epoch: 26	Batch: 96	Training Loss: 3.960930
Epoch: 26	Batch: 104	Training Loss: 3.960171
Epoch: 26	Batch: 8	Validation Loss: 4.185433
Epoch: 27	Batch: 8	Training Loss: 3.899138
Epoch: 27	Batch: 16	Training Loss: 3.881939
Epoch: 27	Batch: 24	Training Loss: 3.878375
Epoch: 27	Batch: 32	Training Loss: 3.873230
Epoch: 27	Batch: 40	Training Loss: 3.892718
Epoch: 27	Batch: 48	Training Loss: 3.885846
Epoch: 27	Batch: 56	Training Loss: 3.888448
Epoch: 27	Batch: 64	Training Loss: 3.893593
Epoch: 27	Batch: 72	Training Loss: 3.907808
Epoch: 27	Batch: 80	Training Loss: 3.911479
Epoch: 27	Batch: 88	Training Loss: 3.922341
Epoch: 27	Batch: 96	Training Loss: 3.928210
Epoch: 27	Batch: 104	Training Loss: 3.923094
Epoch: 27	Batch: 8	Validation Loss: 4.106945
Epoch: 28	Batch: 8	Training Loss: 3.807814
Epoch: 28	Batch: 16	Training Loss: 3.857630
Epoch: 28	Batch: 24	Training Loss: 3.884119
Epoch: 28	Batch: 32	Training Loss: 3.891406
Epoch: 28	Batch: 40	Training Loss: 3.887186
Epoch: 28	Batch: 48	Training Loss: 3.903769
Epoch: 28	Batch: 56	Training Loss: 3.900916
Epoch: 28	Batch: 64	Training Loss: 3.904927
Epoch: 28	Batch: 72	Training Loss: 3.896326
Epoch: 28	Batch: 80	Training Loss: 3.898197
Epoch: 28	Batch: 88	Training Loss: 3.900249
Epoch: 28	Batch: 96	Training Loss: 3.901185
Epoch: 28	Batch: 104	Training Loss: 3.906138
Epoch: 28	Batch: 8	Validation Loss: 4.194258
Epoch: 29	Batch: 8	Training Loss: 3.964470
Epoch: 29	Batch: 16	Training Loss: 3.899947

Epoch: 29	Batch: 24	Training Loss: 3.893852
Epoch: 29	Batch: 32	Training Loss: 3.907084
Epoch: 29	Batch: 40	Training Loss: 3.885828
Epoch: 29	Batch: 48	Training Loss: 3.890189
Epoch: 29	Batch: 56	Training Loss: 3.892578
Epoch: 29	Batch: 64	Training Loss: 3.894359
Epoch: 29	Batch: 72	Training Loss: 3.895208
Epoch: 29	Batch: 80	Training Loss: 3.894407
Epoch: 29	Batch: 88	Training Loss: 3.891220
Epoch: 29	Batch: 96	Training Loss: 3.895340
Epoch: 29	Batch: 104	Training Loss: 3.893799
Epoch: 29	Batch: 8	Validation Loss: 4.137931
Epoch: 30	Batch: 8	Training Loss: 3.872389
Epoch: 30	Batch: 16	Training Loss: 3.866956
Epoch: 30	Batch: 24	Training Loss: 3.870644
Epoch: 30	Batch: 32	Training Loss: 3.858610
Epoch: 30	Batch: 40	Training Loss: 3.865568
Epoch: 30	Batch: 48	Training Loss: 3.871304
Epoch: 30	Batch: 56	Training Loss: 3.869263
Epoch: 30	Batch: 64	Training Loss: 3.862974
Epoch: 30	Batch: 72	Training Loss: 3.874328
Epoch: 30	Batch: 80	Training Loss: 3.869611
Epoch: 30	Batch: 88	Training Loss: 3.869527
Epoch: 30	Batch: 96	Training Loss: 3.867319
Epoch: 30	Batch: 104	Training Loss: 3.864706
Validation loss decreased (3.935259 --> 3.855666). Saving model ...		
Epoch: 30	Batch: 8	Validation Loss: 4.048130
Epoch: 31	Batch: 8	Training Loss: 3.843569
Epoch: 31	Batch: 16	Training Loss: 3.833753
Epoch: 31	Batch: 24	Training Loss: 3.841473
Epoch: 31	Batch: 32	Training Loss: 3.869939
Epoch: 31	Batch: 40	Training Loss: 3.877170
Epoch: 31	Batch: 48	Training Loss: 3.868746
Epoch: 31	Batch: 56	Training Loss: 3.863364
Epoch: 31	Batch: 64	Training Loss: 3.855541
Epoch: 31	Batch: 72	Training Loss: 3.854763
Epoch: 31	Batch: 80	Training Loss: 3.845462
Epoch: 31	Batch: 88	Training Loss: 3.847042
Epoch: 31	Batch: 96	Training Loss: 3.842203
Epoch: 31	Batch: 104	Training Loss: 3.833656
Epoch: 31	Batch: 8	Validation Loss: 4.132536
Epoch: 32	Batch: 8	Training Loss: 3.734853
Epoch: 32	Batch: 16	Training Loss: 3.809427
Epoch: 32	Batch: 24	Training Loss: 3.811942
Epoch: 32	Batch: 32	Training Loss: 3.797264
Epoch: 32	Batch: 40	Training Loss: 3.813999
Epoch: 32	Batch: 48	Training Loss: 3.803384
Epoch: 32	Batch: 56	Training Loss: 3.800055

Epoch: 32	Batch: 64	Training Loss: 3.805225
Epoch: 32	Batch: 72	Training Loss: 3.806744
Epoch: 32	Batch: 80	Training Loss: 3.804704
Epoch: 32	Batch: 88	Training Loss: 3.799562
Epoch: 32	Batch: 96	Training Loss: 3.804862
Epoch: 32	Batch: 104	Training Loss: 3.800869
Epoch: 32	Batch: 8	Validation Loss: 4.174886
Epoch: 33	Batch: 8	Training Loss: 3.923558
Epoch: 33	Batch: 16	Training Loss: 3.811311
Epoch: 33	Batch: 24	Training Loss: 3.802059
Epoch: 33	Batch: 32	Training Loss: 3.788961
Epoch: 33	Batch: 40	Training Loss: 3.791160
Epoch: 33	Batch: 48	Training Loss: 3.805531
Epoch: 33	Batch: 56	Training Loss: 3.802354
Epoch: 33	Batch: 64	Training Loss: 3.804259
Epoch: 33	Batch: 72	Training Loss: 3.798820
Epoch: 33	Batch: 80	Training Loss: 3.803516
Epoch: 33	Batch: 88	Training Loss: 3.800412
Epoch: 33	Batch: 96	Training Loss: 3.797614
Epoch: 33	Batch: 104	Training Loss: 3.802324
Epoch: 33	Batch: 8	Validation Loss: 4.080067
Epoch: 34	Batch: 8	Training Loss: 3.840205
Epoch: 34	Batch: 16	Training Loss: 3.817339
Epoch: 34	Batch: 24	Training Loss: 3.777712
Epoch: 34	Batch: 32	Training Loss: 3.799210
Epoch: 34	Batch: 40	Training Loss: 3.786024
Epoch: 34	Batch: 48	Training Loss: 3.787868
Epoch: 34	Batch: 56	Training Loss: 3.785260
Epoch: 34	Batch: 64	Training Loss: 3.782628
Epoch: 34	Batch: 72	Training Loss: 3.780926
Epoch: 34	Batch: 80	Training Loss: 3.780621
Epoch: 34	Batch: 88	Training Loss: 3.778065
Epoch: 34	Batch: 96	Training Loss: 3.780795
Epoch: 34	Batch: 104	Training Loss: 3.775677
Epoch: 34	Batch: 8	Validation Loss: 4.148314
Epoch: 35	Batch: 8	Training Loss: 3.658389
Epoch: 35	Batch: 16	Training Loss: 3.711934
Epoch: 35	Batch: 24	Training Loss: 3.713428
Epoch: 35	Batch: 32	Training Loss: 3.713822
Epoch: 35	Batch: 40	Training Loss: 3.692571
Epoch: 35	Batch: 48	Training Loss: 3.706054
Epoch: 35	Batch: 56	Training Loss: 3.705535
Epoch: 35	Batch: 64	Training Loss: 3.719198
Epoch: 35	Batch: 72	Training Loss: 3.722713
Epoch: 35	Batch: 80	Training Loss: 3.723822
Epoch: 35	Batch: 88	Training Loss: 3.721053
Epoch: 35	Batch: 96	Training Loss: 3.719155
Epoch: 35	Batch: 104	Training Loss: 3.729067

Epoch: 35	Batch: 8	Validation Loss: 3.990587
Epoch: 36	Batch: 8	Training Loss: 3.650341
Epoch: 36	Batch: 16	Training Loss: 3.724934
Epoch: 36	Batch: 24	Training Loss: 3.734742
Epoch: 36	Batch: 32	Training Loss: 3.747609
Epoch: 36	Batch: 40	Training Loss: 3.757342
Epoch: 36	Batch: 48	Training Loss: 3.744898
Epoch: 36	Batch: 56	Training Loss: 3.741898
Epoch: 36	Batch: 64	Training Loss: 3.732677
Epoch: 36	Batch: 72	Training Loss: 3.736023
Epoch: 36	Batch: 80	Training Loss: 3.734341
Epoch: 36	Batch: 88	Training Loss: 3.735549
Epoch: 36	Batch: 96	Training Loss: 3.734040
Epoch: 36	Batch: 104	Training Loss: 3.733907
Validation loss decreased (3.855666 --> 3.852999). Saving model ...		
Epoch: 36	Batch: 8	Validation Loss: 3.985095
Epoch: 37	Batch: 8	Training Loss: 3.603969
Epoch: 37	Batch: 16	Training Loss: 3.676447
Epoch: 37	Batch: 24	Training Loss: 3.675760
Epoch: 37	Batch: 32	Training Loss: 3.708437
Epoch: 37	Batch: 40	Training Loss: 3.710073
Epoch: 37	Batch: 48	Training Loss: 3.710367
Epoch: 37	Batch: 56	Training Loss: 3.708486
Epoch: 37	Batch: 64	Training Loss: 3.709995
Epoch: 37	Batch: 72	Training Loss: 3.705628
Epoch: 37	Batch: 80	Training Loss: 3.716906
Epoch: 37	Batch: 88	Training Loss: 3.723546
Epoch: 37	Batch: 96	Training Loss: 3.712880
Epoch: 37	Batch: 104	Training Loss: 3.706591
Epoch: 37	Batch: 8	Validation Loss: 4.100803
Epoch: 38	Batch: 8	Training Loss: 3.796095
Epoch: 38	Batch: 16	Training Loss: 3.730492
Epoch: 38	Batch: 24	Training Loss: 3.681269
Epoch: 38	Batch: 32	Training Loss: 3.668859
Epoch: 38	Batch: 40	Training Loss: 3.652880
Epoch: 38	Batch: 48	Training Loss: 3.635301
Epoch: 38	Batch: 56	Training Loss: 3.627818
Epoch: 38	Batch: 64	Training Loss: 3.628731
Epoch: 38	Batch: 72	Training Loss: 3.638401
Epoch: 38	Batch: 80	Training Loss: 3.640868
Epoch: 38	Batch: 88	Training Loss: 3.650222
Epoch: 38	Batch: 96	Training Loss: 3.656876
Epoch: 38	Batch: 104	Training Loss: 3.655005
Epoch: 38	Batch: 8	Validation Loss: 4.024691
Epoch: 39	Batch: 8	Training Loss: 3.640385
Epoch: 39	Batch: 16	Training Loss: 3.660162
Epoch: 39	Batch: 24	Training Loss: 3.661414
Epoch: 39	Batch: 32	Training Loss: 3.665864

