Lab 2: Cats vs Dogs

Deadline: January 30, 11:59pm

Late Penalty: There is a penalty-free grace period of one hour past the deadline. Any work that is submitted between 1 hour and 24 hours past the deadline will receive a 20% grade deduction. No other late work is accepted. Quercus submission time will be used, not your local computer time. You can submit your labs as many times as you want before the deadline, so please submit often and early.

Marking TA: Tinglin (Francis) Duan

This lab is partially based on an assignment developed by Prof. Jonathan Rose and Harris Chan.

In this lab, you will train a convolutional neural network to classify an image into one of two classes: "cat" or "dog". The code for the neural networks you train will be written for you, and you are not (yet!) expected to understand all provided code. However, by the end of the lab, you should be able to:

1. Understand at a high level the training loop for a machine learning model.
2. Understand the distinction between training, validation, and test data.
3. The concepts of overfitting and underfitting.
4. Investigate how different hyperparameters, such as learning rate and batch size, affect the success of training.
5. Compare an ANN (aka Multi-Layer Perceptron) with a CNN.

What to submit

Submit a PDF file containing all your code, outputs, and write-up from parts 1-5. You can produce a PDF of your Google Colab file by going to **File > Print** and then save as PDF. The Colab instructions has more information.

Do not submit any other files produced by your code.

Include a link to your colab file in your submission.

Please use Google Colab to complete this assignment. If you want to use Jupyter Notebook, please complete the assignment and upload your Jupyter Notebook file to Google Colab for submission.

With Colab, you can export a PDF file using the menu option `File -> Print` and save as PDF file. **Adjust the scaling to ensure that the text is not cutoff at the margins.**

Colab Link

Include a link to your colab file here

Colab Link: https://colab.research.google.com/drive/1XnlUmeAW00HVtrQ5CRaLRFg77CTGvDDS

```
import numpy as np
import time
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
from torch.utils.data.sampler import SubsetRandomSampler
import torchvision.transforms as transforms
```

Part 0. Helper Functions

We will be making use of the following helper functions. You will be asked to look at and possibly modify some of these, but you are not expected to understand all of them.

You should look at the function names and read the docstrings. If you are curious, come back and explore the code *after* making some progress on the lab.

```
################################################################################
# Data Loading

def get_relevant_indices(dataset, classes, target_classes):
```

```
        """ Return the indices for datapoints in the dataset that belongs to the
        desired target classes, a subset of all possible classes.

        Args:
            dataset: Dataset object
            classes: A list of strings denoting the name of each class
            target_classes: A list of strings denoting the name of desired classes
                            Should be a subset of the 'classes'
        Returns:
            indices: list of indices that have labels corresponding to one of the
                     target classes
        """
        indices = []
        for i in range(len(dataset)):
            # Check if the label is in the target classes
            label_index = dataset[i][1] # ex: 3
            label_class = classes[label_index] # ex: 'cat'
            if label_class in target_classes:
                indices.append(i)
        return indices

    def get_data_loader(target_classes, batch_size):
        """ Returns the indices for datapoints in the dataset that
        belongs to the desired target classes, a subset of all possible classes.

        Args:
            dataset: Dataset object
            classes: A list of strings denoting the name of each class
            target_classes: A list of strings denoting the name of the desired
                            classes. Should be a subset of the argument 'classes'
        Returns:
            indices: list of indices that have labels corresponding to one of the
                     target classes
        """
        classes = ('plane', 'car', 'bird', 'cat',
                   'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
        ########################################################################
        # The output of torchvision datasets are PILImage images of range [0, 1].
```

```
        # We transform them to Tensors of normalized range [-1, 1].
        transform = transforms.Compose(
            [transforms.ToTensor(),
             transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
        trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                                download=True, transform=transform)
        # Get the list of indices to sample from
        relevant_train_indices = get_relevant_indices(
                trainset,
                classes,
                target_classes)
        # Split into train and validation
        np.random.seed(1000) # Fixed numpy random seed for reproducible shuffling
        np.random.shuffle(relevant_train_indices)
        split = int(len(relevant_train_indices) * 0.8)
        relevant_train_indices, relevant_val_indices = relevant_train_indices[:split], relevant_train_indices[split:]
        train_sampler = SubsetRandomSampler(relevant_train_indices)
        train_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                                   num_workers=1, sampler=train_sampler)
        val_sampler = SubsetRandomSampler(relevant_val_indices)
        val_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                                 num_workers=1, sampler=val_sampler)
        testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                               download=True, transform=transform)
        relevant_test_indices = get_relevant_indices(testset, classes, target_classes)
        test_sampler = SubsetRandomSampler(relevant_test_indices)
        test_loader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                                  num_workers=1, sampler=test_sampler)
        return train_loader, val_loader, test_loader, classes

    ################################################################################
    # Training
    def get_model_name(name, batch_size, learning_rate, epoch):
        """ Generate a name for the model consisting of all the hyperparameter values

        Args:
            config: Configuration object containing the hyperparameters
        Returns:
```

```
        path: A string with the hyperparameter name and value concatenated
    """
    path = "model_{0}_bs{1}_lr{2}_epoch{3}".format(name,
                                                   batch_size,
                                                   learning_rate,
                                                   epoch)
    return path

def normalize_label(labels):
    """
    Given a tensor containing 2 possible values, normalize this to 0/1

    Args:
        labels: a 1D tensor containing two possible scalar values
    Returns:
        A tensor normalize to 0/1 value
    """
    max_val = torch.max(labels)
    min_val = torch.min(labels)
    norm_labels = (labels - min_val)/(max_val - min_val)
    return norm_labels

def evaluate(net, loader, criterion):
    """ Evaluate the network on the validation set.

     Args:
         net: PyTorch neural network object
         loader: PyTorch data loader for the validation set
         criterion: The loss function
     Returns:
         err: A scalar for the avg classification error over the validation set
         loss: A scalar for the average loss function over the validation set
     """
    total_loss = 0.0
    total_err = 0.0
    total_epoch = 0
    for i, data in enumerate(loader, 0):
        inputs, labels = data
```

```
            inputs, labels = data
            labels = normalize_label(labels)  # Convert labels to 0/1
            outputs = net(inputs)
            loss = criterion(outputs, labels.float())
            corr = (outputs > 0.0).squeeze().long() != labels
            total_err += int(corr.sum())
            total_loss += loss.item()
            total_epoch += len(labels)
        err = float(total_err) / total_epoch
        loss = float(total_loss) / (i + 1)
        return err, loss

    ##############################################################################
    # Training Curve
    def plot_training_curve(path):
        """ Plots the training curve for a model run, given the csv files
        containing the train/validation error/loss.

        Args:
            path: The base path of the csv files produced during training
        """
        import matplotlib.pyplot as plt
        train_err = np.loadtxt("{}_train_err.csv".format(path))
        val_err = np.loadtxt("{}_val_err.csv".format(path))
        train_loss = np.loadtxt("{}_train_loss.csv".format(path))
        val_loss = np.loadtxt("{}_val_loss.csv".format(path))
        plt.title("Train vs Validation Error")
        n = len(train_err) # number of epochs
        plt.plot(range(1,n+1), train_err, label="Train")
        plt.plot(range(1,n+1), val_err, label="Validation")
        plt.xlabel("Epoch")
        plt.ylabel("Error")
        plt.legend(loc='best')
        plt.show()
        plt.title("Train vs Validation Loss")
        plt.plot(range(1,n+1), train_loss, label="Train")
        plt.plot(range(1,n+1), val_loss, label="Validation")
        plt.xlabel("Epoch")
```

```
    plt.ylabel("Loss")
    plt.legend(loc='best')
    plt.show()
```

Part 1. Visualizing the Data [7 pt]

We will make use of some of the CIFAR-10 data set, which consists of colour images of size 32x32 pixels belonging to 10 categories. You can find out more about the dataset at https://www.cs.toronto.edu/~kriz/cifar.html

For this assignment, we will only be using the cat and dog categories. We have included code that automatically downloads the dataset the first time that the main script is run.

```
# This will download the CIFAR-10 dataset to a folder called "data"
# the first time you run this code.
train_loader, val_loader, test_loader, classes = get_data_loader(
    target_classes=["cat", "dog"],
    batch_size=1) # One image per batch
```

```
0it [00:00, ?it/s]Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
 99%|█████████▉| 169402368/170498071 [00:12<00:00, 17017325.48it/s]Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

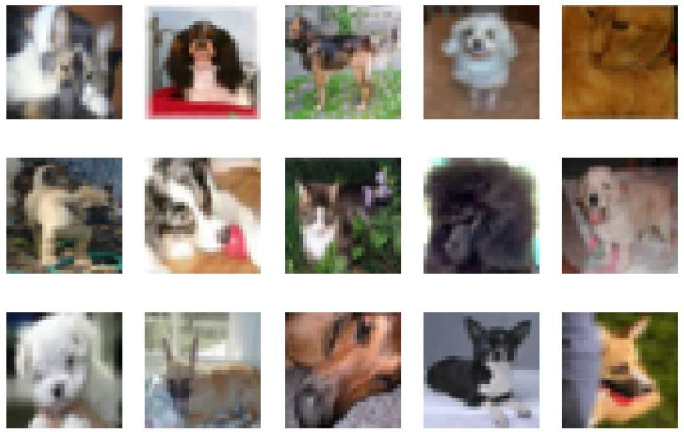
Part (a) -- 1 pt

Visualize some of the data by running the code below. Include the visualization in your writeup.

(You don't need to submit anything else.)