Epoch: 39	Batch: 40	Training Loss: 3.670289
Epoch: 39	Batch: 48	Training Loss: 3.662531
Epoch: 39	Batch: 56	Training Loss: 3.655337
Epoch: 39	Batch: 64	Training Loss: 3.658392
Epoch: 39	Batch: 72	Training Loss: 3.669386
Epoch: 39	Batch: 80	Training Loss: 3.672105
Epoch: 39	Batch: 88	Training Loss: 3.665478
Epoch: 39	Batch: 96	Training Loss: 3.662567
Epoch: 39	Batch: 104	Training Loss: 3.660330
Epoch: 39	Batch: 8	Validation Loss: 4.301437
Epoch: 40	Batch: 8	Training Loss: 3.624210
Epoch: 40	Batch: 16	Training Loss: 3.642632
Epoch: 40	Batch: 24	Training Loss: 3.646333
Epoch: 40	Batch: 32	Training Loss: 3.635532
Epoch: 40	Batch: 40	Training Loss: 3.630101
Epoch: 40	Batch: 48	Training Loss: 3.621295
Epoch: 40	Batch: 56	Training Loss: 3.616670
Epoch: 40	Batch: 64	Training Loss: 3.619149
Epoch: 40	Batch: 72	Training Loss: 3.616174
Epoch: 40	Batch: 80	Training Loss: 3.614899
Epoch: 40	Batch: 88	Training Loss: 3.615886
Epoch: 40	Batch: 96	Training Loss: 3.619943
Epoch: 40	Batch: 104	Training Loss: 3.623511
Epoch: 40	Batch: 8	Validation Loss: 4.131431
Epoch: 41	Batch: 8	Training Loss: 3.521228
Epoch: 41	Batch: 16	Training Loss: 3.590024
Epoch: 41	Batch: 24	Training Loss: 3.593112
Epoch: 41	Batch: 32	Training Loss: 3.609362
Epoch: 41	Batch: 40	Training Loss: 3.631700
Epoch: 41	Batch: 48	Training Loss: 3.641387
Epoch: 41	Batch: 56	Training Loss: 3.633343
Epoch: 41	Batch: 64	Training Loss: 3.654196
Epoch: 41	Batch: 72	Training Loss: 3.635210
Epoch: 41	Batch: 80	Training Loss: 3.635093
Epoch: 41	Batch: 88	Training Loss: 3.621186
Epoch: 41	Batch: 96	Training Loss: 3.622350
Epoch: 41	Batch: 104	Training Loss: 3.616369
Epoch: 41	Batch: 8	Validation Loss: 4.016651
Epoch: 42	Batch: 8	Training Loss: 3.501836
Epoch: 42	Batch: 16	Training Loss: 3.604836
Epoch: 42	Batch: 24	Training Loss: 3.603607
Epoch: 42	Batch: 32	Training Loss: 3.599875
Epoch: 42	Batch: 40	Training Loss: 3.619410
Epoch: 42	Batch: 48	Training Loss: 3.599162
Epoch: 42	Batch: 56	Training Loss: 3.585941
Epoch: 42	Batch: 64	Training Loss: 3.590681
Epoch: 42	Batch: 72	Training Loss: 3.587545
Epoch: 42	Batch: 80	Training Loss: 3.575267

Epoch: 42	Batch: 88	Training Loss: 3.583907
Epoch: 42	Batch: 96	Training Loss: 3.585491
Epoch: 42	Batch: 104	Training Loss: 3.590317
Validation loss decreased (3.852999 --> 3.820007). Saving model ...		
Validation loss decreased (3.820007 --> 3.742852). Saving model ...		
Validation loss decreased (3.742852 --> 3.664562). Saving model ...		
Epoch: 42	Batch: 8	Validation Loss: 3.810106
Epoch: 43	Batch: 8	Training Loss: 3.607287
Epoch: 43	Batch: 16	Training Loss: 3.558954
Epoch: 43	Batch: 24	Training Loss: 3.500495
Epoch: 43	Batch: 32	Training Loss: 3.493044
Epoch: 43	Batch: 40	Training Loss: 3.508948
Epoch: 43	Batch: 48	Training Loss: 3.512225
Epoch: 43	Batch: 56	Training Loss: 3.520880
Epoch: 43	Batch: 64	Training Loss: 3.526600
Epoch: 43	Batch: 72	Training Loss: 3.536625
Epoch: 43	Batch: 80	Training Loss: 3.548856
Epoch: 43	Batch: 88	Training Loss: 3.556978
Epoch: 43	Batch: 96	Training Loss: 3.552421
Epoch: 43	Batch: 104	Training Loss: 3.548932
Epoch: 43	Batch: 8	Validation Loss: 4.051677
Epoch: 44	Batch: 8	Training Loss: 3.550863
Epoch: 44	Batch: 16	Training Loss: 3.578300
Epoch: 44	Batch: 24	Training Loss: 3.552096
Epoch: 44	Batch: 32	Training Loss: 3.528782
Epoch: 44	Batch: 40	Training Loss: 3.543375
Epoch: 44	Batch: 48	Training Loss: 3.545840
Epoch: 44	Batch: 56	Training Loss: 3.549510
Epoch: 44	Batch: 64	Training Loss: 3.556216
Epoch: 44	Batch: 72	Training Loss: 3.553796
Epoch: 44	Batch: 80	Training Loss: 3.557517
Epoch: 44	Batch: 88	Training Loss: 3.558067
Epoch: 44	Batch: 96	Training Loss: 3.557038
Epoch: 44	Batch: 104	Training Loss: 3.551997
Epoch: 44	Batch: 8	Validation Loss: 3.917296
Epoch: 45	Batch: 8	Training Loss: 3.520360
Epoch: 45	Batch: 16	Training Loss: 3.541358
Epoch: 45	Batch: 24	Training Loss: 3.529903
Epoch: 45	Batch: 32	Training Loss: 3.546004
Epoch: 45	Batch: 40	Training Loss: 3.538531
Epoch: 45	Batch: 48	Training Loss: 3.528155
Epoch: 45	Batch: 56	Training Loss: 3.521878
Epoch: 45	Batch: 64	Training Loss: 3.528247
Epoch: 45	Batch: 72	Training Loss: 3.512791
Epoch: 45	Batch: 80	Training Loss: 3.507593
Epoch: 45	Batch: 88	Training Loss: 3.517505
Epoch: 45	Batch: 96	Training Loss: 3.522978
Epoch: 45	Batch: 104	Training Loss: 3.526438

Epoch: 45	Batch: 8	Validation Loss: 3.951923
Epoch: 46	Batch: 8	Training Loss: 3.671772
Epoch: 46	Batch: 16	Training Loss: 3.618694
Epoch: 46	Batch: 24	Training Loss: 3.545596
Epoch: 46	Batch: 32	Training Loss: 3.535644
Epoch: 46	Batch: 40	Training Loss: 3.536922
Epoch: 46	Batch: 48	Training Loss: 3.522468
Epoch: 46	Batch: 56	Training Loss: 3.514943
Epoch: 46	Batch: 64	Training Loss: 3.526025
Epoch: 46	Batch: 72	Training Loss: 3.514052
Epoch: 46	Batch: 80	Training Loss: 3.509518
Epoch: 46	Batch: 88	Training Loss: 3.516112
Epoch: 46	Batch: 96	Training Loss: 3.515162
Epoch: 46	Batch: 104	Training Loss: 3.510474
Epoch: 46	Batch: 8	Validation Loss: 3.962132
Epoch: 47	Batch: 8	Training Loss: 3.474302
Epoch: 47	Batch: 16	Training Loss: 3.535047
Epoch: 47	Batch: 24	Training Loss: 3.542933
Epoch: 47	Batch: 32	Training Loss: 3.518590
Epoch: 47	Batch: 40	Training Loss: 3.505729
Epoch: 47	Batch: 48	Training Loss: 3.491716
Epoch: 47	Batch: 56	Training Loss: 3.494707
Epoch: 47	Batch: 64	Training Loss: 3.474817
Epoch: 47	Batch: 72	Training Loss: 3.477555
Epoch: 47	Batch: 80	Training Loss: 3.464389
Epoch: 47	Batch: 88	Training Loss: 3.470415
Epoch: 47	Batch: 96	Training Loss: 3.481244
Epoch: 47	Batch: 104	Training Loss: 3.474494
Epoch: 47	Batch: 8	Validation Loss: 3.820230
Epoch: 48	Batch: 8	Training Loss: 3.439829
Epoch: 48	Batch: 16	Training Loss: 3.450278
Epoch: 48	Batch: 24	Training Loss: 3.452037
Epoch: 48	Batch: 32	Training Loss: 3.458890
Epoch: 48	Batch: 40	Training Loss: 3.460919
Epoch: 48	Batch: 48	Training Loss: 3.448795
Epoch: 48	Batch: 56	Training Loss: 3.447933
Epoch: 48	Batch: 64	Training Loss: 3.450011
Epoch: 48	Batch: 72	Training Loss: 3.455512
Epoch: 48	Batch: 80	Training Loss: 3.461026
Epoch: 48	Batch: 88	Training Loss: 3.451031
Epoch: 48	Batch: 96	Training Loss: 3.458155
Epoch: 48	Batch: 104	Training Loss: 3.452154
Epoch: 48	Batch: 8	Validation Loss: 4.006319
Epoch: 49	Batch: 8	Training Loss: 3.489316
Epoch: 49	Batch: 16	Training Loss: 3.473076
Epoch: 49	Batch: 24	Training Loss: 3.474347
Epoch: 49	Batch: 32	Training Loss: 3.483469
Epoch: 49	Batch: 40	Training Loss: 3.472261

Epoch: 49	Batch: 48	Training Loss: 3.459555
Epoch: 49	Batch: 56	Training Loss: 3.454224
Epoch: 49	Batch: 64	Training Loss: 3.440406
Epoch: 49	Batch: 72	Training Loss: 3.445911
Epoch: 49	Batch: 80	Training Loss: 3.446751
Epoch: 49	Batch: 88	Training Loss: 3.442994
Epoch: 49	Batch: 96	Training Loss: 3.438337
Epoch: 49	Batch: 104	Training Loss: 3.442112
Validation loss decreased (3.664562 --> 3.662063). Saving model ...		
Validation loss decreased (3.662063 --> 3.661447). Saving model ...		
Validation loss decreased (3.661447 --> 3.634730). Saving model ...		
Epoch: 49	Batch: 8	Validation Loss: 3.779524
Epoch: 50	Batch: 8	Training Loss: 3.472746
Epoch: 50	Batch: 16	Training Loss: 3.439781
Epoch: 50	Batch: 24	Training Loss: 3.444173
Epoch: 50	Batch: 32	Training Loss: 3.463939
Epoch: 50	Batch: 40	Training Loss: 3.434967
Epoch: 50	Batch: 48	Training Loss: 3.418890
Epoch: 50	Batch: 56	Training Loss: 3.425733
Epoch: 50	Batch: 64	Training Loss: 3.418543
Epoch: 50	Batch: 72	Training Loss: 3.414479
Epoch: 50	Batch: 80	Training Loss: 3.407320
Epoch: 50	Batch: 88	Training Loss: 3.421701
Epoch: 50	Batch: 96	Training Loss: 3.424888
Epoch: 50	Batch: 104	Training Loss: 3.417190
Epoch: 50	Batch: 8	Validation Loss: 3.890421
Epoch: 51	Batch: 8	Training Loss: 3.375601
Epoch: 51	Batch: 16	Training Loss: 3.457289
Epoch: 51	Batch: 24	Training Loss: 3.427478
Epoch: 51	Batch: 32	Training Loss: 3.415305
Epoch: 51	Batch: 40	Training Loss: 3.404811
Epoch: 51	Batch: 48	Training Loss: 3.433170
Epoch: 51	Batch: 56	Training Loss: 3.415406
Epoch: 51	Batch: 64	Training Loss: 3.414666
Epoch: 51	Batch: 72	Training Loss: 3.417574
Epoch: 51	Batch: 80	Training Loss: 3.407825
Epoch: 51	Batch: 88	Training Loss: 3.408336
Epoch: 51	Batch: 96	Training Loss: 3.413469
Epoch: 51	Batch: 104	Training Loss: 3.414849
Validation loss decreased (3.634730 --> 3.632473). Saving model ...		
Epoch: 51	Batch: 8	Validation Loss: 4.119710
Epoch: 52	Batch: 8	Training Loss: 3.427764
Epoch: 52	Batch: 16	Training Loss: 3.439160
Epoch: 52	Batch: 24	Training Loss: 3.409808
Epoch: 52	Batch: 32	Training Loss: 3.414318
Epoch: 52	Batch: 40	Training Loss: 3.398144
Epoch: 52	Batch: 48	Training Loss: 3.410171
Epoch: 52	Batch: 56	Training Loss: 3.400274

Epoch: 52	Batch: 64	Training Loss: 3.397197
Epoch: 52	Batch: 72	Training Loss: 3.396511
Epoch: 52	Batch: 80	Training Loss: 3.389667
Epoch: 52	Batch: 88	Training Loss: 3.390727
Epoch: 52	Batch: 96	Training Loss: 3.383853
Epoch: 52	Batch: 104	Training Loss: 3.379719
Epoch: 52	Batch: 8	Validation Loss: 3.790082
Epoch: 53	Batch: 8	Training Loss: 3.380495
Epoch: 53	Batch: 16	Training Loss: 3.296735
Epoch: 53	Batch: 24	Training Loss: 3.324998
Epoch: 53	Batch: 32	Training Loss: 3.331621
Epoch: 53	Batch: 40	Training Loss: 3.349017
Epoch: 53	Batch: 48	Training Loss: 3.345762
Epoch: 53	Batch: 56	Training Loss: 3.335889
Epoch: 53	Batch: 64	Training Loss: 3.341421
Epoch: 53	Batch: 72	Training Loss: 3.328129
Epoch: 53	Batch: 80	Training Loss: 3.333595
Epoch: 53	Batch: 88	Training Loss: 3.328733
Epoch: 53	Batch: 96	Training Loss: 3.323114
Epoch: 53	Batch: 104	Training Loss: 3.330420
Validation loss decreased (3.632473 --> 3.622995). Saving model ...		
Epoch: 53	Batch: 8	Validation Loss: 3.690286
Epoch: 54	Batch: 8	Training Loss: 3.409206
Epoch: 54	Batch: 16	Training Loss: 3.335424
Epoch: 54	Batch: 24	Training Loss: 3.341788
Epoch: 54	Batch: 32	Training Loss: 3.315019
Epoch: 54	Batch: 40	Training Loss: 3.327127
Epoch: 54	Batch: 48	Training Loss: 3.337098
Epoch: 54	Batch: 56	Training Loss: 3.327721
Epoch: 54	Batch: 64	Training Loss: 3.323296
Epoch: 54	Batch: 72	Training Loss: 3.318587
Epoch: 54	Batch: 80	Training Loss: 3.328288
Epoch: 54	Batch: 88	Training Loss: 3.328589
Epoch: 54	Batch: 96	Training Loss: 3.339075
Epoch: 54	Batch: 104	Training Loss: 3.344267
Validation loss decreased (3.622995 --> 3.544091). Saving model ...		
Epoch: 54	Batch: 8	Validation Loss: 3.816114
Epoch: 55	Batch: 8	Training Loss: 3.321927
Epoch: 55	Batch: 16	Training Loss: 3.339829
Epoch: 55	Batch: 24	Training Loss: 3.303926
Epoch: 55	Batch: 32	Training Loss: 3.317523
Epoch: 55	Batch: 40	Training Loss: 3.321011
Epoch: 55	Batch: 48	Training Loss: 3.325920
Epoch: 55	Batch: 56	Training Loss: 3.340044
Epoch: 55	Batch: 64	Training Loss: 3.331610
Epoch: 55	Batch: 72	Training Loss: 3.325409
Epoch: 55	Batch: 80	Training Loss: 3.311527
Epoch: 55	Batch: 88	Training Loss: 3.313266

Epoch: 55	Batch: 96	Training Loss: 3.315692
Epoch: 55	Batch: 104	Training Loss: 3.316365
Epoch: 55	Batch: 8	Validation Loss: 3.845119
Epoch: 56	Batch: 8	Training Loss: 3.303870
Epoch: 56	Batch: 16	Training Loss: 3.271700
Epoch: 56	Batch: 24	Training Loss: 3.315419
Epoch: 56	Batch: 32	Training Loss: 3.317233
Epoch: 56	Batch: 40	Training Loss: 3.315530
Epoch: 56	Batch: 48	Training Loss: 3.318702
Epoch: 56	Batch: 56	Training Loss: 3.304477
Epoch: 56	Batch: 64	Training Loss: 3.297183
Epoch: 56	Batch: 72	Training Loss: 3.283211
Epoch: 56	Batch: 80	Training Loss: 3.298607
Epoch: 56	Batch: 88	Training Loss: 3.310634
Epoch: 56	Batch: 96	Training Loss: 3.312422
Epoch: 56	Batch: 104	Training Loss: 3.312122
Epoch: 56	Batch: 8	Validation Loss: 3.961900
Epoch: 57	Batch: 8	Training Loss: 3.257571
Epoch: 57	Batch: 16	Training Loss: 3.302606
Epoch: 57	Batch: 24	Training Loss: 3.264338
Epoch: 57	Batch: 32	Training Loss: 3.281494
Epoch: 57	Batch: 40	Training Loss: 3.269907
Epoch: 57	Batch: 48	Training Loss: 3.278461
Epoch: 57	Batch: 56	Training Loss: 3.290658
Epoch: 57	Batch: 64	Training Loss: 3.297187
Epoch: 57	Batch: 72	Training Loss: 3.289115
Epoch: 57	Batch: 80	Training Loss: 3.292513
Epoch: 57	Batch: 88	Training Loss: 3.302468
Epoch: 57	Batch: 96	Training Loss: 3.312425
Epoch: 57	Batch: 104	Training Loss: 3.306023
Epoch: 57	Batch: 8	Validation Loss: 3.684831
Epoch: 58	Batch: 8	Training Loss: 3.267272
Epoch: 58	Batch: 16	Training Loss: 3.319350
Epoch: 58	Batch: 24	Training Loss: 3.315586
Epoch: 58	Batch: 32	Training Loss: 3.298454
Epoch: 58	Batch: 40	Training Loss: 3.292540
Epoch: 58	Batch: 48	Training Loss: 3.309815
Epoch: 58	Batch: 56	Training Loss: 3.317781
Epoch: 58	Batch: 64	Training Loss: 3.312335
Epoch: 58	Batch: 72	Training Loss: 3.307841
Epoch: 58	Batch: 80	Training Loss: 3.308441
Epoch: 58	Batch: 88	Training Loss: 3.308388
Epoch: 58	Batch: 96	Training Loss: 3.299799
Epoch: 58	Batch: 104	Training Loss: 3.304079
Epoch: 58	Batch: 8	Validation Loss: 3.843679
Epoch: 59	Batch: 8	Training Loss: 3.120282
Epoch: 59	Batch: 16	Training Loss: 3.129648
Epoch: 59	Batch: 24	Training Loss: 3.155001

Epoch: 59	Batch: 32	Training Loss: 3.151587
Epoch: 59	Batch: 40	Training Loss: 3.169172
Epoch: 59	Batch: 48	Training Loss: 3.187181
Epoch: 59	Batch: 56	Training Loss: 3.193321
Epoch: 59	Batch: 64	Training Loss: 3.203193
Epoch: 59	Batch: 72	Training Loss: 3.201726
Epoch: 59	Batch: 80	Training Loss: 3.214143
Epoch: 59	Batch: 88	Training Loss: 3.217450
Epoch: 59	Batch: 96	Training Loss: 3.224256
Epoch: 59	Batch: 104	Training Loss: 3.232213
Validation loss decreased (3.544091 --> 3.491889). Saving model ...		
Epoch: 59	Batch: 8	Validation Loss: 3.836005
Epoch: 60	Batch: 8	Training Loss: 3.104386
Epoch: 60	Batch: 16	Training Loss: 3.141695
Epoch: 60	Batch: 24	Training Loss: 3.163954
Epoch: 60	Batch: 32	Training Loss: 3.201118
Epoch: 60	Batch: 40	Training Loss: 3.188636
Epoch: 60	Batch: 48	Training Loss: 3.213060
Epoch: 60	Batch: 56	Training Loss: 3.225197
Epoch: 60	Batch: 64	Training Loss: 3.243590
Epoch: 60	Batch: 72	Training Loss: 3.244733
Epoch: 60	Batch: 80	Training Loss: 3.246063
Epoch: 60	Batch: 88	Training Loss: 3.253175
Epoch: 60	Batch: 96	Training Loss: 3.250437
Epoch: 60	Batch: 104	Training Loss: 3.247987
Epoch: 60	Batch: 8	Validation Loss: 3.834765
Epoch: 61	Batch: 8	Training Loss: 3.203528
Epoch: 61	Batch: 16	Training Loss: 3.218679
Epoch: 61	Batch: 24	Training Loss: 3.210175
Epoch: 61	Batch: 32	Training Loss: 3.194060
Epoch: 61	Batch: 40	Training Loss: 3.209374
Epoch: 61	Batch: 48	Training Loss: 3.213675
Epoch: 61	Batch: 56	Training Loss: 3.207292
Epoch: 61	Batch: 64	Training Loss: 3.187554
Epoch: 61	Batch: 72	Training Loss: 3.183715
Epoch: 61	Batch: 80	Training Loss: 3.192463
Epoch: 61	Batch: 88	Training Loss: 3.192728
Epoch: 61	Batch: 96	Training Loss: 3.194553
Epoch: 61	Batch: 104	Training Loss: 3.197103
Epoch: 61	Batch: 8	Validation Loss: 3.920379
Epoch: 62	Batch: 8	Training Loss: 3.131145
Epoch: 62	Batch: 16	Training Loss: 3.168837
Epoch: 62	Batch: 24	Training Loss: 3.165307
Epoch: 62	Batch: 32	Training Loss: 3.194575
Epoch: 62	Batch: 40	Training Loss: 3.187465
Epoch: 62	Batch: 48	Training Loss: 3.171309
Epoch: 62	Batch: 56	Training Loss: 3.169615
Epoch: 62	Batch: 64	Training Loss: 3.164732