```
import matplotlib.pyplot as plt

k = 0
for images, labels in train_loader:
    # since batch_size = 1, there is only 1 image in `images`
    image = images[0]
    # place the colour channel at the end, instead of at the beginning
```

```
    img = np.transpose(image, [1,2,0])
    # normalize pixel intensity values to [0, 1]
    img = img / 2 + 0.5
    plt.subplot(3, 5, k+1)
    plt.axis('off')
    plt.imshow(img)

    k += 1
    if k > 14:
        break
```



Part (b) -- 3 pt

How many training examples do we have for the combined `cat` and `dog` classes? What about validation examples? What about test examples?

```
print("Number of training examples: ", len(train_loader))
print("Number of validation examples: ", len(val_loader))
print("Number of test examples: ", len(test_loader))
```

```
Number of training examples:  8000
Number of validation examples:  2000
Number of test examples:  2000
```

Part (c) -- 3pt

Why do we need a validation set when training our model? What happens if we judge the performance of our models using the training set loss/error instead of the validation set loss/error?

Answer for part 1(c)

A validation set is required to interprete the performance of our model. We need to identify whether the model is improving or not, and doing so is not possible with the training data set as our model. This is because our model was trained on that data set, and so our interpretations of performance based on training data will be inaccurate as our model will be biased towards the training data set.

Part 2. Training [15 pt]

We define two neural networks, a `LargeNet` and `SmallNet`. We'll be training the networks in this section.

You won't understand fully what these networks are doing until the next few classes, and that's okay. For this assignment, please focus on learning how to train networks, and how hyperparameters affect training.

```
class LargeNet(nn.Module):
    def __init__(self):
        super(LargeNet, self).__init__()
        self.name = "large"
        self.conv1 = nn.Conv2d(3, 5, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(5, 10, 5)
        self.fc1 = nn.Linear(10 * 5 * 5, 32)
        self.fc2 = nn.Linear(32, 1)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 10 * 5 * 5)
        x = F.relu(self.fc1(x))
```

```
        x = self.fc2(x)
        x = x.squeeze(1) # Flatten to [batch_size]
        return x


class SmallNet(nn.Module):
    def __init__(self):
        super(SmallNet, self).__init__()
        self.name = "small"
        self.conv = nn.Conv2d(3, 5, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc = nn.Linear(5 * 7 * 7, 1)

    def forward(self, x):
        x = self.pool(F.relu(self.conv(x)))
        x = self.pool(x)
        x = x.view(-1, 5 * 7 * 7)
        x = self.fc(x)
        x = x.squeeze(1) # Flatten to [batch_size]
        return x


small_net = SmallNet()
large_net = LargeNet()
```

Part (a) -- 2pt

The methods `small_net.parameters()` and `large_net.parameters()` produces an iterator of all the trainable parameters of the network. These parameters are torch tensors containing many scalar values.

We haven't learned how how the parameters in these high-dimensional tensors will be used, but we should be able to count the number of parameters. Measuring the number of parameters in a network is one way of measuring the "size" of a network.

What is the total number of parameters in `small_net` and in `large_net`? (Hint: how many numbers are in each tensor?)

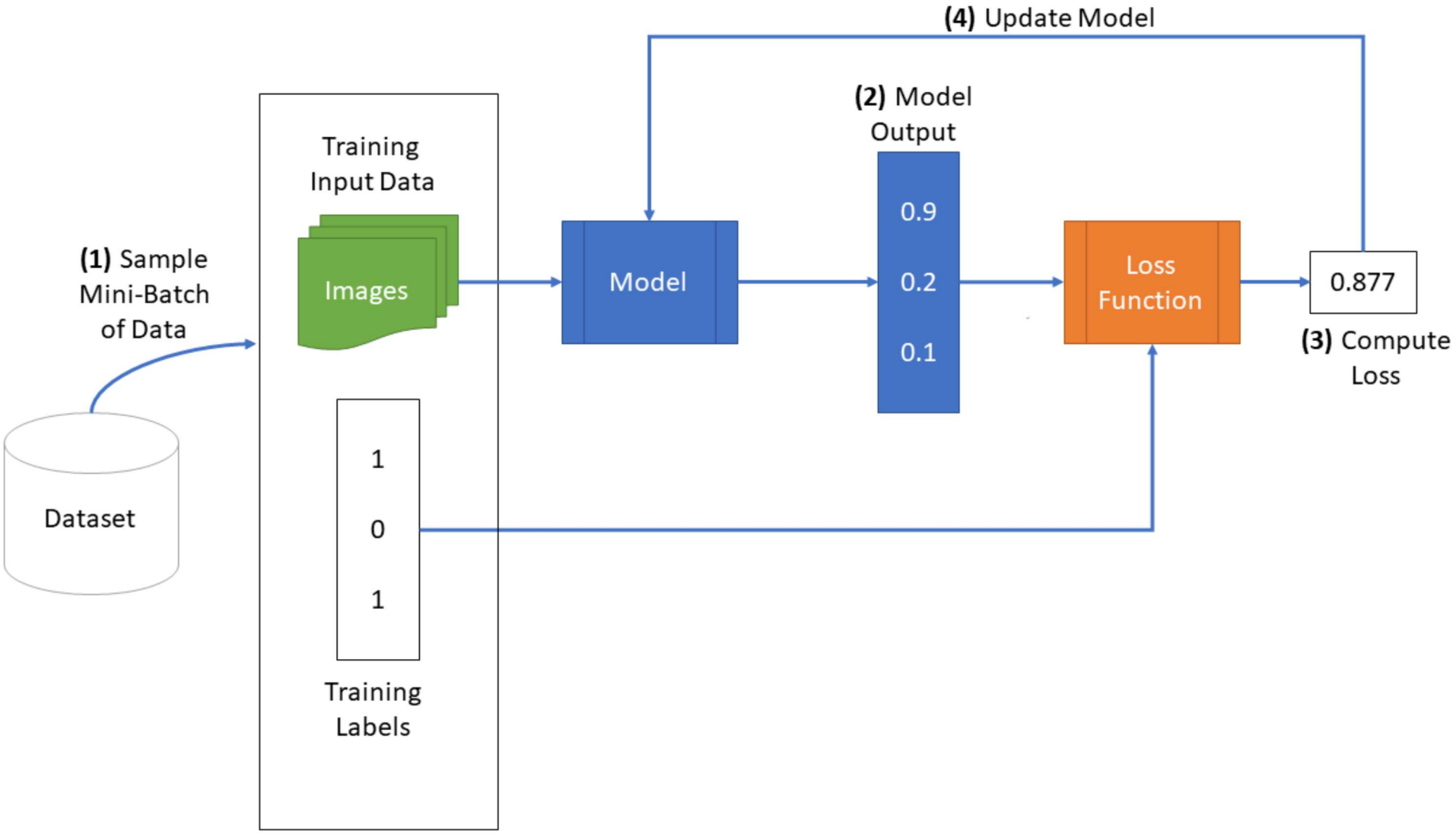
```
small_net_size = 0
large_net_size = 0
```

```
for param in small_net.parameters():
    small_net_size += param.nelement()  # nelement() gives number of elements in tensor

for param in large_net.parameters():
    large_net_size += param.nelement()

print("Size of small_net: ", small_net_size)
print("Size of large_net: ", large_net_size)
```

```
Size of small_net:  386
Size of large_net:  9705
```

The function train_net

The function `train_net` below takes an untrained neural network (like `small_net` and `large_net`) and several other parameters. You should be able to understand how this function works. The figure below shows the high level training loop for a machine learning model:




```
def train_net(net, batch_size=64, learning_rate=0.01, num_epochs=30):
    ########################################################################
    # Train a classifier on cats vs dogs
    target_classes = ["cat", "dog"]
    ########################################################################
```

```
# Fixed PyTorch random seed for reproducible result
torch.manual_seed(1000)
#################################################################################
# Obtain the PyTorch data loader objects to load batches of the datasets
train_loader, val_loader, test_loader, classes = get_data_loader(
        target_classes, batch_size)
#################################################################################
# Define the Loss function and optimizer
# The loss function will be Binary Cross Entropy (BCE). In this case we
# will use the BCEWithLogitsLoss which takes unnormalized output from
# the neural network and scalar label.
# Optimizer will be SGD with Momentum.
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
#################################################################################
# Set up some numpy arrays to store the training/test loss/erruracy
train_err = np.zeros(num_epochs)
train_loss = np.zeros(num_epochs)
val_err = np.zeros(num_epochs)
val_loss = np.zeros(num_epochs)
#################################################################################
# Train the network
# Loop over the data iterator and sample a new batch of training data
# Get the output from the network, and optimize our loss function.
start_time = time.time()
for epoch in range(num_epochs):  # loop over the dataset multiple times
    total_train_loss = 0.0
    total_train_err = 0.0
    total_epoch = 0
    for i, data in enumerate(train_loader, 0):
        # Get the inputs
        inputs, labels = data
        labels = normalize_label(labels) # Convert labels to 0/1
        # Zero the parameter gradients
        optimizer.zero_grad()
        # Forward pass, backward pass, and optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels.float())
```