Epoch: 62	Batch: 72	Training Loss: 3.163241
Epoch: 62	Batch: 80	Training Loss: 3.175229
Epoch: 62	Batch: 88	Training Loss: 3.177676
Epoch: 62	Batch: 96	Training Loss: 3.182260
Epoch: 62	Batch: 104	Training Loss: 3.186320
Epoch: 62	Batch: 8	Validation Loss: 4.030638
Epoch: 63	Batch: 8	Training Loss: 3.159479
Epoch: 63	Batch: 16	Training Loss: 3.226304
Epoch: 63	Batch: 24	Training Loss: 3.226128
Epoch: 63	Batch: 32	Training Loss: 3.192253
Epoch: 63	Batch: 40	Training Loss: 3.146023
Epoch: 63	Batch: 48	Training Loss: 3.166061
Epoch: 63	Batch: 56	Training Loss: 3.155512
Epoch: 63	Batch: 64	Training Loss: 3.160964
Epoch: 63	Batch: 72	Training Loss: 3.157032
Epoch: 63	Batch: 80	Training Loss: 3.160398
Epoch: 63	Batch: 88	Training Loss: 3.167213
Epoch: 63	Batch: 96	Training Loss: 3.163365
Epoch: 63	Batch: 104	Training Loss: 3.174526
Epoch: 63	Batch: 8	Validation Loss: 4.007175
Epoch: 64	Batch: 8	Training Loss: 3.157003
Epoch: 64	Batch: 16	Training Loss: 3.211111
Epoch: 64	Batch: 24	Training Loss: 3.205399
Epoch: 64	Batch: 32	Training Loss: 3.197787
Epoch: 64	Batch: 40	Training Loss: 3.193900
Epoch: 64	Batch: 48	Training Loss: 3.184242
Epoch: 64	Batch: 56	Training Loss: 3.185831
Epoch: 64	Batch: 64	Training Loss: 3.175651
Epoch: 64	Batch: 72	Training Loss: 3.181048
Epoch: 64	Batch: 80	Training Loss: 3.178691
Epoch: 64	Batch: 88	Training Loss: 3.181072
Epoch: 64	Batch: 96	Training Loss: 3.181022
Epoch: 64	Batch: 104	Training Loss: 3.172537
Validation loss decreased (3.491889 --> 3.474332). Saving model ...		
Epoch: 64	Batch: 8	Validation Loss: 3.666077
Epoch: 65	Batch: 8	Training Loss: 3.013524
Epoch: 65	Batch: 16	Training Loss: 3.116772
Epoch: 65	Batch: 24	Training Loss: 3.083451
Epoch: 65	Batch: 32	Training Loss: 3.085065
Epoch: 65	Batch: 40	Training Loss: 3.095364
Epoch: 65	Batch: 48	Training Loss: 3.120743
Epoch: 65	Batch: 56	Training Loss: 3.132827
Epoch: 65	Batch: 64	Training Loss: 3.135868
Epoch: 65	Batch: 72	Training Loss: 3.130980
Epoch: 65	Batch: 80	Training Loss: 3.147973
Epoch: 65	Batch: 88	Training Loss: 3.138690
Epoch: 65	Batch: 96	Training Loss: 3.140388
Epoch: 65	Batch: 104	Training Loss: 3.130298

Epoch: 65	Batch: 8	Validation Loss: 3.846780
Epoch: 66	Batch: 8	Training Loss: 3.111078
Epoch: 66	Batch: 16	Training Loss: 3.114412
Epoch: 66	Batch: 24	Training Loss: 3.133663
Epoch: 66	Batch: 32	Training Loss: 3.104189
Epoch: 66	Batch: 40	Training Loss: 3.084793
Epoch: 66	Batch: 48	Training Loss: 3.077518
Epoch: 66	Batch: 56	Training Loss: 3.087894
Epoch: 66	Batch: 64	Training Loss: 3.099971
Epoch: 66	Batch: 72	Training Loss: 3.102946
Epoch: 66	Batch: 80	Training Loss: 3.099938
Epoch: 66	Batch: 88	Training Loss: 3.097789
Epoch: 66	Batch: 96	Training Loss: 3.099748
Epoch: 66	Batch: 104	Training Loss: 3.097083
Epoch: 66	Batch: 8	Validation Loss: 3.861316
Epoch: 67	Batch: 8	Training Loss: 3.016212
Epoch: 67	Batch: 16	Training Loss: 3.095229
Epoch: 67	Batch: 24	Training Loss: 3.124879
Epoch: 67	Batch: 32	Training Loss: 3.118137
Epoch: 67	Batch: 40	Training Loss: 3.109143
Epoch: 67	Batch: 48	Training Loss: 3.105655
Epoch: 67	Batch: 56	Training Loss: 3.112195
Epoch: 67	Batch: 64	Training Loss: 3.119435
Epoch: 67	Batch: 72	Training Loss: 3.127636
Epoch: 67	Batch: 80	Training Loss: 3.123437
Epoch: 67	Batch: 88	Training Loss: 3.113087
Epoch: 67	Batch: 96	Training Loss: 3.109606
Epoch: 67	Batch: 104	Training Loss: 3.117323
Epoch: 67	Batch: 8	Validation Loss: 3.658059
Epoch: 68	Batch: 8	Training Loss: 3.165295
Epoch: 68	Batch: 16	Training Loss: 3.078204
Epoch: 68	Batch: 24	Training Loss: 3.104284
Epoch: 68	Batch: 32	Training Loss: 3.098454
Epoch: 68	Batch: 40	Training Loss: 3.097895
Epoch: 68	Batch: 48	Training Loss: 3.100591
Epoch: 68	Batch: 56	Training Loss: 3.103373
Epoch: 68	Batch: 64	Training Loss: 3.115694
Epoch: 68	Batch: 72	Training Loss: 3.120045
Epoch: 68	Batch: 80	Training Loss: 3.107651
Epoch: 68	Batch: 88	Training Loss: 3.107408
Epoch: 68	Batch: 96	Training Loss: 3.116116
Epoch: 68	Batch: 104	Training Loss: 3.115839
Epoch: 68	Batch: 8	Validation Loss: 3.658625
Epoch: 69	Batch: 8	Training Loss: 3.143062
Epoch: 69	Batch: 16	Training Loss: 3.110222
Epoch: 69	Batch: 24	Training Loss: 3.080648
Epoch: 69	Batch: 32	Training Loss: 3.057550
Epoch: 69	Batch: 40	Training Loss: 3.051056

Epoch: 69	Batch: 48	Training Loss: 3.079161
Epoch: 69	Batch: 56	Training Loss: 3.066260
Epoch: 69	Batch: 64	Training Loss: 3.069948
Epoch: 69	Batch: 72	Training Loss: 3.076152
Epoch: 69	Batch: 80	Training Loss: 3.066847
Epoch: 69	Batch: 88	Training Loss: 3.072935
Epoch: 69	Batch: 96	Training Loss: 3.066960
Epoch: 69	Batch: 104	Training Loss: 3.071358
Epoch: 69	Batch: 8	Validation Loss: 3.635171
Epoch: 70	Batch: 8	Training Loss: 3.048151
Epoch: 70	Batch: 16	Training Loss: 3.059593
Epoch: 70	Batch: 24	Training Loss: 3.099464
Epoch: 70	Batch: 32	Training Loss: 3.108723
Epoch: 70	Batch: 40	Training Loss: 3.088293
Epoch: 70	Batch: 48	Training Loss: 3.068630
Epoch: 70	Batch: 56	Training Loss: 3.088613
Epoch: 70	Batch: 64	Training Loss: 3.085744
Epoch: 70	Batch: 72	Training Loss: 3.093704
Epoch: 70	Batch: 80	Training Loss: 3.088321
Epoch: 70	Batch: 88	Training Loss: 3.079679
Epoch: 70	Batch: 96	Training Loss: 3.079744
Epoch: 70	Batch: 104	Training Loss: 3.079937
Epoch: 70	Batch: 8	Validation Loss: 3.582577
Epoch: 71	Batch: 8	Training Loss: 3.153661
Epoch: 71	Batch: 16	Training Loss: 3.118615
Epoch: 71	Batch: 24	Training Loss: 3.080418
Epoch: 71	Batch: 32	Training Loss: 3.085645
Epoch: 71	Batch: 40	Training Loss: 3.057372
Epoch: 71	Batch: 48	Training Loss: 3.082988
Epoch: 71	Batch: 56	Training Loss: 3.082202
Epoch: 71	Batch: 64	Training Loss: 3.096480
Epoch: 71	Batch: 72	Training Loss: 3.082887
Epoch: 71	Batch: 80	Training Loss: 3.077959
Epoch: 71	Batch: 88	Training Loss: 3.078482
Epoch: 71	Batch: 96	Training Loss: 3.075523
Epoch: 71	Batch: 104	Training Loss: 3.070536
Epoch: 71	Batch: 8	Validation Loss: 3.634604
Epoch: 72	Batch: 8	Training Loss: 2.979038
Epoch: 72	Batch: 16	Training Loss: 3.029528
Epoch: 72	Batch: 24	Training Loss: 3.047419
Epoch: 72	Batch: 32	Training Loss: 3.062127
Epoch: 72	Batch: 40	Training Loss: 3.058715
Epoch: 72	Batch: 48	Training Loss: 3.063158
Epoch: 72	Batch: 56	Training Loss: 3.061428
Epoch: 72	Batch: 64	Training Loss: 3.066969
Epoch: 72	Batch: 72	Training Loss: 3.064760
Epoch: 72	Batch: 80	Training Loss: 3.062265
Epoch: 72	Batch: 88	Training Loss: 3.061891

Epoch: 72	Batch: 96	Training Loss: 3.059702
Epoch: 72	Batch: 104	Training Loss: 3.058568
Epoch: 72	Batch: 8	Validation Loss: 3.871324
Epoch: 73	Batch: 8	Training Loss: 3.144484
Epoch: 73	Batch: 16	Training Loss: 3.043843
Epoch: 73	Batch: 24	Training Loss: 3.012925
Epoch: 73	Batch: 32	Training Loss: 3.021205
Epoch: 73	Batch: 40	Training Loss: 2.999390
Epoch: 73	Batch: 48	Training Loss: 3.034668
Epoch: 73	Batch: 56	Training Loss: 3.039553
Epoch: 73	Batch: 64	Training Loss: 3.034772
Epoch: 73	Batch: 72	Training Loss: 3.050042
Epoch: 73	Batch: 80	Training Loss: 3.046816
Epoch: 73	Batch: 88	Training Loss: 3.044956
Epoch: 73	Batch: 96	Training Loss: 3.027593
Epoch: 73	Batch: 104	Training Loss: 3.028100
Epoch: 73	Batch: 8	Validation Loss: 3.628634
Epoch: 74	Batch: 8	Training Loss: 3.012694
Epoch: 74	Batch: 16	Training Loss: 3.031309
Epoch: 74	Batch: 24	Training Loss: 3.049982
Epoch: 74	Batch: 32	Training Loss: 3.009701
Epoch: 74	Batch: 40	Training Loss: 3.002243
Epoch: 74	Batch: 48	Training Loss: 2.991927
Epoch: 74	Batch: 56	Training Loss: 2.987724
Epoch: 74	Batch: 64	Training Loss: 2.993480
Epoch: 74	Batch: 72	Training Loss: 2.992035
Epoch: 74	Batch: 80	Training Loss: 3.003316
Epoch: 74	Batch: 88	Training Loss: 3.003649
Epoch: 74	Batch: 96	Training Loss: 3.008543
Epoch: 74	Batch: 104	Training Loss: 3.017559
Epoch: 74	Batch: 8	Validation Loss: 3.640407
Epoch: 75	Batch: 8	Training Loss: 2.891105
Epoch: 75	Batch: 16	Training Loss: 2.954149
Epoch: 75	Batch: 24	Training Loss: 3.014764
Epoch: 75	Batch: 32	Training Loss: 3.000148
Epoch: 75	Batch: 40	Training Loss: 3.039491
Epoch: 75	Batch: 48	Training Loss: 3.025683
Epoch: 75	Batch: 56	Training Loss: 3.016325
Epoch: 75	Batch: 64	Training Loss: 3.024775
Epoch: 75	Batch: 72	Training Loss: 3.023419
Epoch: 75	Batch: 80	Training Loss: 3.019303
Epoch: 75	Batch: 88	Training Loss: 3.024870
Epoch: 75	Batch: 96	Training Loss: 3.031342
Epoch: 75	Batch: 104	Training Loss: 3.028040
Epoch: 75	Batch: 8	Validation Loss: 3.674393
Epoch: 76	Batch: 8	Training Loss: 3.054081
Epoch: 76	Batch: 16	Training Loss: 2.938780
Epoch: 76	Batch: 24	Training Loss: 2.975292

Epoch: 76	Batch: 32	Training Loss: 2.965740
Epoch: 76	Batch: 40	Training Loss: 2.964418
Epoch: 76	Batch: 48	Training Loss: 2.941455
Epoch: 76	Batch: 56	Training Loss: 2.955802
Epoch: 76	Batch: 64	Training Loss: 2.970704
Epoch: 76	Batch: 72	Training Loss: 2.954649
Epoch: 76	Batch: 80	Training Loss: 2.946644
Epoch: 76	Batch: 88	Training Loss: 2.946783
Epoch: 76	Batch: 96	Training Loss: 2.959437
Epoch: 76	Batch: 104	Training Loss: 2.959383
Epoch: 76	Batch: 8	Validation Loss: 3.671800
Epoch: 77	Batch: 8	Training Loss: 2.982028
Epoch: 77	Batch: 16	Training Loss: 2.922458
Epoch: 77	Batch: 24	Training Loss: 2.953825
Epoch: 77	Batch: 32	Training Loss: 2.955415
Epoch: 77	Batch: 40	Training Loss: 2.966220
Epoch: 77	Batch: 48	Training Loss: 2.933119
Epoch: 77	Batch: 56	Training Loss: 2.939503
Epoch: 77	Batch: 64	Training Loss: 2.954704
Epoch: 77	Batch: 72	Training Loss: 2.964694
Epoch: 77	Batch: 80	Training Loss: 2.959704
Epoch: 77	Batch: 88	Training Loss: 2.944042
Epoch: 77	Batch: 96	Training Loss: 2.938987
Epoch: 77	Batch: 104	Training Loss: 2.949833
Epoch: 77	Batch: 8	Validation Loss: 3.633309
Epoch: 78	Batch: 8	Training Loss: 2.989327
Epoch: 78	Batch: 16	Training Loss: 2.941486
Epoch: 78	Batch: 24	Training Loss: 2.949729
Epoch: 78	Batch: 32	Training Loss: 2.946249
Epoch: 78	Batch: 40	Training Loss: 2.925371
Epoch: 78	Batch: 48	Training Loss: 2.933274
Epoch: 78	Batch: 56	Training Loss: 2.931310
Epoch: 78	Batch: 64	Training Loss: 2.935925
Epoch: 78	Batch: 72	Training Loss: 2.948479
Epoch: 78	Batch: 80	Training Loss: 2.953581
Epoch: 78	Batch: 88	Training Loss: 2.960774
Epoch: 78	Batch: 96	Training Loss: 2.957115
Epoch: 78	Batch: 104	Training Loss: 2.957993
Epoch: 78	Batch: 8	Validation Loss: 3.820217
Epoch: 79	Batch: 8	Training Loss: 2.990614
Epoch: 79	Batch: 16	Training Loss: 3.044400
Epoch: 79	Batch: 24	Training Loss: 2.969449
Epoch: 79	Batch: 32	Training Loss: 3.000227
Epoch: 79	Batch: 40	Training Loss: 2.965564
Epoch: 79	Batch: 48	Training Loss: 2.957823
Epoch: 79	Batch: 56	Training Loss: 2.963915
Epoch: 79	Batch: 64	Training Loss: 2.963068
Epoch: 79	Batch: 72	Training Loss: 2.959830

Epoch: 79	Batch: 80	Training Loss: 2.962685
Epoch: 79	Batch: 88	Training Loss: 2.965570
Epoch: 79	Batch: 96	Training Loss: 2.964722
Epoch: 79	Batch: 104	Training Loss: 2.965654
Epoch: 79	Batch: 8	Validation Loss: 3.706014
Epoch: 80	Batch: 8	Training Loss: 3.006773
Epoch: 80	Batch: 16	Training Loss: 2.965561
Epoch: 80	Batch: 24	Training Loss: 2.963495
Epoch: 80	Batch: 32	Training Loss: 2.938648
Epoch: 80	Batch: 40	Training Loss: 2.926570
Epoch: 80	Batch: 48	Training Loss: 2.915927
Epoch: 80	Batch: 56	Training Loss: 2.911213
Epoch: 80	Batch: 64	Training Loss: 2.933989
Epoch: 80	Batch: 72	Training Loss: 2.928097
Epoch: 80	Batch: 80	Training Loss: 2.915147
Epoch: 80	Batch: 88	Training Loss: 2.916257
Epoch: 80	Batch: 96	Training Loss: 2.918419
Epoch: 80	Batch: 104	Training Loss: 2.917778
Validation loss decreased (3.474332 --> 3.472241). Saving model ...		
Epoch: 80	Batch: 8	Validation Loss: 3.777790
Epoch: 81	Batch: 8	Training Loss: 3.061189
Epoch: 81	Batch: 16	Training Loss: 2.963220
Epoch: 81	Batch: 24	Training Loss: 2.931504
Epoch: 81	Batch: 32	Training Loss: 2.921775
Epoch: 81	Batch: 40	Training Loss: 2.898912
Epoch: 81	Batch: 48	Training Loss: 2.877989
Epoch: 81	Batch: 56	Training Loss: 2.892551
Epoch: 81	Batch: 64	Training Loss: 2.898098
Epoch: 81	Batch: 72	Training Loss: 2.910366
Epoch: 81	Batch: 80	Training Loss: 2.900993
Epoch: 81	Batch: 88	Training Loss: 2.898607
Epoch: 81	Batch: 96	Training Loss: 2.907455
Epoch: 81	Batch: 104	Training Loss: 2.916133
Validation loss decreased (3.472241 --> 3.411304). Saving model ...		
Epoch: 81	Batch: 8	Validation Loss: 3.773191
Epoch: 82	Batch: 8	Training Loss: 2.826399
Epoch: 82	Batch: 16	Training Loss: 2.842500
Epoch: 82	Batch: 24	Training Loss: 2.852025
Epoch: 82	Batch: 32	Training Loss: 2.841558
Epoch: 82	Batch: 40	Training Loss: 2.842560
Epoch: 82	Batch: 48	Training Loss: 2.860301
Epoch: 82	Batch: 56	Training Loss: 2.849246
Epoch: 82	Batch: 64	Training Loss: 2.860094
Epoch: 82	Batch: 72	Training Loss: 2.864918
Epoch: 82	Batch: 80	Training Loss: 2.854735
Epoch: 82	Batch: 88	Training Loss: 2.861319
Epoch: 82	Batch: 96	Training Loss: 2.871937
Epoch: 82	Batch: 104	Training Loss: 2.874182

Epoch: 82	Batch: 8	Validation Loss: 3.684075
Epoch: 83	Batch: 8	Training Loss: 2.800114
Epoch: 83	Batch: 16	Training Loss: 2.816131
Epoch: 83	Batch: 24	Training Loss: 2.857943
Epoch: 83	Batch: 32	Training Loss: 2.853509
Epoch: 83	Batch: 40	Training Loss: 2.869317
Epoch: 83	Batch: 48	Training Loss: 2.891206
Epoch: 83	Batch: 56	Training Loss: 2.897976
Epoch: 83	Batch: 64	Training Loss: 2.894206
Epoch: 83	Batch: 72	Training Loss: 2.888700
Epoch: 83	Batch: 80	Training Loss: 2.882461
Epoch: 83	Batch: 88	Training Loss: 2.893391
Epoch: 83	Batch: 96	Training Loss: 2.884703
Epoch: 83	Batch: 104	Training Loss: 2.886393
Validation loss decreased (3.411304 --> 3.181310). Saving model ...		
Epoch: 83	Batch: 8	Validation Loss: 3.746846
Epoch: 84	Batch: 8	Training Loss: 2.905344
Epoch: 84	Batch: 16	Training Loss: 2.938833
Epoch: 84	Batch: 24	Training Loss: 2.909964
Epoch: 84	Batch: 32	Training Loss: 2.896530
Epoch: 84	Batch: 40	Training Loss: 2.904703
Epoch: 84	Batch: 48	Training Loss: 2.895450
Epoch: 84	Batch: 56	Training Loss: 2.884685
Epoch: 84	Batch: 64	Training Loss: 2.887645
Epoch: 84	Batch: 72	Training Loss: 2.878604
Epoch: 84	Batch: 80	Training Loss: 2.877522
Epoch: 84	Batch: 88	Training Loss: 2.881271
Epoch: 84	Batch: 96	Training Loss: 2.882052
Epoch: 84	Batch: 104	Training Loss: 2.879869
Epoch: 84	Batch: 8	Validation Loss: 3.654876
Epoch: 85	Batch: 8	Training Loss: 2.773849
Epoch: 85	Batch: 16	Training Loss: 2.846406
Epoch: 85	Batch: 24	Training Loss: 2.880906
Epoch: 85	Batch: 32	Training Loss: 2.861892
Epoch: 85	Batch: 40	Training Loss: 2.869810
Epoch: 85	Batch: 48	Training Loss: 2.866945
Epoch: 85	Batch: 56	Training Loss: 2.856598
Epoch: 85	Batch: 64	Training Loss: 2.874120
Epoch: 85	Batch: 72	Training Loss: 2.884882
Epoch: 85	Batch: 80	Training Loss: 2.881074
Epoch: 85	Batch: 88	Training Loss: 2.877464
Epoch: 85	Batch: 96	Training Loss: 2.868620
Epoch: 85	Batch: 104	Training Loss: 2.864324
Epoch: 85	Batch: 8	Validation Loss: 3.634012
Epoch: 86	Batch: 8	Training Loss: 2.821005
Epoch: 86	Batch: 16	Training Loss: 2.808497
Epoch: 86	Batch: 24	Training Loss: 2.827445
Epoch: 86	Batch: 32	Training Loss: 2.843187