```
            loss = criterion(outputs, labels.float())
            loss.backward()
            optimizer.step()
            # Calculate the statistics
            corr = (outputs > 0.0).squeeze().long() != labels
            total_train_err += int(corr.sum())
            total_train_loss += loss.item()
            total_epoch += len(labels)
        train_err[epoch] = float(total_train_err) / total_epoch
        train_loss[epoch] = float(total_train_loss) / (i+1)
        val_err[epoch], val_loss[epoch] = evaluate(net, val_loader, criterion)
        print(("Epoch {}: Train err: {}, Train loss: {} |"+
               "Validation err: {}, Validation loss: {}").format(
                   epoch + 1,
                   train_err[epoch],
                   train_loss[epoch],
                   val_err[epoch],
                   val_loss[epoch]))
        # Save the current model (checkpoint) to a file
        model_path = get_model_name(net.name, batch_size, learning_rate, epoch)
        torch.save(net.state_dict(), model_path)
    print('Finished Training')
    end_time = time.time()
    elapsed_time = end_time - start_time
    print("Total time elapsed: {:.2f} seconds".format(elapsed_time))
    # Write the train/test loss/err into CSV file for plotting later
    epochs = np.arange(1, num_epochs + 1)
    np.savetxt("{}_train_err.csv".format(model_path), train_err)
    np.savetxt("{}_train_loss.csv".format(model_path), train_loss)
    np.savetxt("{}_val_err.csv".format(model_path), val_err)
    np.savetxt("{}_val_loss.csv".format(model_path), val_loss)
```

Part (b) -- 1pt

The parameters to the function `train_net` are hyperparameters of our neural network. We made these hyperparameters easy to modify so that we can tune them later on.

What are the default values of the parameters `batch_size`, `learning_rate`, and `num_epochs`?

Answer to part 2(b)

The default value of `batch_size` is 64.
The default value of `learning_rate` is 0.01.
The default value of `num_epochs` is 30.

Part (c) -- 3 pt

What files are written to disk when we call `train_net` with `small_net`, and train for 5 epochs? Provide a list of all the files written to disk, and what information the files contain.

Answer to part 2(c)

The files written to disk are:

`model_small_bs64_lr0.01_epoch0`: Contains the model state after the first epoch.
`model_small_bs64_lr0.01_epoch1`: Contains the model state after the second epoch.
`model_small_bs64_lr0.01_epoch2`: Contains the model state after the third epoch.
`model_small_bs64_lr0.01_epoch3`: Contains the model state after the fourth epoch.
`model_small_bs64_lr0.01_epoch4`: Contains the model state after the fifth epoch.

`model_small_bs64_lr0.01_epoch4_train_err.csv`: Contains the training error for each epoch.
`model_small_bs64_lr0.01_epoch4_train_loss.csv`: Contains the training loss for each epoch.
`model_small_bs64_lr0.01_epoch4_val_err.csv`: Contains the validation error for each epoch.
`model_small_bs64_lr0.01_epoch4_val_loss.csv`: Contains the validation loss for each epoch.

Part (d) -- 2pt

Train both `small_net` and `large_net` using the function `train_net` and its default parameters. The function will write many files to disk, including a model checkpoint (saved values of model weights) at the end of each epoch.

If you are using Google Colab, you will need to mount Google Drive so that the files generated by `train_net` gets saved. We will be using these files in part (d). (See the Google Colab tutorial for more information about this.)

Report the total time elapsed when training each network. Which network took longer to train? Why?

```
# Since the function writes files to disk, you will need to mount
# your Google Drive. If you are working on the lab locally, you
# can comment out this code.

from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.a

Enter your authorization code:
··········
Mounted at /content/gdrive
```

```
train_net(small_net)
train_net(large_net)
```

```
Epoch 9: Train err: 0.28825, Train loss: 0.5644446897506714 |Validation err: 0.3055, Validation loss: 0.58567407540977
Epoch 10: Train err: 0.28725, Train loss: 0.5581038711071015 |Validation err: 0.3145, Validation loss: 0.5816246382892132
Epoch 11: Train err: 0.28475, Train loss: 0.5567858433723449 |Validation err: 0.318, Validation loss: 0.5790610983967781
Epoch 12: Train err: 0.2805, Train loss: 0.55062513256073 |Validation err: 0.3115, Validation loss: 0.5929448371753097
Epoch 13: Train err: 0.283, Train loss: 0.5540926654338837 |Validation err: 0.311, Validation loss: 0.5808670064434409
Epoch 14: Train err: 0.276625, Train loss: 0.5480497236251831 |Validation err: 0.315, Validation loss: 0.5971809262409806
Epoch 15: Train err: 0.279125, Train loss: 0.5472877254486084 |Validation err: 0.3085, Validation loss: 0.5812386386096478
Epoch 16: Train err: 0.28075, Train loss: 0.5510434954166412 |Validation err: 0.3135, Validation loss: 0.5923737743869424
Epoch 17: Train err: 0.27525, Train loss: 0.5469000973701477 |Validation err: 0.3105, Validation loss: 0.5756622822955251
Epoch 18: Train err: 0.274375, Train loss: 0.5438379611968994 |Validation err: 0.3025, Validation loss: 0.5784980366006494
Epoch 19: Train err: 0.2725, Train loss: 0.5408743357658387 |Validation err: 0.303, Validation loss: 0.5863784374669194
Epoch 20: Train err: 0.27075, Train loss: 0.5414348757266998 |Validation err: 0.2975, Validation loss: 0.5861031990498304
Epoch 21: Train err: 0.27625, Train loss: 0.5417127969264984 |Validation err: 0.3, Validation loss: 0.5744085051119328
Epoch 22: Train err: 0.27575, Train loss: 0.5432483026981354 |Validation err: 0.31, Validation loss: 0.5896512549370527
Epoch 23: Train err: 0.27525, Train loss: 0.5411429040431976 |Validation err: 0.305, Validation loss: 0.5806681029498577
Epoch 24: Train err: 0.270625, Train loss: 0.5392013971805573 |Validation err: 0.304, Validation loss: 0.5936433784663677
Epoch 25: Train err: 0.272375, Train loss: 0.5372497029304505 |Validation err: 0.3, Validation loss: 0.5793415121734142
Epoch 26: Train err: 0.2695, Train loss: 0.5370332744121552 |Validation err: 0.295, Validation loss: 0.5753222815692425
Epoch 27: Train err: 0.268, Train loss: 0.5378211262226105 |Validation err: 0.3, Validation loss: 0.5877141170203686
Epoch 28: Train err: 0.27175, Train loss: 0.5387655966281891 |Validation err: 0.2955, Validation loss: 0.5770982261747122
Epoch 29: Train err: 0.27175, Train loss: 0.5361493678092957 |Validation err: 0.299, Validation loss: 0.5862517161294818
Epoch 30: Train err: 0.273, Train loss: 0.5411494836807251 |Validation err: 0.2975, Validation loss: 0.582723043859005
Finished Training
Total time elapsed: 91.28 seconds
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.456125, Train loss: 0.6903426952362061 |Validation err: 0.436, Validation loss: 0.6829197835177183
Epoch 2: Train err: 0.411125, Train loss: 0.6731154193878174 |Validation err: 0.4185, Validation loss: 0.6815852019935846
Epoch 3: Train err: 0.378625, Train loss: 0.6544155316352844 |Validation err: 0.377, Validation loss: 0.6441417522728443
Epoch 4: Train err: 0.35925, Train loss: 0.6314589819908142 |Validation err: 0.3575, Validation loss: 0.6405176166445017
Epoch 5: Train err: 0.340125, Train loss: 0.6146110210418702 |Validation err: 0.335, Validation loss: 0.6136596575379372
Epoch 6: Train err: 0.323375, Train loss: 0.5959996874332428 |Validation err: 0.313, Validation loss: 0.6006096377968788
Epoch 7: Train err: 0.31175, Train loss: 0.5867838211059571 |Validation err: 0.322, Validation loss: 0.5927210161462426
Epoch 8: Train err: 0.299125, Train loss: 0.5698475897312164 |Validation err: 0.3135, Validation loss: 0.5873603783547878
Epoch 9: Train err: 0.295125, Train loss: 0.563510098695755 |Validation err: 0.3135, Validation loss: 0.5823186310008168
Epoch 10: Train err: 0.2825, Train loss: 0.5522625997066498 |Validation err: 0.2885, Validation loss: 0.5653563849627972
Epoch 11: Train err: 0.28, Train loss: 0.5414507019519806 |Validation err: 0.2945, Validation loss: 0.5801051538437605
Epoch 12: Train err: 0.2645, Train loss: 0.5267171688079834 |Validation err: 0.2885, Validation loss: 0.5763991326093674
Epoch 13: Train err: 0.262875, Train loss: 0.520485140800476 |Validation err: 0.294, Validation loss: 0.5769719425588846
Epoch 14: Train err: 0.257125, Train loss: 0.5079327461719513 |Validation err: 0.2835, Validation loss: 0.5680468725040555
Epoch 15: Train err: 0.248875, Train loss: 0.5032503781318665 |Validation err: 0.2825, Validation loss: 0.571021655574441
Epoch 16: Train err: 0.243125, Train loss: 0.49480326771736144 |Validation err: 0.292, Validation loss: 0.5743452254682779
```