Epoch: 86	Batch: 40	Training Loss: 2.816725
Epoch: 86	Batch: 48	Training Loss: 2.821687
Epoch: 86	Batch: 56	Training Loss: 2.818307
Epoch: 86	Batch: 64	Training Loss: 2.821612
Epoch: 86	Batch: 72	Training Loss: 2.810220
Epoch: 86	Batch: 80	Training Loss: 2.796315
Epoch: 86	Batch: 88	Training Loss: 2.806474
Epoch: 86	Batch: 96	Training Loss: 2.812170
Epoch: 86	Batch: 104	Training Loss: 2.815538
Epoch: 86	Batch: 8	Validation Loss: 3.498751
Epoch: 87	Batch: 8	Training Loss: 2.771608
Epoch: 87	Batch: 16	Training Loss: 2.757217
Epoch: 87	Batch: 24	Training Loss: 2.777431
Epoch: 87	Batch: 32	Training Loss: 2.789461
Epoch: 87	Batch: 40	Training Loss: 2.794753
Epoch: 87	Batch: 48	Training Loss: 2.786325
Epoch: 87	Batch: 56	Training Loss: 2.791941
Epoch: 87	Batch: 64	Training Loss: 2.790050
Epoch: 87	Batch: 72	Training Loss: 2.792142
Epoch: 87	Batch: 80	Training Loss: 2.806416
Epoch: 87	Batch: 88	Training Loss: 2.806133
Epoch: 87	Batch: 96	Training Loss: 2.810993
Epoch: 87	Batch: 104	Training Loss: 2.812778
Epoch: 87	Batch: 8	Validation Loss: 3.587577
Epoch: 88	Batch: 8	Training Loss: 2.678478
Epoch: 88	Batch: 16	Training Loss: 2.696443
Epoch: 88	Batch: 24	Training Loss: 2.739215
Epoch: 88	Batch: 32	Training Loss: 2.778592
Epoch: 88	Batch: 40	Training Loss: 2.772485
Epoch: 88	Batch: 48	Training Loss: 2.791800
Epoch: 88	Batch: 56	Training Loss: 2.797437
Epoch: 88	Batch: 64	Training Loss: 2.815084
Epoch: 88	Batch: 72	Training Loss: 2.808384
Epoch: 88	Batch: 80	Training Loss: 2.819070
Epoch: 88	Batch: 88	Training Loss: 2.828051
Epoch: 88	Batch: 96	Training Loss: 2.824370
Epoch: 88	Batch: 104	Training Loss: 2.820176
Epoch: 88	Batch: 8	Validation Loss: 3.551600
Epoch: 89	Batch: 8	Training Loss: 2.849056
Epoch: 89	Batch: 16	Training Loss: 2.870520
Epoch: 89	Batch: 24	Training Loss: 2.801221
Epoch: 89	Batch: 32	Training Loss: 2.801505
Epoch: 89	Batch: 40	Training Loss: 2.788284
Epoch: 89	Batch: 48	Training Loss: 2.789310
Epoch: 89	Batch: 56	Training Loss: 2.803231
Epoch: 89	Batch: 64	Training Loss: 2.823727
Epoch: 89	Batch: 72	Training Loss: 2.822965
Epoch: 89	Batch: 80	Training Loss: 2.815832

Epoch: 89	Batch: 88	Training Loss: 2.808524
Epoch: 89	Batch: 96	Training Loss: 2.803384
Epoch: 89	Batch: 104	Training Loss: 2.802253
Epoch: 89	Batch: 8	Validation Loss: 3.706259
Epoch: 90	Batch: 8	Training Loss: 2.651485
Epoch: 90	Batch: 16	Training Loss: 2.707715
Epoch: 90	Batch: 24	Training Loss: 2.740087
Epoch: 90	Batch: 32	Training Loss: 2.771836
Epoch: 90	Batch: 40	Training Loss: 2.772439
Epoch: 90	Batch: 48	Training Loss: 2.773932
Epoch: 90	Batch: 56	Training Loss: 2.786294
Epoch: 90	Batch: 64	Training Loss: 2.788383
Epoch: 90	Batch: 72	Training Loss: 2.795963
Epoch: 90	Batch: 80	Training Loss: 2.800784
Epoch: 90	Batch: 88	Training Loss: 2.784107
Epoch: 90	Batch: 96	Training Loss: 2.778112
Epoch: 90	Batch: 104	Training Loss: 2.776717
Epoch: 90	Batch: 8	Validation Loss: 3.807458
Epoch: 91	Batch: 8	Training Loss: 2.790716
Epoch: 91	Batch: 16	Training Loss: 2.858960
Epoch: 91	Batch: 24	Training Loss: 2.820336
Epoch: 91	Batch: 32	Training Loss: 2.825900
Epoch: 91	Batch: 40	Training Loss: 2.796630
Epoch: 91	Batch: 48	Training Loss: 2.795894
Epoch: 91	Batch: 56	Training Loss: 2.800459
Epoch: 91	Batch: 64	Training Loss: 2.797832
Epoch: 91	Batch: 72	Training Loss: 2.795772
Epoch: 91	Batch: 80	Training Loss: 2.794529
Epoch: 91	Batch: 88	Training Loss: 2.803136
Epoch: 91	Batch: 96	Training Loss: 2.791321
Epoch: 91	Batch: 104	Training Loss: 2.789677
Epoch: 91	Batch: 8	Validation Loss: 3.618351
Epoch: 92	Batch: 8	Training Loss: 2.678402
Epoch: 92	Batch: 16	Training Loss: 2.699584
Epoch: 92	Batch: 24	Training Loss: 2.760685
Epoch: 92	Batch: 32	Training Loss: 2.742068
Epoch: 92	Batch: 40	Training Loss: 2.722647
Epoch: 92	Batch: 48	Training Loss: 2.727848
Epoch: 92	Batch: 56	Training Loss: 2.744501
Epoch: 92	Batch: 64	Training Loss: 2.738050
Epoch: 92	Batch: 72	Training Loss: 2.756573
Epoch: 92	Batch: 80	Training Loss: 2.747060
Epoch: 92	Batch: 88	Training Loss: 2.744365
Epoch: 92	Batch: 96	Training Loss: 2.738437
Epoch: 92	Batch: 104	Training Loss: 2.746472
Epoch: 92	Batch: 8	Validation Loss: 3.554716
Epoch: 93	Batch: 8	Training Loss: 2.833957
Epoch: 93	Batch: 16	Training Loss: 2.815432

Epoch: 93	Batch: 24	Training Loss: 2.767169
Epoch: 93	Batch: 32	Training Loss: 2.751800
Epoch: 93	Batch: 40	Training Loss: 2.770301
Epoch: 93	Batch: 48	Training Loss: 2.758579
Epoch: 93	Batch: 56	Training Loss: 2.751752
Epoch: 93	Batch: 64	Training Loss: 2.750995
Epoch: 93	Batch: 72	Training Loss: 2.755659
Epoch: 93	Batch: 80	Training Loss: 2.766056
Epoch: 93	Batch: 88	Training Loss: 2.761304
Epoch: 93	Batch: 96	Training Loss: 2.755390
Epoch: 93	Batch: 104	Training Loss: 2.760415
Epoch: 93	Batch: 8	Validation Loss: 3.581565
Epoch: 94	Batch: 8	Training Loss: 2.660611
Epoch: 94	Batch: 16	Training Loss: 2.690305
Epoch: 94	Batch: 24	Training Loss: 2.671259
Epoch: 94	Batch: 32	Training Loss: 2.707053
Epoch: 94	Batch: 40	Training Loss: 2.707433
Epoch: 94	Batch: 48	Training Loss: 2.707296
Epoch: 94	Batch: 56	Training Loss: 2.707674
Epoch: 94	Batch: 64	Training Loss: 2.716090
Epoch: 94	Batch: 72	Training Loss: 2.721942
Epoch: 94	Batch: 80	Training Loss: 2.726972
Epoch: 94	Batch: 88	Training Loss: 2.724187
Epoch: 94	Batch: 96	Training Loss: 2.727689
Epoch: 94	Batch: 104	Training Loss: 2.729263
Epoch: 94	Batch: 8	Validation Loss: 3.680174
Epoch: 95	Batch: 8	Training Loss: 2.620099
Epoch: 95	Batch: 16	Training Loss: 2.701012
Epoch: 95	Batch: 24	Training Loss: 2.705206
Epoch: 95	Batch: 32	Training Loss: 2.736152
Epoch: 95	Batch: 40	Training Loss: 2.734511
Epoch: 95	Batch: 48	Training Loss: 2.723797
Epoch: 95	Batch: 56	Training Loss: 2.717304
Epoch: 95	Batch: 64	Training Loss: 2.726488
Epoch: 95	Batch: 72	Training Loss: 2.739628
Epoch: 95	Batch: 80	Training Loss: 2.737938
Epoch: 95	Batch: 88	Training Loss: 2.744324
Epoch: 95	Batch: 96	Training Loss: 2.737755
Epoch: 95	Batch: 104	Training Loss: 2.740401
Epoch: 95	Batch: 8	Validation Loss: 3.605968
Epoch: 96	Batch: 8	Training Loss: 2.670628
Epoch: 96	Batch: 16	Training Loss: 2.668263
Epoch: 96	Batch: 24	Training Loss: 2.674782
Epoch: 96	Batch: 32	Training Loss: 2.726374
Epoch: 96	Batch: 40	Training Loss: 2.734926
Epoch: 96	Batch: 48	Training Loss: 2.726255
Epoch: 96	Batch: 56	Training Loss: 2.722383
Epoch: 96	Batch: 64	Training Loss: 2.726874

Epoch: 96	Batch: 72	Training Loss: 2.734051
Epoch: 96	Batch: 80	Training Loss: 2.719608
Epoch: 96	Batch: 88	Training Loss: 2.723898
Epoch: 96	Batch: 96	Training Loss: 2.726094
Epoch: 96	Batch: 104	Training Loss: 2.719156
Epoch: 96	Batch: 8	Validation Loss: 3.895114
Epoch: 97	Batch: 8	Training Loss: 2.869263
Epoch: 97	Batch: 16	Training Loss: 2.773883
Epoch: 97	Batch: 24	Training Loss: 2.739584
Epoch: 97	Batch: 32	Training Loss: 2.715837
Epoch: 97	Batch: 40	Training Loss: 2.710060
Epoch: 97	Batch: 48	Training Loss: 2.721425
Epoch: 97	Batch: 56	Training Loss: 2.725686
Epoch: 97	Batch: 64	Training Loss: 2.739587
Epoch: 97	Batch: 72	Training Loss: 2.732514
Epoch: 97	Batch: 80	Training Loss: 2.726458
Epoch: 97	Batch: 88	Training Loss: 2.713547
Epoch: 97	Batch: 96	Training Loss: 2.705659
Epoch: 97	Batch: 104	Training Loss: 2.712822
Epoch: 97	Batch: 8	Validation Loss: 3.703794
Epoch: 98	Batch: 8	Training Loss: 2.689025
Epoch: 98	Batch: 16	Training Loss: 2.703820
Epoch: 98	Batch: 24	Training Loss: 2.695616
Epoch: 98	Batch: 32	Training Loss: 2.724057
Epoch: 98	Batch: 40	Training Loss: 2.711726
Epoch: 98	Batch: 48	Training Loss: 2.707067
Epoch: 98	Batch: 56	Training Loss: 2.706733
Epoch: 98	Batch: 64	Training Loss: 2.704034
Epoch: 98	Batch: 72	Training Loss: 2.714863
Epoch: 98	Batch: 80	Training Loss: 2.711123
Epoch: 98	Batch: 88	Training Loss: 2.706491
Epoch: 98	Batch: 96	Training Loss: 2.698954
Epoch: 98	Batch: 104	Training Loss: 2.705498
Epoch: 98	Batch: 8	Validation Loss: 3.473099
Epoch: 99	Batch: 8	Training Loss: 2.614511
Epoch: 99	Batch: 16	Training Loss: 2.625719
Epoch: 99	Batch: 24	Training Loss: 2.636107
Epoch: 99	Batch: 32	Training Loss: 2.612623
Epoch: 99	Batch: 40	Training Loss: 2.640444
Epoch: 99	Batch: 48	Training Loss: 2.642154
Epoch: 99	Batch: 56	Training Loss: 2.633201
Epoch: 99	Batch: 64	Training Loss: 2.636908
Epoch: 99	Batch: 72	Training Loss: 2.646098
Epoch: 99	Batch: 80	Training Loss: 2.656324
Epoch: 99	Batch: 88	Training Loss: 2.653836
Epoch: 99	Batch: 96	Training Loss: 2.653910
Epoch: 99	Batch: 104	Training Loss: 2.655555
Epoch: 99	Batch: 8	Validation Loss: 3.582605

Epoch: 100	Batch: 8	Training Loss: 2.566752
Epoch: 100	Batch: 16	Training Loss: 2.662109
Epoch: 100	Batch: 24	Training Loss: 2.675428
Epoch: 100	Batch: 32	Training Loss: 2.703871
Epoch: 100	Batch: 40	Training Loss: 2.708627
Epoch: 100	Batch: 48	Training Loss: 2.686762
Epoch: 100	Batch: 56	Training Loss: 2.683862
Epoch: 100	Batch: 64	Training Loss: 2.684140
Epoch: 100	Batch: 72	Training Loss: 2.688793
Epoch: 100	Batch: 80	Training Loss: 2.680430
Epoch: 100	Batch: 88	Training Loss: 2.682321
Epoch: 100	Batch: 96	Training Loss: 2.670723
Epoch: 100	Batch: 104	Training Loss: 2.681499
Epoch: 100	Batch: 8	Validation Loss: 3.586544

1.1.11 (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 [18]: 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))
```

```
In [19]: # call test function
```

```
test(loaders_scratch, model_scratch, criterion_scratch, use_cuda)
```

```
Test Loss: 3.409301
```

```
Test Accuracy: 21% (183/836)
```

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.

1.1.12 (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 [20]: ## TODO: Specify data loaders
```

```
data_dir = '/data/dog_images'
```

```
batch_size = 64
```

```
num_workers = 0
```

```
transforms = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
])
```

```
data_transfer = {'train': datasets.ImageFolder(data_dir + '/train', transform=transform),
                 'valid': datasets.ImageFolder(data_dir + '/valid', transform=transform),
                 'test': datasets.ImageFolder(data_dir + '/test', transform=transforms)}
}
```

```
loaders_transfer = {'train': DataLoader(data_transfer['train'], batch_size=batch_size,
                                       'valid': DataLoader(data_transfer['valid'], batch_size=batch_size,
                                       'test': DataLoader(data_transfer['test'], batch_size=batch_size, num_workers=num_workers)}
}
```

1.1.13 (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 [21]: import torchvision.models as models
import torch.nn as nn

## TODO: Specify model architecture
use_cuda = torch.cuda.is_available()
model_transfer = models.vgg16(pretrained=True)
last_layer = len(model_transfer.classifier) - 1
in_features = model_transfer.classifier[last_layer].in_features

#Lets freeze training for feature layers
for param in model_transfer.features.parameters():
    param.requires_grad = False

output_classes = len(data_transfer['train'].classes)
model_transfer.classifier[last_layer] = nn.Linear(in_features, output_classes)

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)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace)
    (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)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace)
    (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)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace)
    (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)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace)
    (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)
(26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(27): ReLU(inplace)
(28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(29): ReLU(inplace)
(30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(classifier): Sequential(
  (0): Linear(in_features=25088, out_features=4096, bias=True)
  (1): ReLU(inplace)
  (2): Dropout(p=0.5)
  (3): Linear(in_features=4096, out_features=4096, bias=True)
  (4): ReLU(inplace)
  (5): Dropout(p=0.5)
  (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: Steps to get the final CNN architecture for transfer learning. 1. Since I had already experimented with AlexNet and VGG16, I choose VGG16 because I got better results than with AlexNet on previous classification tasks (didn't try with other model architectures for this task) . 2. First test: transforms: RandomCropResize 224, No normalization, 2 epochs, batch size of 64 and learning rate of 0.001. I got a Test Accuracy of 44%. 3. Second test: RandomCropResize 224, No normalization, 10 epochs, batch size of 20 and learning rate of 0.001. I got a Test Accuracy of 57%. 4. Third test: RandomCropResize 224, No normalization, 2 epochs, batch size of 64 and learning rate of 0.01. I got a Test Accuracy of 63%. 5. Fourth test: RandomCropResize 224, No normalization, 2 epochs, batch size of 64 and learning rate of 0.01. I got a Test Accuracy of 63%. 6. Fifth and final test: Resize 224, Normalization to 0.5, 1 epoch, batch size of 64 and learning rate of 0.01. I got a Test Accuracy of 70%.

After a bit of reading about data augmentation I realized that it was suitable for training the features layers, but since I'm using an already trained features layer, I didn't need to augment data and avoid this kind of transformations. Only added the resize because of the VGG16 requirement and the normalization, as I learned in previous lessons. With a little more tweeking and training I think it will get an accuracy near 90% or more, but time is running out.

1.1.14 (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 [22]: import torch.optim as optim

criterion_transfer = nn.CrossEntropyLoss()
optimizer_transfer = optim.SGD(model_transfer.classifier.parameters(), lr=0.01)

```

1.1.15 (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 [23]: # train the model
        model_transfer = train(1, loaders_transfer, model_transfer, optimizer_transfer, criterion_transfer,
                               # load the model that got the best validation accuracy (uncomment the line below)
                               model_transfer.load_state_dict(torch.load('model_transfer.pt'))

Epoch: 1      Batch: 8      Training Loss: 4.794701
Epoch: 1      Batch: 16     Training Loss: 4.532710
Epoch: 1      Batch: 24     Training Loss: 4.310666
Epoch: 1      Batch: 32     Training Loss: 4.053474
Epoch: 1      Batch: 40     Training Loss: 3.816247
Epoch: 1      Batch: 48     Training Loss: 3.572870
Epoch: 1      Batch: 56     Training Loss: 3.339445
Epoch: 1      Batch: 64     Training Loss: 3.151958
Epoch: 1      Batch: 72     Training Loss: 2.987041
Epoch: 1      Batch: 80     Training Loss: 2.840433
Epoch: 1      Batch: 88     Training Loss: 2.706416
Epoch: 1      Batch: 96     Training Loss: 2.594364
Epoch: 1      Batch: 104    Training Loss: 2.490723
Validation loss decreased (inf --> 0.797821). Saving model ...
Validation loss decreased (0.797821 --> 0.719694). Saving model ...
Epoch: 1      Batch: 8      Validation Loss: 0.962438
```

1.1.16 (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 [24]: test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

Test Loss: 1.035649

Test Accuracy: 70% (590/836)

1.1.17 (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 [25]: ### TODO: Write a function that takes a path to an image as input
        ### and returns the dog breed that is predicted by the model.
```

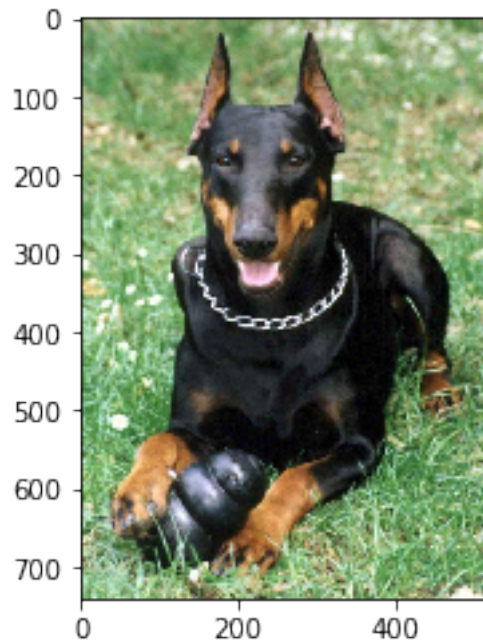
```

# 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 data_transfer['train'].classes]

def predict_breed_transfer(img_path):
    # load the image and return the predicted breed
    transform = transforms.Compose([transforms.Resize((224,224)),
                                    transforms.ToTensor(),
                                    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
                                    ])
    image = Image.open(img_path)
    img = transform(image)[None, ...]
    output = model_transfer(img.cuda() if use_cuda else img)
    _, predictions = torch.max(output,1)
    class_index = predictions.item()
    return class_names[class_index]

image_file = dog_files_short[82]
plt.imshow(Image.open(image_file))
plt.show()
predict_breed_transfer(image_file)

```



Out[25]: 'Doberman pinscher'



```

hello, human!

0
200
400
600
800
1000
1200
1400
0 500 1000

You look like a ...
Chinese_shar-pei

```



Sample Human Output

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!

1.1.18 (IMPLEMENTATION) Write your Algorithm

In [28]: *### 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):
        picture_kind = 'dog of the {} breed'
    elif face_detector(img_path):
        picture_kind = 'person that looks like a {}'
    dog_breed = predict_breed_transfer(img_path)
    print('This is a picture of a {} '.format(picture_kind.format(dog_breed)))
    plt.imshow(Image.open(img_path))
    plt.show()

```

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?

1.1.19 (IMPLEMENTATION) Test Your Algorithm on Sample Images!

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

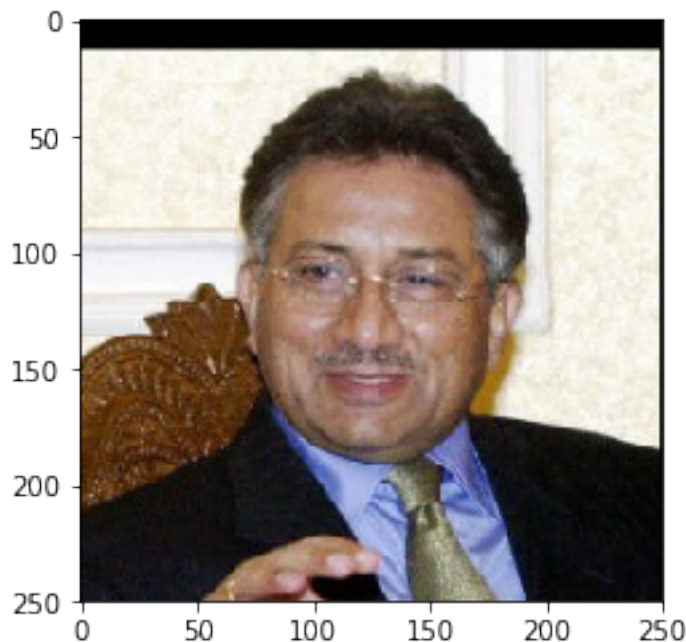
Question 6: Is the output better than you expected :) ? Or worse :(? Provide at least three possible points of improvement for your algorithm.