```
Epoch 17: Train err: 0.23775, Train loss: 0.4856313602924347 |Validation err: 0.297, Validation loss: 0.5728962095454335
Epoch 18: Train err: 0.22325, Train loss: 0.46759171795845034 |Validation err: 0.289, Validation loss: 0.5747311022132635
Epoch 19: Train err: 0.224, Train loss: 0.4628159022331238 |Validation err: 0.2795, Validation loss: 0.5686680972576141
Epoch 20: Train err: 0.217625, Train loss: 0.44815078806877134 |Validation err: 0.3125, Validation loss: 0.6260916534811258
Epoch 21: Train err: 0.218125, Train loss: 0.450642728805542 |Validation err: 0.298, Validation loss: 0.5889916503801942
Epoch 22: Train err: 0.215625, Train loss: 0.4435607440471649 |Validation err: 0.291, Validation loss: 0.6031636940315366
Epoch 23: Train err: 0.202, Train loss: 0.43080435919761656 |Validation err: 0.2905, Validation loss: 0.6026991698890924
Epoch 24: Train err: 0.193125, Train loss: 0.41328356313705444 |Validation err: 0.3065, Validation loss: 0.6266334261745214
Epoch 25: Train err: 0.18725, Train loss: 0.4024776523113251 |Validation err: 0.2935, Validation loss: 0.6215320937335491
Epoch 26: Train err: 0.180125, Train loss: 0.38705208897590637 |Validation err: 0.287, Validation loss: 0.6452029421925545
Epoch 27: Train err: 0.171875, Train loss: 0.37652074682712555 |Validation err: 0.2985, Validation loss: 0.6688472265377641
Epoch 28: Train err: 0.164, Train loss: 0.3582114727497101 |Validation err: 0.3025, Validation loss: 0.6762499855831265
Epoch 29: Train err: 0.1575, Train loss: 0.34462737309932706 |Validation err: 0.3025, Validation loss: 0.7205265946686268
Epoch 30: Train err: 0.15125, Train loss: 0.3398781954050064 |Validation err: 0.2965, Validation loss: 0.6769934436306357
Finished Training
Total time elapsed: 106.44 seconds
```

`small_net` took 91.28 seconds to train. `large_net` took 106.44 seconds to train. `large_net` took 15.16 seconds longer to train that `small_net`.

This is because `large_net` has 9705 parameters compared to `small_net` which has 386 parameters. The larger number of parameters means it takes more processing time to calculate and update values for all of the parameters.

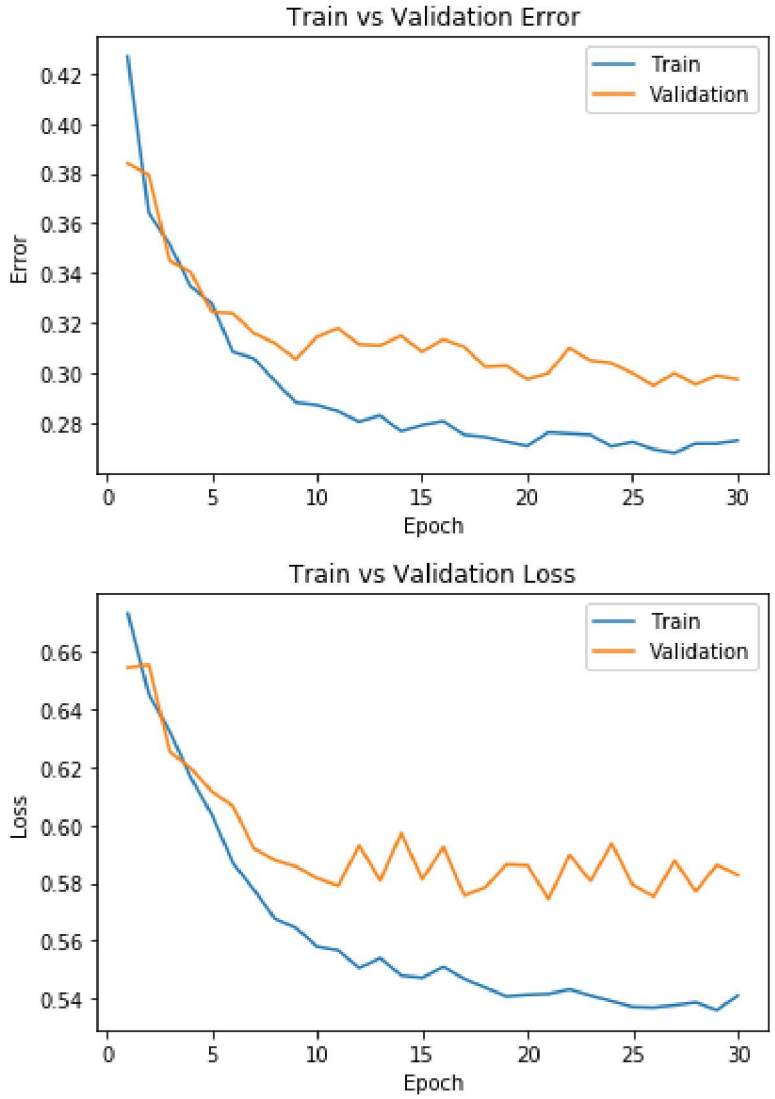
Part (e) - 2pt

Use the function `plot_training_curve` to display the trajectory of the training/validation error and the training/validation loss. You will need to use the function `get_model_name` to generate the argument to the `plot_training_curve` function.

Do this for both the small network and the large network. Include both plots in your writeup.

```
print("Plots for small network:\n")
model_path = get_model_name("small", batch_size=64, learning_rate=0.01, epoch=29)
plot_training_curve(model_path)
```

```
Plots for small network:
```



```
print("Plots for large network:\n")
model_path = get_model_name("large", batch_size=64, learning_rate=0.01, epoch=29)
plot_training_curve(model_path)
```

```
Plots for large network:

                    Train vs Validation Error
```

Part (f) - 5pt

Describe what you notice about the training curve. How do the curves differ for `small_net` and `large_net`? Identify any occurences of underfitting and overfitting.

Answer to part 2(f)

The training error and loss both decrease as the epoch number increases, which makes sense as the model is learning.

The training loss and error decrease much faster for `large_net` than for `small_net`. However, there is a case of overfitting for `large_net`. We can see from the plot that the training loss decreases while the validation loss increases. This is a sign of overfitting as the model is overly biased towards the training data and so underperforms for the validation data.
Meanwhile, for `small_net`, the training loss/error eventually stops decreasing after a certain number of epochs (around 10-15). This is a sign of underfitting as the model is no longer improving on the training data set.

So `large_net` is overfitting to the training data and `small_net` is underfitting to the training data.

Part 3. Optimization Parameters [12 pt]

For this section, we will work with `large_net` only.

Part (a) - 3pt

Train `large_net` with all default parameters, except set `learning_rate=0.001`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *lowering* the learning rate.

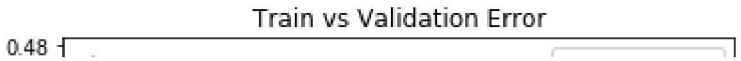
```
# Note: When we re-construct the model, we start the training
```

```
# with *random weights*. If we omit this code, the values of
# the weights will still be the previously trained values.
large_net = LargeNet()
train_net(large_net, learning_rate=0.001)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.47625, Train loss: 0.6928360023498535 |Validation err: 0.467, Validation loss: 0.6924686655402184
Epoch 2: Train err: 0.448625, Train loss: 0.6922589688301086 |Validation err: 0.4305, Validation loss: 0.6916493382304907
Epoch 3: Train err: 0.43575, Train loss: 0.6916067261695862 |Validation err: 0.4285, Validation loss: 0.6908544152975082
Epoch 4: Train err: 0.43, Train loss: 0.6908614072799683 |Validation err: 0.4245, Validation loss: 0.6896600145846605
Epoch 5: Train err: 0.434375, Train loss: 0.689919647693634 |Validation err: 0.4195, Validation loss: 0.6886944100260735
Epoch 6: Train err: 0.435875, Train loss: 0.6887412457466126 |Validation err: 0.4195, Validation loss: 0.6867826543748379
Epoch 7: Train err: 0.43675, Train loss: 0.6873777341842652 |Validation err: 0.418, Validation loss: 0.6851988434791565
Epoch 8: Train err: 0.437375, Train loss: 0.6859265742301941 |Validation err: 0.4115, Validation loss: 0.6831980552524328
Epoch 9: Train err: 0.4245, Train loss: 0.6844038491249085 |Validation err: 0.4115, Validation loss: 0.6808850187808275
Epoch 10: Train err: 0.424125, Train loss: 0.6828485760688782 |Validation err: 0.408, Validation loss: 0.6783478930592537
Epoch 11: Train err: 0.42525, Train loss: 0.6812336773872375 |Validation err: 0.4125, Validation loss: 0.6780175268650055
Epoch 12: Train err: 0.419875, Train loss: 0.6796316781044006 |Validation err: 0.4125, Validation loss: 0.6753130666911602
Epoch 13: Train err: 0.41475, Train loss: 0.6777912259101868 |Validation err: 0.415, Validation loss: 0.675702191889286
Epoch 14: Train err: 0.41225, Train loss: 0.6761095609664917 |Validation err: 0.412, Validation loss: 0.6739692315459251
Epoch 15: Train err: 0.409125, Train loss: 0.6744713163375855 |Validation err: 0.4145, Validation loss: 0.6706809662282467
Epoch 16: Train err: 0.406375, Train loss: 0.6727418246269226 |Validation err: 0.4105, Validation loss: 0.6707680653780699
Epoch 17: Train err: 0.40125, Train loss: 0.6713045845031739 |Validation err: 0.4045, Validation loss: 0.6671513859182596
Epoch 18: Train err: 0.399625, Train loss: 0.6696718058586121 |Validation err: 0.4055, Validation loss: 0.6646745707839727
Epoch 19: Train err: 0.400875, Train loss: 0.667906973361969 |Validation err: 0.396, Validation loss: 0.665499659255147
Epoch 20: Train err: 0.392125, Train loss: 0.665784423828125 |Validation err: 0.405, Validation loss: 0.6625944431871176
Epoch 21: Train err: 0.389375, Train loss: 0.664623098373413 |Validation err: 0.395, Validation loss: 0.6606688015162945
Epoch 22: Train err: 0.388375, Train loss: 0.6623682513236999 |Validation err: 0.3935, Validation loss: 0.6616888716816902
Epoch 23: Train err: 0.384125, Train loss: 0.660144766330719 |Validation err: 0.3975, Validation loss: 0.657391270622611
Epoch 24: Train err: 0.382375, Train loss: 0.6583982200622559 |Validation err: 0.386, Validation loss: 0.6561238467693329
Epoch 25: Train err: 0.378625, Train loss: 0.6554944009780884 |Validation err: 0.3875, Validation loss: 0.6552787534892559
Epoch 26: Train err: 0.376875, Train loss: 0.6531199479103088 |Validation err: 0.3865, Validation loss: 0.6531632654368877
Epoch 27: Train err: 0.375125, Train loss: 0.6503734455108643 |Validation err: 0.3875, Validation loss: 0.6519878040999174
Epoch 28: Train err: 0.3715, Train loss: 0.6476535792350769 |Validation err: 0.388, Validation loss: 0.6483295131474733
Epoch 29: Train err: 0.367875, Train loss: 0.645130642414093 |Validation err: 0.382, Validation loss: 0.6458809245377779
Epoch 30: Train err: 0.3625, Train loss: 0.6423453125953674 |Validation err: 0.3785, Validation loss: 0.6438876297324896
Finished Training
Total time elapsed: 105.82 seconds
```

The model took 105.82 seconds to train. Compared to the previous time of 106.44 seconds, this is not a significant difference, so the time elapsed is about the same.

```
model_path = get_model_name("large", batch_size=64, learning_rate=0.001, epoch=29)
plot_training_curve(model_path)
```



At this lower learning rate of 0.001, the rate at which training error/loss decreases is much slower than it was at a learning rate of 0.01. Compared to the previous plots, these new plots reach about the same error/loss values for training and validation at 30 epochs that the previous plots reached as soon as 5-7 epochs. As the training loss/error values and validation loss/error values are still strongly correlated, no overfitting has occurred yet after 30 epochs for the lower learning rate.

Part (b) - 3pt

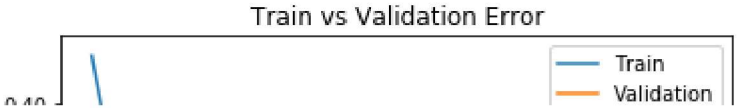
Train `large_net` with all default parameters, except set `learning_rate=0.1`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *increasing* the learning rate.

```
large_net = LargeNet()
train_net(large_net, learning_rate=0.1)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.426875, Train loss: 0.6742900676727295 |Validation err: 0.3695, Validation loss: 0.6364889536052942
Epoch 2: Train err: 0.361625, Train loss: 0.6373908457756042 |Validation err: 0.3655, Validation loss: 0.6439082939177752
Epoch 3: Train err: 0.371625, Train loss: 0.6370397086143493 |Validation err: 0.3515, Validation loss: 0.6133540160953999
Epoch 4: Train err: 0.348, Train loss: 0.6141975626945496 |Validation err: 0.3365, Validation loss: 0.6044933125376701
Epoch 5: Train err: 0.333875, Train loss: 0.6029216282367706 |Validation err: 0.3245, Validation loss: 0.5935317613184452
Epoch 6: Train err: 0.3175, Train loss: 0.5871682240962982 |Validation err: 0.324, Validation loss: 0.5884642750024796
Epoch 7: Train err: 0.308, Train loss: 0.5718630583286285 |Validation err: 0.3095, Validation loss: 0.5934673277661204
Epoch 8: Train err: 0.306625, Train loss: 0.5659790558815002 |Validation err: 0.3305, Validation loss: 0.5951285297051072
Epoch 9: Train err: 0.300875, Train loss: 0.5650376663208008 |Validation err: 0.3365, Validation loss: 0.6021498944610357
Epoch 10: Train err: 0.281625, Train loss: 0.5438764057159424 |Validation err: 0.3135, Validation loss: 0.5791513873264194
Epoch 11: Train err: 0.27325, Train loss: 0.5326492521762848 |Validation err: 0.303, Validation loss: 0.6063874159008265
Epoch 12: Train err: 0.265, Train loss: 0.5173962540626525 |Validation err: 0.301, Validation loss: 0.5923424074426293
Epoch 13: Train err: 0.260625, Train loss: 0.5154193744659424 |Validation err: 0.3165, Validation loss: 0.6365817207843065
Epoch 14: Train err: 0.26425, Train loss: 0.5193490188121795 |Validation err: 0.343, Validation loss: 0.6715553980320692
Epoch 15: Train err: 0.25475, Train loss: 0.5023991143703461 |Validation err: 0.3225, Validation loss: 0.6021238509565592
Epoch 16: Train err: 0.245875, Train loss: 0.4958816692829132 |Validation err: 0.315, Validation loss: 0.6298653511330485
Epoch 17: Train err: 0.232875, Train loss: 0.4806650359630585 |Validation err: 0.314, Validation loss: 0.6941358242183924
Epoch 18: Train err: 0.229375, Train loss: 0.47152764201164243 |Validation err: 0.3315, Validation loss: 0.6477560913190246
Epoch 19: Train err: 0.234125, Train loss: 0.4797332081794739 |Validation err: 0.333, Validation loss: 0.646070503629744
Epoch 20: Train err: 0.219, Train loss: 0.4615775098800659 |Validation err: 0.324, Validation loss: 0.6784388227388263
Epoch 21: Train err: 0.218375, Train loss: 0.4566231663227081 |Validation err: 0.331, Validation loss: 0.730235455557704
Epoch 22: Train err: 0.2185, Train loss: 0.46099011921882627 |Validation err: 0.3285, Validation loss: 0.6987572200596333
Epoch 23: Train err: 0.21025, Train loss: 0.4500437686443329 |Validation err: 0.3325, Validation loss: 0.7979978676885366
Epoch 24: Train err: 0.205, Train loss: 0.4398671627044678 |Validation err: 0.335, Validation loss: 0.7653014371171594
Epoch 25: Train err: 0.216, Train loss: 0.4674131067991257 |Validation err: 0.3365, Validation loss: 0.7716402560472488
Epoch 26: Train err: 0.20975, Train loss: 0.4478739421367645 |Validation err: 0.343, Validation loss: 0.7236454039812088
Epoch 27: Train err: 0.200375, Train loss: 0.434726841211319 |Validation err: 0.3535, Validation loss: 0.7641464285552502
Epoch 28: Train err: 0.2115, Train loss: 0.4540190188884735 |Validation err: 0.3145, Validation loss: 0.8971310313791037
Epoch 29: Train err: 0.215875, Train loss: 0.4729398670196533 |Validation err: 0.3375, Validation loss: 0.8915724642574787
Epoch 30: Train err: 0.212, Train loss: 0.4490383931398392 |Validation err: 0.3845, Validation loss: 0.7677891906350851
Finished Training
Total time elapsed: 102.93 seconds
```

The model took 102.93 seconds to train, which compared to the 106.44 seconds at the default values can be considered a slightly significant improvement. So the time taken for the model to train at a learning rate of 0.1 was lower than the time taken for the model to train at a learning rate of 0.01.

```
model_path = get_model_name("large", batch_size=64, learning_rate=0.1, epoch=29)
plot_training_curve(model_path)
```



At the increased learning rate of 0.1, the rate at which training error/loss decreases is slower than the rate at the default learning rate of 0.01, as evidenced by the less steep gradient. We can also see that the model is not training properly and is very heavily overfitting to the training data, as after about 5-7 epochs the validation loss/error increases while the training loss/error decreases.

Part (c) - 3pt

Train `large_net` with all default parameters, including with `learning_rate=0.01`. Now, set `batch_size=512`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *increasing* the batch size.

```
large_net = LargeNet()
train_net(large_net, batch_size=512)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.48175, Train loss: 0.6929379403591156 |Validation err: 0.478, Validation loss: 0.6926823854446411
Epoch 2: Train err: 0.457625, Train loss: 0.6924103908240795 |Validation err: 0.434, Validation loss: 0.6917425096035004
Epoch 3: Train err: 0.437, Train loss: 0.6916500441730022 |Validation err: 0.4265, Validation loss: 0.6909130066633224
Epoch 4: Train err: 0.433625, Train loss: 0.6908450052142143 |Validation err: 0.424, Validation loss: 0.6897870898246765
Epoch 5: Train err: 0.434, Train loss: 0.6896936446428299 |Validation err: 0.424, Validation loss: 0.6881358623504639
Epoch 6: Train err: 0.43825, Train loss: 0.6883534081280231 |Validation err: 0.428, Validation loss: 0.68601293861866
Epoch 7: Train err: 0.439375, Train loss: 0.6866869702935219 |Validation err: 0.426, Validation loss: 0.6836968064308167
Epoch 8: Train err: 0.435375, Train loss: 0.6849769502878189 |Validation err: 0.412, Validation loss: 0.6814655214548111
Epoch 9: Train err: 0.423875, Train loss: 0.6832012832164764 |Validation err: 0.414, Validation loss: 0.6795923411846161
Epoch 10: Train err: 0.421125, Train loss: 0.6811087355017662 |Validation err: 0.416, Validation loss: 0.6771558523178101
Epoch 11: Train err: 0.42075, Train loss: 0.679402157664299 |Validation err: 0.4095, Validation loss: 0.6748124063014984
Epoch 12: Train err: 0.414875, Train loss: 0.6768044196069241 |Validation err: 0.4125, Validation loss: 0.6737014949321747
Epoch 13: Train err: 0.410375, Train loss: 0.6749668307602406 |Validation err: 0.412, Validation loss: 0.670610174536705
Epoch 14: Train err: 0.40725, Train loss: 0.6730880029499531 |Validation err: 0.4125, Validation loss: 0.6692066043615341
Epoch 15: Train err: 0.400375, Train loss: 0.6706768870353699 |Validation err: 0.41, Validation loss: 0.6672501415014267
Epoch 16: Train err: 0.397625, Train loss: 0.6691729240119457 |Validation err: 0.405, Validation loss: 0.6649037003517151
Epoch 17: Train err: 0.39375, Train loss: 0.6675690039992332 |Validation err: 0.401, Validation loss: 0.663024365901947
Epoch 18: Train err: 0.392875, Train loss: 0.664790865033865 |Validation err: 0.394, Validation loss: 0.6623962968587875
Epoch 19: Train err: 0.38625, Train loss: 0.6627316661179066 |Validation err: 0.3875, Validation loss: 0.6597267240285873
Epoch 20: Train err: 0.38175, Train loss: 0.6596068292856216 |Validation err: 0.4005, Validation loss: 0.6564352214336395
Epoch 21: Train err: 0.38575, Train loss: 0.6584859304130077 |Validation err: 0.3885, Validation loss: 0.6586581021547318
Epoch 22: Train err: 0.378375, Train loss: 0.6551184356212616 |Validation err: 0.386, Validation loss: 0.6528748720884323
Epoch 23: Train err: 0.372125, Train loss: 0.650881964713335 |Validation err: 0.3835, Validation loss: 0.6498024016618729
Epoch 24: Train err: 0.376875, Train loss: 0.6488037817180157 |Validation err: 0.386, Validation loss: 0.6474764347076416
Epoch 25: Train err: 0.368625, Train loss: 0.6445966511964798 |Validation err: 0.3825, Validation loss: 0.6473759114742279
Epoch 26: Train err: 0.372625, Train loss: 0.6428800038993359 |Validation err: 0.375, Validation loss: 0.6425630450248718
Epoch 27: Train err: 0.35925, Train loss: 0.6372313089668751 |Validation err: 0.3785, Validation loss: 0.6397574543952942
Epoch 28: Train err: 0.354375, Train loss: 0.6337887421250343 |Validation err: 0.37, Validation loss: 0.6403995752334595
Epoch 29: Train err: 0.35375, Train loss: 0.631144043058157 |Validation err: 0.366, Validation loss: 0.6335429400205612
Epoch 30: Train err: 0.35275, Train loss: 0.6283803880214691 |Validation err: 0.3675, Validation loss: 0.6324474662542343
Finished Training
Total time elapsed: 96.16 seconds
```

The time elapsed was 96.16 seconds which is a significant improvement to the 106.44 seconds at default values. So a larger batch size trained much faster.

```
model_path = get_model_name("large", batch_size=512, learning_rate=0.01, epoch=29)
plot_training_curve(model_path)
```

Increasing the batch size causes the model to learn very slowly. We can see the training and validation error/loss decrease very slowly as the epoch number increases. As was the case with a slow learning rate, the model has not yet overfitted to training data after 30 epochs.

Part (d) - 3pt

Train `large_net` with all default parameters, including with `learning_rate=0.01`. Now, set `batch_size=16`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *decreasing* the batch size.

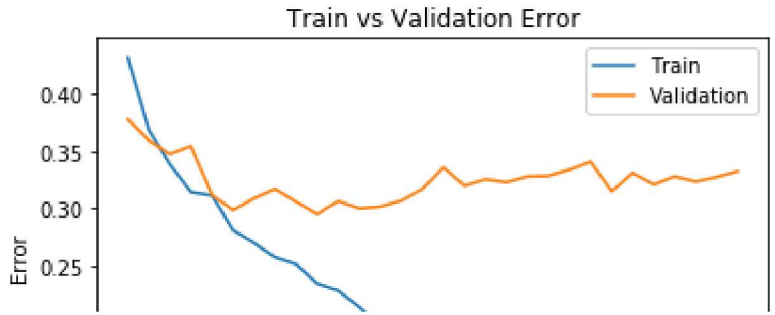
```
large_net = LargeNet()
train_net(large_net, batch_size=16)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.43175, Train loss: 0.6774821187257767 |Validation err: 0.378, Validation loss: 0.651921395778656
Epoch 2: Train err: 0.36875, Train loss: 0.6395755406022072 |Validation err: 0.3595, Validation loss: 0.6264406447410583
Epoch 3: Train err: 0.33875, Train loss: 0.6120929001569748 |Validation err: 0.3475, Validation loss: 0.6289729187488556
Epoch 4: Train err: 0.314375, Train loss: 0.5869516692757607 |Validation err: 0.3545, Validation loss: 0.6142054274082184
Epoch 5: Train err: 0.3115, Train loss: 0.5694829801917076 |Validation err: 0.312, Validation loss: 0.5763112711906433
Epoch 6: Train err: 0.281125, Train loss: 0.5458917077481746 |Validation err: 0.2985, Validation loss: 0.5741617560386658
Epoch 7: Train err: 0.269875, Train loss: 0.5297637034058571 |Validation err: 0.309, Validation loss: 0.6013907868862152
```

The time elapsed was 151.27 seconds which is significantly higher than the 106.44 seconds at default values. So a smaller batch size takes a lot longer to train.

```
Epoch 12: Train err: 0.21375, Train loss: 0.4457400677502155 |Validation err: 0.3, Validation loss: 0.6069887349605556
```

```
model_path = get_model_name("large", batch_size=16, learning_rate=0.01, epoch=29)
plot_training_curve(model_path)
```



From the plots, it can be seen that the training error/loss decreases faster at a smaller batch size. However, the plots indicate that the model is vastly overfitting to the training data, as the validation error/loss is increasing while the training error/loss is decreasing.

Part 4. Hyperparameter Search [6 pt]

Part (a) - 2pt

Based on the plots from above, choose another set of values for the hyperparameters (network, batch_size, learning_rate) that you think would help you improve the validation accuracy. Justify your choice.

```
network = "large"
my_batch_size = 128
my_learning_rate = 0.005
```

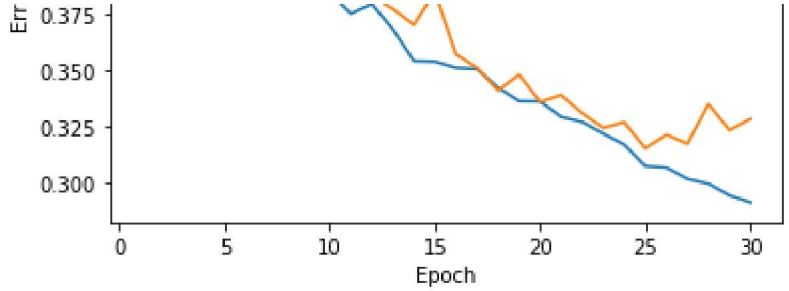
I chose the large network as it gave better results than the small network in the default cases.
The model overfits to the training data with the default hyperparameters. So, I need to slow down the rate at which the model trains.
From part 3, I can conclude that this can be achieved by increasing the batch size and/or decreasing the learning rate. So, I chose a batch size that is double the default value and a learning rate that is half the default value.

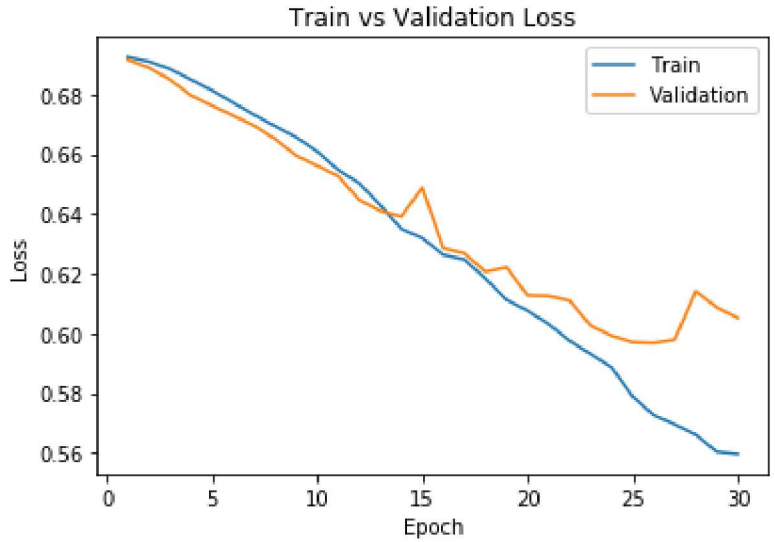
Part (b) - 1pt

Train the model with the hyperparameters you chose in part(a), and include the training curve.

```
large_net = LargeNet()
train_net(large_net, my_batch_size, my_learning_rate)
model_path = get_model_name(network, batch_size=my_batch_size, learning_rate=my_learning_rate, epoch=29)
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.466625, Train loss: 0.6925613293572078 |Validation err: 0.4305, Validation loss: 0.691625040024519
Epoch 2: Train err: 0.450375, Train loss: 0.6910346566684662 |Validation err: 0.4295, Validation loss: 0.6889703683555126
Epoch 3: Train err: 0.43025, Train loss: 0.6885915540513539 |Validation err: 0.417, Validation loss: 0.6850201450288296
Epoch 4: Train err: 0.43075, Train loss: 0.6850066270147052 |Validation err: 0.4135, Validation loss: 0.6797112785279751
Epoch 5: Train err: 0.421125, Train loss: 0.6813876486959911 |Validation err: 0.411, Validation loss: 0.676275547593832
Epoch 6: Train err: 0.41575, Train loss: 0.677297902485681 |Validation err: 0.4125, Validation loss: 0.6729632318019867
Epoch 7: Train err: 0.40525, Train loss: 0.6732194035772293 |Validation err: 0.4055, Validation loss: 0.6695893667638302
Epoch 8: Train err: 0.4005, Train loss: 0.669406773552062 |Validation err: 0.402, Validation loss: 0.6651132963597775
Epoch 9: Train err: 0.390375, Train loss: 0.6657012265826029 |Validation err: 0.397, Validation loss: 0.6595702804625034
Epoch 10: Train err: 0.38425, Train loss: 0.6609561291951982 |Validation err: 0.399, Validation loss: 0.6561421491205692
Epoch 11: Train err: 0.374875, Train loss: 0.6547279717430236 |Validation err: 0.39, Validation loss: 0.6526764445006847
Epoch 12: Train err: 0.379125, Train loss: 0.6502850140844073 |Validation err: 0.3825, Validation loss: 0.644701637327671
Epoch 13: Train err: 0.368125, Train loss: 0.6430118538084484 |Validation err: 0.377, Validation loss: 0.641075611114502
Epoch 14: Train err: 0.354, Train loss: 0.6349688399405706 |Validation err: 0.37, Validation loss: 0.6391588747501373
Epoch 15: Train err: 0.35375, Train loss: 0.6319383061121381 |Validation err: 0.3845, Validation loss: 0.6488435678184032
Epoch 16: Train err: 0.351, Train loss: 0.6263029357743641 |Validation err: 0.357, Validation loss: 0.6285315304994583
Epoch 17: Train err: 0.3505, Train loss: 0.6247079713003976 |Validation err: 0.351, Validation loss: 0.6268058456480503
Epoch 18: Train err: 0.34225, Train loss: 0.6183563111320375 |Validation err: 0.341, Validation loss: 0.6206603012979031
Epoch 19: Train err: 0.336375, Train loss: 0.6112425809814817 |Validation err: 0.348, Validation loss: 0.6221032552421093
Epoch 20: Train err: 0.33625, Train loss: 0.6075829740554567 |Validation err: 0.336, Validation loss: 0.6127016469836235
Epoch 21: Train err: 0.329375, Train loss: 0.6031279157078455 |Validation err: 0.339, Validation loss: 0.6125184185802937
Epoch 22: Train err: 0.327125, Train loss: 0.5975301123800731 |Validation err: 0.331, Validation loss: 0.6109913475811481
Epoch 23: Train err: 0.322, Train loss: 0.5930708031805735 |Validation err: 0.3245, Validation loss: 0.6026063859462738
Epoch 24: Train err: 0.317125, Train loss: 0.5886016421847873 |Validation err: 0.327, Validation loss: 0.5991837531328201
Epoch 25: Train err: 0.307625, Train loss: 0.5788375758935534 |Validation err: 0.3155, Validation loss: 0.5970390513539314
Epoch 26: Train err: 0.307, Train loss: 0.572593421216995 |Validation err: 0.3215, Validation loss: 0.5967941656708717
Epoch 27: Train err: 0.302, Train loss: 0.569515291660551 |Validation err: 0.3175, Validation loss: 0.5978767573833466
Epoch 28: Train err: 0.299875, Train loss: 0.5660480677135407 |Validation err: 0.335, Validation loss: 0.6140783578157425
Epoch 29: Train err: 0.29475, Train loss: 0.5603112285099332 |Validation err: 0.3235, Validation loss: 0.6086705103516579
Epoch 30: Train err: 0.291625, Train loss: 0.5595763666289193 |Validation err: 0.3285, Validation loss: 0.6051585711538792
Finished Training
Total time elapsed: 96.78 seconds
```





Part (c) - 2pt

Based on your result from Part(a), suggest another set of hyperparameter values to try. Justify your choice.

```
network = "large"
my_batch_size = 160
my_learning_rate = 0.004
```

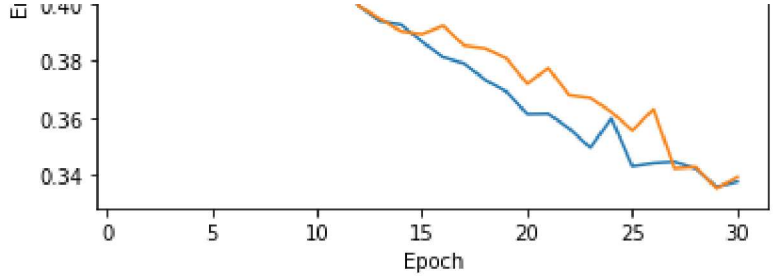
From the results in part (a), I can see that the validation error/loss is just beginning to diverge from the training error/loss at just under 30 epochs. This means my initial values chosen in part (a) trained the model well and required some fine tuning. So, I lowered the learning rate from 0.005 to 0.004 and increased the batch size from 128 to 160.

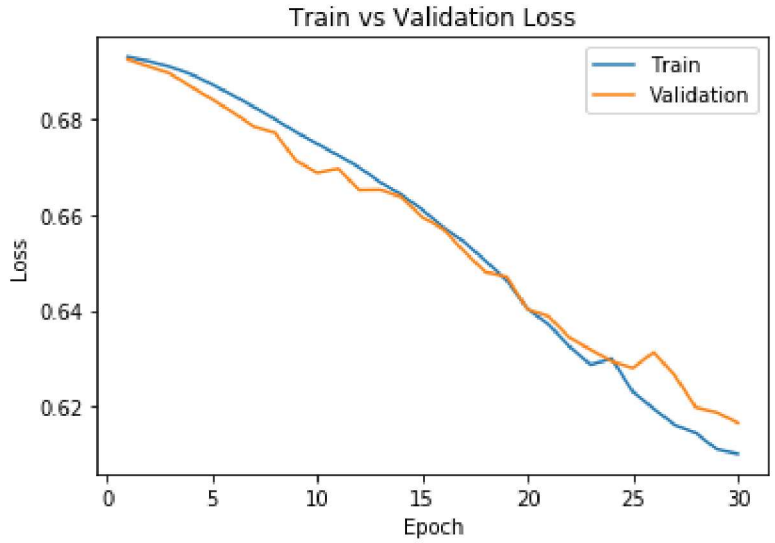
Part (d) - 1pt

Train the model with the hyperparameters you chose in part(c), and include the training curve.

```
large_net = LargeNet()
train_net(large_net, my_batch_size, my_learning_rate)
model_path = get_model_name(network, batch_size=my_batch_size, learning_rate=my_learning_rate, epoch=29)
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.47525, Train loss: 0.6927601087093354 |Validation err: 0.4515, Validation loss: 0.6922393166101896
Epoch 2: Train err: 0.444375, Train loss: 0.6918846428394317 |Validation err: 0.4365, Validation loss: 0.6907833356123704
Epoch 3: Train err: 0.431875, Train loss: 0.6907380867004395 |Validation err: 0.4215, Validation loss: 0.6893187302809495
Epoch 4: Train err: 0.4325, Train loss: 0.6891735577583313 |Validation err: 0.42, Validation loss: 0.6865964531898499
Epoch 5: Train err: 0.43775, Train loss: 0.6870565688610077 |Validation err: 0.416, Validation loss: 0.6839699286680955
Epoch 6: Train err: 0.431625, Train loss: 0.6847723662853241 |Validation err: 0.413, Validation loss: 0.6812272438636193
Epoch 7: Train err: 0.424, Train loss: 0.6823482763767242 |Validation err: 0.414, Validation loss: 0.6782730726095346
Epoch 8: Train err: 0.420625, Train loss: 0.6797999215126037 |Validation err: 0.4125, Validation loss: 0.6769580015769372
Epoch 9: Train err: 0.41375, Train loss: 0.6771175467967987 |Validation err: 0.41, Validation loss: 0.6711714542829074
Epoch 10: Train err: 0.409375, Train loss: 0.6747007894515992 |Validation err: 0.403, Validation loss: 0.668683831508343
Epoch 11: Train err: 0.4085, Train loss: 0.6723196375370025 |Validation err: 0.4095, Validation loss: 0.6695523858070374
Epoch 12: Train err: 0.3995, Train loss: 0.6697942447662354 |Validation err: 0.3995, Validation loss: 0.6650875394160931
Epoch 13: Train err: 0.394125, Train loss: 0.6667249929904938 |Validation err: 0.395, Validation loss: 0.6651410001974839
Epoch 14: Train err: 0.392875, Train loss: 0.6641173648834229 |Validation err: 0.3905, Validation loss: 0.6636131497529837
Epoch 15: Train err: 0.386875, Train loss: 0.6610246562957763 |Validation err: 0.3895, Validation loss: 0.6594449373391958
Epoch 16: Train err: 0.381375, Train loss: 0.6573198235034943 |Validation err: 0.3925, Validation loss: 0.6569574291889484
Epoch 17: Train err: 0.379125, Train loss: 0.6542769515514374 |Validation err: 0.3855, Validation loss: 0.6522834025896512
Epoch 18: Train err: 0.373375, Train loss: 0.6503806507587433 |Validation err: 0.3845, Validation loss: 0.6480285892119775
Epoch 19: Train err: 0.36925, Train loss: 0.64632164478302 |Validation err: 0.381, Validation loss: 0.6470905450674204
Epoch 20: Train err: 0.36125, Train loss: 0.6403820168972015 |Validation err: 0.372, Validation loss: 0.6402685275444617
Epoch 21: Train err: 0.361375, Train loss: 0.6371624970436096 |Validation err: 0.3775, Validation loss: 0.6388314366340637
Epoch 22: Train err: 0.356125, Train loss: 0.6325425946712494 |Validation err: 0.368, Validation loss: 0.6344084327037518
Epoch 23: Train err: 0.349375, Train loss: 0.6287547385692597 |Validation err: 0.367, Validation loss: 0.631902066560892
Epoch 24: Train err: 0.35975, Train loss: 0.6300295770168305 |Validation err: 0.362, Validation loss: 0.6294633975395789
Epoch 25: Train err: 0.34275, Train loss: 0.6231861472129822 |Validation err: 0.3555, Validation loss: 0.6280876306387094
Epoch 26: Train err: 0.343875, Train loss: 0.619621057510376 |Validation err: 0.363, Validation loss: 0.6313256942308866
Epoch 27: Train err: 0.344375, Train loss: 0.6162595891952515 |Validation err: 0.342, Validation loss: 0.6265690005742587
Epoch 28: Train err: 0.342, Train loss: 0.6145787894725799 |Validation err: 0.3425, Validation loss: 0.6198945824916546
Epoch 29: Train err: 0.3355, Train loss: 0.6111907041072846 |Validation err: 0.335, Validation loss: 0.6188406348228455
Epoch 30: Train err: 0.337375, Train loss: 0.6102187347412109 |Validation err: 0.339, Validation loss: 0.6166794529327979
Finished Training
Total time elapsed: 95.08 seconds
```





Part 4. Evaluating the Best Model [15 pt]

Part (a) - 1pt

Choose the **best** model that you have so far. This means choosing the best model checkpoint, including the choice of `small_net` vs `large_net`, the `batch_size`, `learning_rate`, **and the epoch number**.

Modify the code below to load your chosen set of weights to the model object `net`.

```
best_batch_size = 128
best_learning_rate = 0.005
```

```
best_epoch = 25


net = LargeNet()
model_path = get_model_name(net.name, batch_size=best_batch_size, learning_rate=best_learning_rate, epoch=best_epoch)
state = torch.load(model_path)
net.load_state_dict(state)
```

```
<All keys matched successfully>
```

Part (b) - 2pt

Justify your choice of model from part (a).

I chose the large net as it seemed to be performing better than the small net. The hyperparameter values chosen achieved good results for loss/error on both test and validation data sets while not overfitting to the data.

Part (c) - 2pt

Using the code in Part 0, any code from lecture notes, or any code that you write, compute and report the **test classification error** for your chosen model.

```
# If you use the `evaluate` function provided in part 0, you will need to
# set batch_size > 1
train_loader, val_loader, test_loader, classes = get_data_loader(
    target_classes=["cat", "dog"],
    batch_size=best_batch_size)

cr = nn.BCEWithLogitsLoss()

err, loss = evaluate(net, test_loader, cr)

print("Test Classification Error:", 100*(err), "%")
```

```
Files already downloaded and verified
Files already downloaded and verified
Test Classification Error: 32.4 %
```

Part (d) - 3pt

How does the test classification error compare with the **validation error**? Explain why you would expect the test error to be *higher* than the validation error.

Answer to part 5(d)

The test error is very similar to the validation error.
I would expect the test error to be higher than the validation error because the validation error is slowly improved upon.

Part (e) - 2pt

Why did we only use the test data set at the very end? Why is it important that we use the test data as little as possible?

Answer to part 5(e)

The test data is used to verify that the model is working as intended. The test data should be used as little as possible so that we are not biased towards the testing set when choosing a suitable model.

Part (f) - 5pt

Train a 2-layer ANN similar to what was used in Lab 1 to classific cats and dogs. Try out different hyperparameter settings to determine how well you can do on the validation dataset. Once satisified with the performance, you may test it out on the test data. How does the ANN model compare to your CNN model?

```
class ann(nn.Module):
```

```
    def __init__(self):
      super(ann, self).__init__()
      self.name = "dog_cat_ann"
      self.layer1 = nn.Linear(32*32*3, 50)
      self.layer2 = nn.Linear(50, 1)
    def forward(self, img):
      flattened = img.reshape(-1, 32*32*3)
      activation1 = self.layer1(flattened)
      activation1 = F.relu(activation1)
      activation2 = self.layer2(activation1)
      return activation2.squeeze(1)


ann_net = ann()
train_net(ann_net, batch_size=best_batch_size, learning_rate=best_learning_rate)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.4235, Train loss: 0.6740091878270346 |Validation err: 0.3995, Validation loss: 0.6599471867084503
Epoch 2: Train err: 0.384125, Train loss: 0.6510869444362701 |Validation err: 0.3905, Validation loss: 0.6533326134085655
Epoch 3: Train err: 0.369, Train loss: 0.6391554323453752 |Validation err: 0.3835, Validation loss: 0.6499537937343121
Epoch 4: Train err: 0.35975, Train loss: 0.6306753697849455 |Validation err: 0.391, Validation loss: 0.6529671959578991
Epoch 5: Train err: 0.3515, Train loss: 0.6247555017471313 |Validation err: 0.3895, Validation loss: 0.6463541723787785
Epoch 6: Train err: 0.3405, Train loss: 0.6166401902834574 |Validation err: 0.389, Validation loss: 0.6491648741066456
Epoch 7: Train err: 0.33625, Train loss: 0.6103231982579307 |Validation err: 0.379, Validation loss: 0.6463090963661671
Epoch 8: Train err: 0.32775, Train loss: 0.6018946208651104 |Validation err: 0.373, Validation loss: 0.6425482369959354
Epoch 9: Train err: 0.3225, Train loss: 0.5960990181044926 |Validation err: 0.385, Validation loss: 0.6476771794259548
Epoch 10: Train err: 0.315125, Train loss: 0.5893129971292284 |Validation err: 0.3735, Validation loss: 0.6474233418703079
Epoch 11: Train err: 0.30475, Train loss: 0.5793002399187239 |Validation err: 0.372, Validation loss: 0.6480303704738617
Epoch 12: Train err: 0.299, Train loss: 0.5696150452371628 |Validation err: 0.3675, Validation loss: 0.6473077572882175
Epoch 13: Train err: 0.29, Train loss: 0.5626712550246527 |Validation err: 0.375, Validation loss: 0.6553607508540154
Epoch 14: Train err: 0.283, Train loss: 0.5543543819397215 |Validation err: 0.378, Validation loss: 0.6576543711125851
Epoch 15: Train err: 0.27325, Train loss: 0.544013441082031 |Validation err: 0.363, Validation loss: 0.6527563184499741
Epoch 16: Train err: 0.270375, Train loss: 0.5366643298239935 |Validation err: 0.368, Validation loss: 0.6522750556468964
Epoch 17: Train err: 0.262625, Train loss: 0.5280466136478242 |Validation err: 0.3665, Validation loss: 0.6535562761127949
Epoch 18: Train err: 0.25175, Train loss: 0.5212060723985944 |Validation err: 0.3565, Validation loss: 0.6566001921892166
Epoch 19: Train err: 0.25025, Train loss: 0.512242631306724 |Validation err: 0.3735, Validation loss: 0.6784833408892155
Epoch 20: Train err: 0.244625, Train loss: 0.49993099674345953 |Validation err: 0.368, Validation loss: 0.6594806909561157
Epoch 21: Train err: 0.229625, Train loss: 0.48866517203194754 |Validation err: 0.3635, Validation loss: 0.6610830500721931
Epoch 22: Train err: 0.227875, Train loss: 0.47842979336541797 |Validation err: 0.37, Validation loss: 0.6829414516687393
Epoch 23: Train err: 0.21225, Train loss: 0.4671331601483481 |Validation err: 0.3555, Validation loss: 0.6594752557575703
Epoch 24: Train err: 0.204625, Train loss: 0.45567074086931014 |Validation err: 0.3555, Validation loss: 0.6655578054487705
Epoch 25: Train err: 0.19625, Train loss: 0.4445208145512475 |Validation err: 0.369, Validation loss: 0.6705720350146294
Epoch 26: Train err: 0.192875, Train loss: 0.4327349520864941 |Validation err: 0.3665, Validation loss: 0.6815636530518532
Epoch 27: Train err: 0.1905, Train loss: 0.42707013468893745 |Validation err: 0.355, Validation loss: 0.6927022933959961
Epoch 28: Train err: 0.185625, Train loss: 0.4176167817342849 |Validation err: 0.3605, Validation loss: 0.6986956708133221
Epoch 29: Train err: 0.172, Train loss: 0.400107601332286O7 |Validation err: 0.3585, Validation loss: 0.7014249227941036
Epoch 30: Train err: 0.16525, Train loss: 0.3915802964142391 |Validation err: 0.358, Validation loss: 0.7013579718768597
Finished Training
Total time elapsed: 66.49 seconds
```

```
model_path = get_model_name("dog_cat_ann", batch_size=best_batch_size, learning_rate=best_learning_rate, epoch=29)
plot_training_curve(model_path)
```