Answer: (Three possible points for improvement):

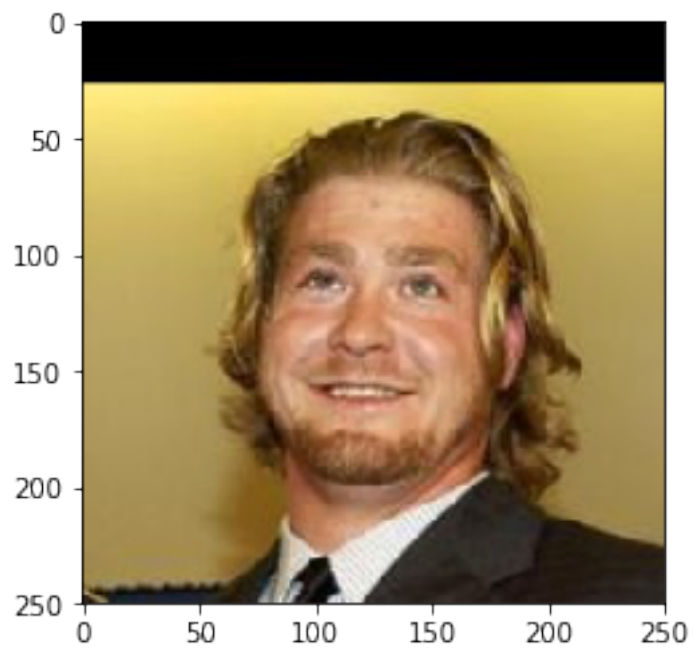
The predictions were better than I expected in some cases, specially for detecting breeds, and pretty good in telling the likeness of people to a dog breed. Some improvements could be: 1. Train a bit more the transfer model to reach a 90% accuracy or more. 2. Add a picture of the dog breed the person looks like. 3. Add a picture of the dog breed the given dog picture is being predicted.

```
In [30]: ## 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  
np.random.shuffle(human_files)  
np.random.shuffle(dog_files)  
for file in np.hstack((human_files[:5], dog_files[:5])):  
    run_app(file)
```

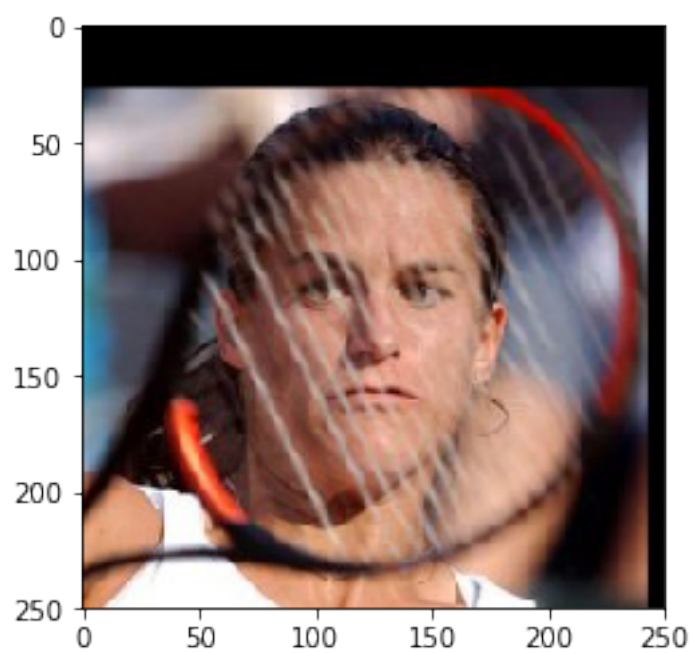
This is a picture of a person that looks like a Irish red and white setter



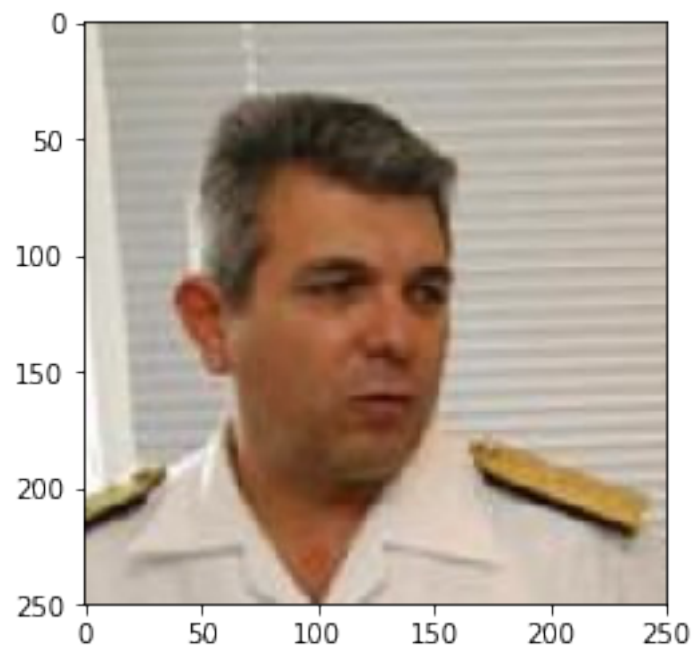
This is a picture of a person that looks like a Japanese chin



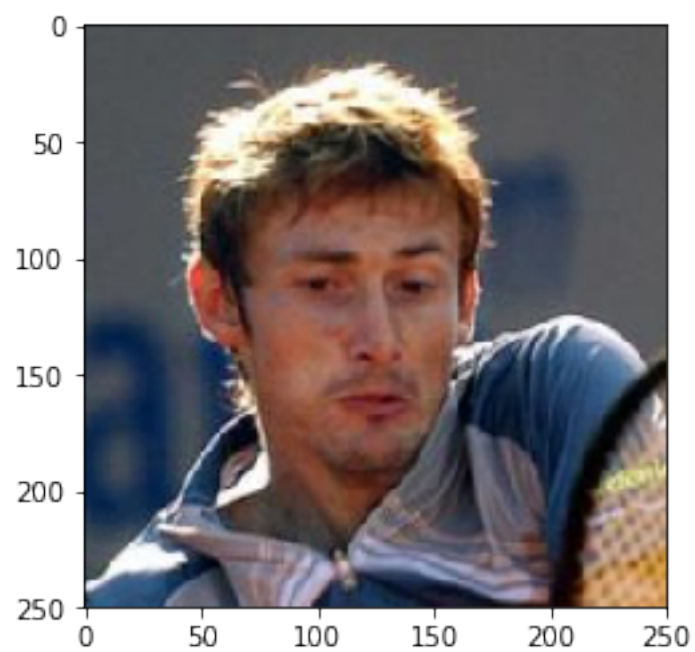
This is a picture of a person that looks like a Australian shepherd



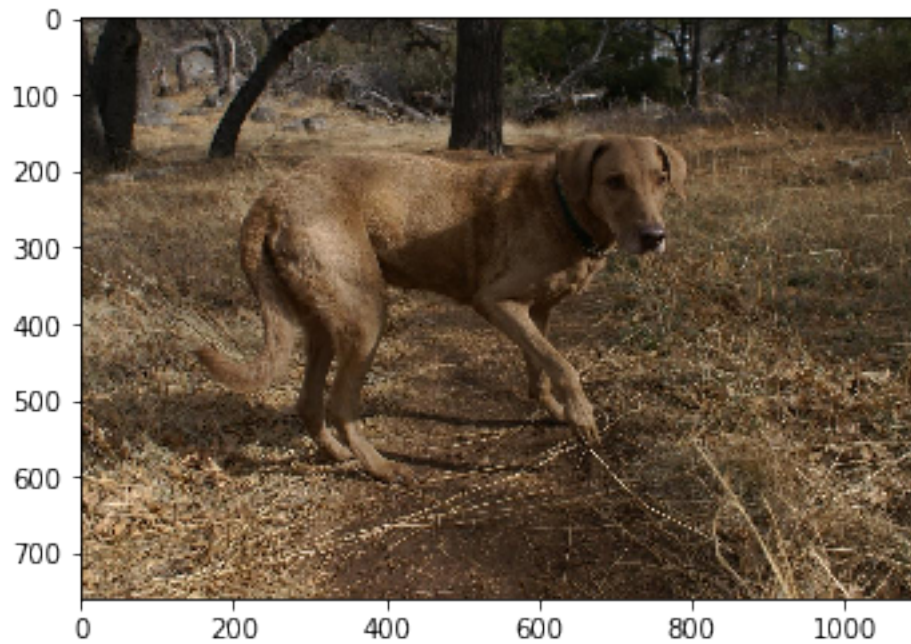
This is a picture of a person that looks like a Australian shepherd



This is a picture of a person that looks like a Chinese crested



This is a picture of a dog of the Chesapeake bay retriever breed



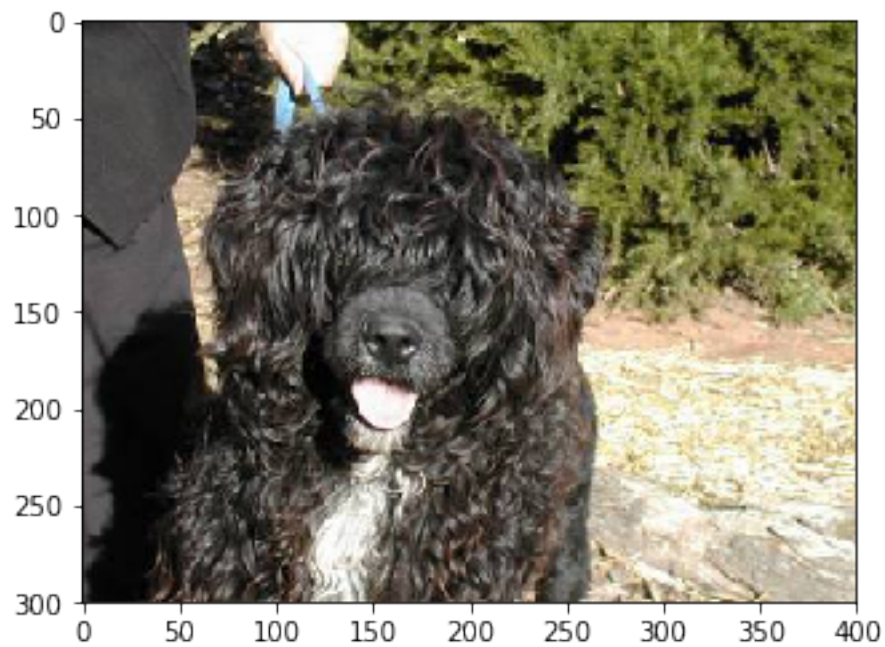
This is a picture of a dog of the Havanese breed



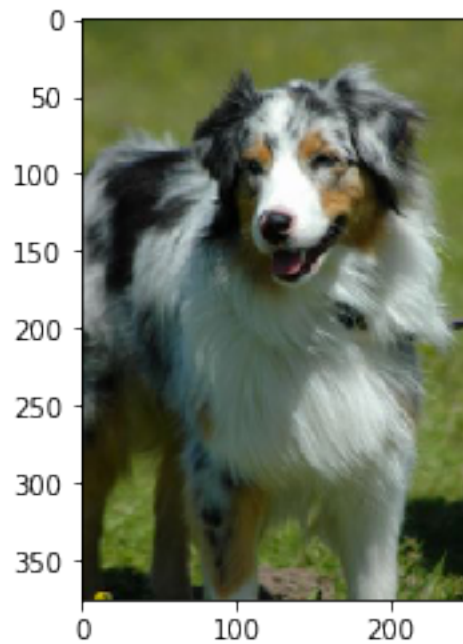
This is a picture of a dog of the Bearded collie breed



This is a picture of a dog of the Black russian terrier breed



This is a picture of a dog of the Australian shepherd breed



In []: