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/1GWUvEFIGl_6ZJ-X_vPBAFMt-cu4lTpQg

```
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
        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.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
17050096it [00:04, 42338552.88it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Part (a) -- 1 pt

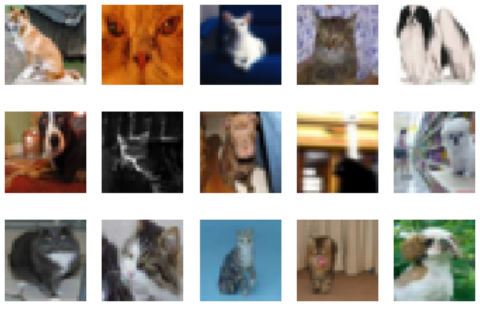
Visualize some of the data by running the code below. Include the visualization in your writeup.

(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?

```
'''
The validation set is used to tune the hyperparameters of the model to improve its accuracy. By using a second data
set, we can prevent overfitting the model to the training dataset when we run the model multiple times to determine
which parameters result in best model performance.

By plotting the validation error and loss alongside the training error and loss, we can determine if the model overfit
to the training data. Overfitting is indicated by a large difference between the training and validation curves in the
error graph.
'''
```

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?)

```
print("small_net parameter tensors:")
for param in small_net.parameters():
    print(param.shape)

print("\nlarge_net parameter tensors:")
for param in large_net.parameters():
    print(param.shape)
```

```
small_net parameter tensors:
torch.Size([5, 3, 3, 3])
torch.Size([5])
torch.Size([1, 245])
torch.Size([1])

large_net parameter tensors:
torch.Size([5, 3, 5, 5])
torch.Size([5])
torch.Size([10, 5, 5, 5])
torch.Size([10])
torch.Size([32, 250])
torch.Size([32])
torch.Size([1, 32])
torch.Size([1])
```

```
"""
There are 5*3*3*3 + 5 + 245*1 + 1 = 386 parameters in small_net and
5*3*5*5 + 5 + 10*5*5*5 + 10 + 32*250 + 32 + 32*1 + 1 = 9705 parameters in large_net
"""
```

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.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`?

```
#default values are: batch_size=64, learning_rate=0.01, num_epochs=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.

```
train_net(small_net, 64, 0.01, 5)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.424625, Train loss: 0.6730796055793762 |Validation err: 0.3725, Validation loss: 0.6565424017608166
Epoch 2: Train err: 0.359375, Train loss: 0.640557909488678 |Validation err: 0.3585, Validation loss: 0.6441871151328087
Epoch 3: Train err: 0.34575, Train loss: 0.6276503925323487 |Validation err: 0.366, Validation loss: 0.6305062938481569
Epoch 4: Train err: 0.33825, Train loss: 0.6188080906867981 |Validation err: 0.3755, Validation loss: 0.6403826735913754
Epoch 5: Train err: 0.334625, Train loss: 0.6099846625328064 |Validation err: 0.3435, Validation loss: 0.6254174262285233
Finished Training
Total time elapsed: 16.14 seconds
```

```
"""
The files written to the disk include:
- train_err.csv
    - contains: training error values for each epoch for the model that was trained
- train_loss.csv
    - contains: training loss values for each epoch for the model that was trained
- val_err.csv
    - contains: validation error values for each epoch for the model that was trained
- val_loss.csv
    - contains: validation loss values for each epoch for the model that was trained
"""
```

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.ap

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

```
print("Training small_net:")
small_net = SmallNet()
train_net(small_net)
```

```
Training small_net:
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.446375, Train loss: 0.6813716759681702 |Validation err: 0.3865, Validation loss: 0.6602997574955225
Epoch 2: Train err: 0.37325, Train loss: 0.649762942314148 |Validation err: 0.3845, Validation loss: 0.657599113881588
Epoch 3: Train err: 0.360125, Train loss: 0.6388907418251037 |Validation err: 0.3485, Validation loss: 0.629109650850296
Epoch 4: Train err: 0.346125, Train loss: 0.6246512727737427 |Validation err: 0.3555, Validation loss: 0.6221790071576834
Epoch 5: Train err: 0.3345, Train loss: 0.6153933835029602 |Validation err: 0.328, Validation loss: 0.6188624110072851
Epoch 6: Train err: 0.3175, Train loss: 0.6037003107070923 |Validation err: 0.339, Validation loss: 0.6092050103470683
Epoch 7: Train err: 0.315875, Train loss: 0.5944590408802033 |Validation err: 0.329, Validation loss: 0.597393348813057
Epoch 8: Train err: 0.308, Train loss: 0.5828994708061218 |Validation err: 0.3075, Validation loss: 0.588386045768857
Epoch 9: Train err: 0.302, Train loss: 0.5804825727939605 |Validation err: 0.311, Validation loss: 0.5844876822084188
Epoch 10: Train err: 0.2995, Train loss: 0.5730265159606933 |Validation err: 0.308, Validation loss: 0.578460194170475
Epoch 11: Train err: 0.287375, Train loss: 0.5632513077259064 |Validation err: 0.313, Validation loss: 0.5817054454237223
Epoch 12: Train err: 0.293375, Train loss: 0.5565420839786529 |Validation err: 0.312, Validation loss: 0.5856076134368777
Epoch 13: Train err: 0.288625, Train loss: 0.5560892834663391 |Validation err: 0.3045, Validation loss: 0.5766977267339826
Epoch 14: Train err: 0.279625, Train loss: 0.5472920253276825 |Validation err: 0.3105, Validation loss: 0.572071947157383
Epoch 15: Train err: 0.28475, Train loss: 0.5480572285652161 |Validation err: 0.304, Validation loss: 0.5625222157686949
Epoch 16: Train err: 0.292625, Train loss: 0.5542038972377777 |Validation err: 0.3105, Validation loss: 0.5773482695221901
Epoch 17: Train err: 0.282875, Train loss: 0.5474525504112243 |Validation err: 0.2975, Validation loss: 0.5682832039892673
Epoch 18: Train err: 0.279875, Train loss: 0.5442741246223449 |Validation err: 0.318, Validation loss: 0.5763898333534598
Epoch 19: Train err: 0.2765, Train loss: 0.5401130385398865 |Validation err: 0.329, Validation loss: 0.6069856593385339
Epoch 20: Train err: 0.27325, Train loss: 0.5384650847911835 |Validation err: 0.297, Validation loss: 0.5782049279659986
Epoch 21: Train err: 0.275, Train loss: 0.5400777981281281 |Validation err: 0.3025, Validation loss: 0.5672480063512921
Epoch 22: Train err: 0.277625, Train loss: 0.5401211383342743 |Validation err: 0.2905, Validation loss: 0.5704732658341527
Epoch 23: Train err: 0.272375, Train loss: 0.5353888559341431 |Validation err: 0.3025, Validation loss: 0.5672092009335756
Epoch 24: Train err: 0.271875, Train loss: 0.5359435317516327 |Validation err: 0.297, Validation loss: 0.587195829488337
Epoch 25: Train err: 0.272875, Train loss: 0.5348421568870545 |Validation err: 0.2985, Validation loss: 0.563989233225584
Epoch 26: Train err: 0.271375, Train loss: 0.5317093801498413 |Validation err: 0.296, Validation loss: 0.5690154964104295
Epoch 27: Train err: 0.269875, Train loss: 0.5299493942260742 |Validation err: 0.301, Validation loss: 0.5788832632824779
Epoch 28: Train err: 0.26925, Train loss: 0.5352410960197449 |Validation err: 0.2995, Validation loss: 0.5660756453871727
Epoch 29: Train err: 0.2715, Train loss: 0.531860348701477 |Validation err: 0.295, Validation loss: 0.5838955584913492
Epoch 30: Train err: 0.272, Train loss: 0.536940132856369 |Validation err: 0.3125, Validation loss: 0.5808273386210203
Finished Training
Total time elapsed: 97.62 seconds
```

```
print("Training large_net:")
large_net = LargeNet()
train_net(large_net)
```

```
Training large_net:
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.44475, Train loss: 0.6900203099250793 |Validation err: 0.4285, Validation loss: 0.680754292756319
Epoch 2: Train err: 0.419125, Train loss: 0.678190191745758 |Validation err: 0.413, Validation loss: 0.6741814017295837
Epoch 3: Train err: 0.39875, Train loss: 0.6658317875862122 |Validation err: 0.391, Validation loss: 0.6517764702439308
Epoch 4: Train err: 0.374125, Train loss: 0.6491540780067444 |Validation err: 0.4095, Validation loss: 0.6662181690335274
Epoch 5: Train err: 0.35675, Train loss: 0.6333453297615051 |Validation err: 0.353, Validation loss: 0.6291991733014584
Epoch 6: Train err: 0.33925, Train loss: 0.6163788948059082 |Validation err: 0.344, Validation loss: 0.6148867644369602
Epoch 7: Train err: 0.327875, Train loss: 0.6005767168998718 |Validation err: 0.3315, Validation loss: 0.6076090820133686
Epoch 8: Train err: 0.311875, Train loss: 0.5835636842250824 |Validation err: 0.322, Validation loss: 0.5945568196475506
Epoch 9: Train err: 0.30675, Train loss: 0.5753059720993042 |Validation err: 0.321, Validation loss: 0.5969209987670183
Epoch 10: Train err: 0.292, Train loss: 0.5619129114151001 |Validation err: 0.311, Validation loss: 0.5900639891624451
Epoch 11: Train err: 0.28625, Train loss: 0.5491175475120544 |Validation err: 0.3125, Validation loss: 0.612250036559999
Epoch 12: Train err: 0.278625, Train loss: 0.5407402107715606 |Validation err: 0.302, Validation loss: 0.5912075499072671
Epoch 13: Train err: 0.272375, Train loss: 0.5305765857696533 |Validation err: 0.298, Validation loss: 0.58992516156286
Epoch 14: Train err: 0.26575, Train loss: 0.5176558227539062 |Validation err: 0.303, Validation loss: 0.5985351065173745
Epoch 15: Train err: 0.255, Train loss: 0.5113149104118347 |Validation err: 0.3025, Validation loss: 0.6041473727673292
Epoch 16: Train err: 0.24925, Train loss: 0.5016681816577911 |Validation err: 0.301, Validation loss: 0.5914413323625922
Epoch 17: Train err: 0.245375, Train loss: 0.4960096251964569 |Validation err: 0.302, Validation loss: 0.5816831914708018
Epoch 18: Train err: 0.241125, Train loss: 0.48287230825424193 |Validation err: 0.2975, Validation loss: 0.6047292938455939
Epoch 19: Train err: 0.239, Train loss: 0.47653640270233155 |Validation err: 0.3175, Validation loss: 0.6174056949093938
Epoch 20: Train err: 0.23175, Train loss: 0.468037611246109 |Validation err: 0.301, Validation loss: 0.5940169263631105
Epoch 21: Train err: 0.2215, Train loss: 0.4554054045677185 |Validation err: 0.282, Validation loss: 0.6005946267396212
Epoch 22: Train err: 0.216375, Train loss: 0.44967878103256226 |Validation err: 0.295, Validation loss: 0.627867478877306
Epoch 23: Train err: 0.21375, Train loss: 0.4401419384479523 |Validation err: 0.298, Validation loss: 0.6050756443291903
Epoch 24: Train err: 0.201875, Train loss: 0.42461662316322324 |Validation err: 0.307, Validation loss: 0.6475039189681411
Epoch 25: Train err: 0.2, Train loss: 0.412550235748291 |Validation err: 0.3005, Validation loss: 0.6428515408188105
Epoch 26: Train err: 0.18825, Train loss: 0.3993633940219879 |Validation err: 0.297, Validation loss: 0.6563977729529142
Epoch 27: Train err: 0.177, Train loss: 0.3825074262619019 |Validation err: 0.296, Validation loss: 0.6500479569658637
Epoch 28: Train err: 0.173875, Train loss: 0.3786721342802048 |Validation err: 0.3, Validation loss: 0.6525309970602393
Epoch 29: Train err: 0.167, Train loss: 0.3608296457529068 |Validation err: 0.3335, Validation loss: 0.7983271488919854
Epoch 30: Train err: 0.158625, Train loss: 0.3484336166381836 |Validation err: 0.2965, Validation loss: 0.7060811650007963
Finished Training
Total time elapsed: 115.25 seconds
```

```
"""
- large_net takes longer to train
- large_net has more parameters (and convolution layers, and fully-connected layers) than small_net

```

```
    - there are more calculations involved in the forward pass part of the model
"""
```

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.

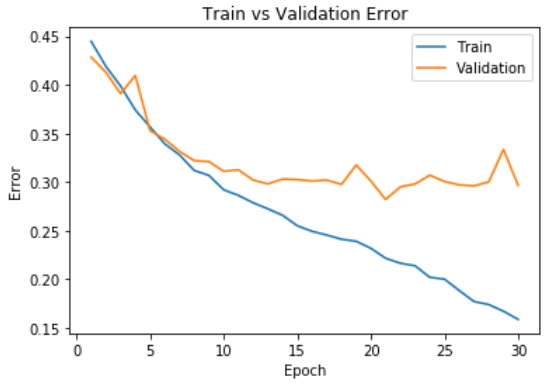
Do this for both the small network and the large network. Include both plots in your writeup.

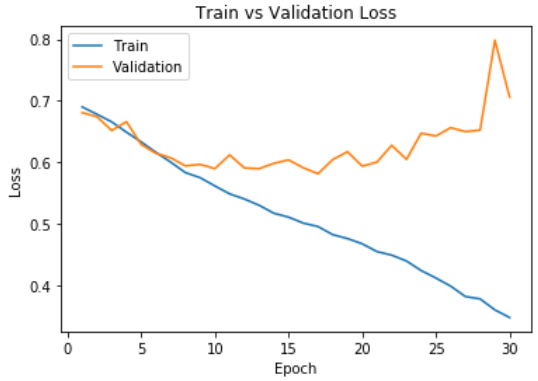
```
print("Graphs for small_net: ")
model_path_small = get_model_name("small", 64, 0.01, 29)
plot_training_curve(model_path_small)
```

```
Graphs for small_net:
```


```
print("Graphs for large_net: ")
model_path_large = get_model_name("large", 64, 0.01, 29)
plot_training_curve(model_path_large)
```

```
Graphs for large_net:
```





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.

```
'''
small_net
In the error graph, both the training error and the validation error flatten out as the number of epochs increases.
As expected, the training error is lower than the validation error because the model has seen the training data more
frequently than the validation data. Overall, this suggests that these parameters are a better fit than large_net as
it does not have as much overfitting.

In the loss graph, both the training loss and the validation loss flatten out as the number of epochs increases,
similar to the error graph, again suggesting that these parameters result in a better fit than large_net.

large_net
In the error graph, the training error is monotonically decreasing as the number of epochs increases, whereas the
validation error flattens out after ~15 epochs. This indicates that the model experienced overfitting. The error
in the training data is minimized because the model incorporated the patterns present in the training data into its
network. However, because these patterns may not be present in the validation data, the model has a greater error.

In the loss graph, the training loss is monotonically decreasing as the number of epochs increases, whereas the
validation loss decreases when 10 < number of epochs < 20, and increases for number of epochs > 20. This is also
evidence that the model experienced overfitting.
'''
```

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.

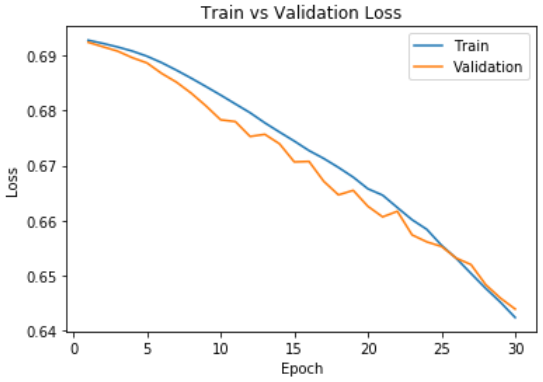
```
'''
```

```
- the new model takes more time to train as the previous model
- in both the error and loss graphs, the training and validation curves are very close together
    - in the error graph, the validation error is less than the training error when less than 15 epochs are used
    - after this point, the validation error is greater than the training error, except when 19 epochs are used
    - in the loss graph, the validation loss is less than the training loss until the number of epochs exceeds 25
- these curves suggest that the model is well-fit to the data for these parameter choices and did not overfit
- by lowering the learning rate, we improved the model's fit to the data, but the accuracy in the training data decreased
'''
```

```
# 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, 64, 0.001, 30)
```

```
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: 115.98 seconds
```

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



Part (b) - 3pt

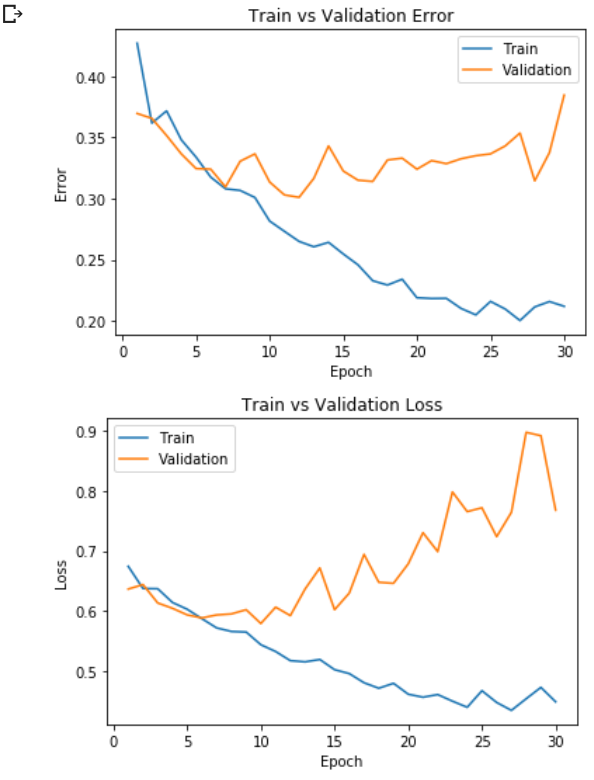
Train `large_net` with all default parameters, except set `learning_rate=0.1`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *increasing* the learning rate.

```
'''
- the new model takes more time to train than the original model
- in both the error and loss graphs, the curves diverge after 6 epochs
    - the validation error becomes higher than the training error as the number of epochs increases
    - the validation loss continues to grow even when the training loss is decreasing
- this divergence suggests that the model overfit to the training data
    - this is not desireable in the model
    - so setting learning_rate = 0.1 is not a good parameter choice
- when the learning rate is 0.1, the model experiences overfitting but the training data accuracy is better than part (a)
'''
```

```
large_net = LargeNet()
train_net(large_net, 64, 0.1, 30)
```

```
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: 114.19 seconds
```

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



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.

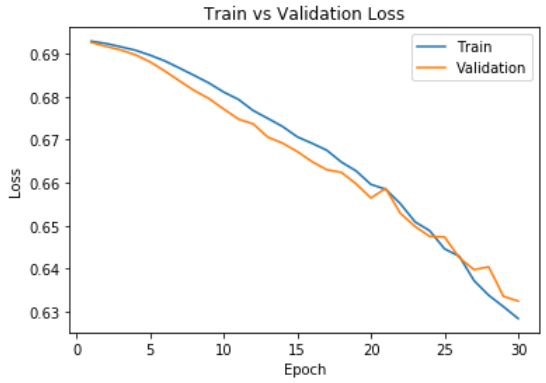
```
'''
```

```
- the new model takes less time to train than the original model
- this parameter change produces graphs similar to part (a) where the training curve and validation curve overlap frequently
    - the range of error in part (c) is the same as part (a)
    - the range of loss in part (c) is slightly less than part (a)
    - in the error graph, the validation error is less than the training error when the number of epochs used is less than 15
    - in the loss graph, the validation loss is lower than the training loss when the number of epochs is less than 25
- these curves suggest that the model is well-fit to the data for these parameter choices
- by increasing the batch_size, the model's fit improves but its accuracy on the training data decreases
'''
```

```
large_net = LargeNet()
train_net(large_net, 512, 0.01, 30)
```

```
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: 96.26 seconds
```

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



Part (d) - 3pt

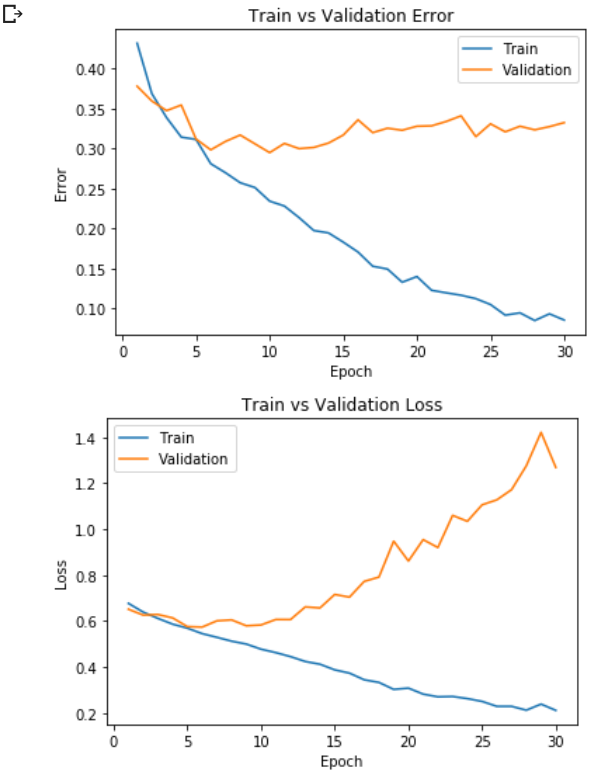
Train `large_net` with all default parameters, including with `learning_rate=0.01`. Now, set `batch_size=16`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *decreasing* the batch size.

```
'''
- this model takes longer to train than the original model
- this model exhibits a similar curve divergence as was found in part (b)
    - the divergence in these graphs also occurs around 5 epochs
    - the validation error and loss are less than the training error and loss when 2 epochs are used
- compared to the graphs in part (b), this model diverges after less epochs
    - this means this model overfits more than part (b)
- this model has lower errors than part (a), (b), and (c)
- when the batch_size is decreased, the model overfit to the training data but the highest accuracy was achieved
'''
```

```
large_net = LargeNet()
train_net(large_net, 16, 0.01, 30)
```

```
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
Epoch 8: Train err: 0.257375, Train loss: 0.5129605046510697 |Validation err: 0.317, Validation loss: 0.6054974246025085
Epoch 9: Train err: 0.251375, Train loss: 0.5003467800319195 |Validation err: 0.306, Validation loss: 0.5799579107761383
Epoch 10: Train err: 0.234375, Train loss: 0.4781089634001255 |Validation err: 0.295, Validation loss: 0.5835002071857452
Epoch 11: Train err: 0.22825, Train loss: 0.46318685048818586 |Validation err: 0.3065, Validation loss: 0.6072176740169525
Epoch 12: Train err: 0.21375, Train loss: 0.4457400677502155 |Validation err: 0.3, Validation loss: 0.606988734960556
Epoch 13: Train err: 0.197625, Train loss: 0.42451206052303314 |Validation err: 0.3015, Validation loss: 0.6619625856876373
Epoch 14: Train err: 0.194625, Train loss: 0.41265408125519754 |Validation err: 0.307, Validation loss: 0.6576042010784149
Epoch 15: Train err: 0.183125, Train loss: 0.38814702521264555 |Validation err: 0.317, Validation loss: 0.716176066160202
Epoch 16: Train err: 0.17075, Train loss: 0.37398081965744495 |Validation err: 0.336, Validation loss: 0.704552215218544
Epoch 17: Train err: 0.153, Train loss: 0.3449616933763027 |Validation err: 0.32, Validation loss: 0.7729343641996383
Epoch 18: Train err: 0.1495, Train loss: 0.3334929691925645 |Validation err: 0.3255, Validation loss: 0.7917270390987396
Epoch 19: Train err: 0.133, Train loss: 0.3032665306478739 |Validation err: 0.323, Validation loss: 0.9481484520435334
Epoch 20: Train err: 0.140125, Train loss: 0.3086603755950928 |Validation err: 0.328, Validation loss: 0.8615521432161332
Epoch 21: Train err: 0.122875, Train loss: 0.2826835075467825 |Validation err: 0.3285, Validation loss: 0.9546727517843246
Epoch 22: Train err: 0.11975, Train loss: 0.27087254472449424 |Validation err: 0.334, Validation loss: 0.9202701450586319
Epoch 23: Train err: 0.11675, Train loss: 0.2722733011692762 |Validation err: 0.341, Validation loss: 1.0601593099832536
Epoch 24: Train err: 0.1125, Train loss: 0.2631180996969342 |Validation err: 0.315, Validation loss: 1.0344713234901428
Epoch 25: Train err: 0.105125, Train loss: 0.25092605347186325 |Validation err: 0.331, Validation loss: 1.1061395760774613
Epoch 26: Train err: 0.091875, Train loss: 0.22940256588160993 |Validation err: 0.321, Validation loss: 1.127762097477913
Epoch 27: Train err: 0.09475, Train loss: 0.22944023758545518 |Validation err: 0.328, Validation loss: 1.1720888905525209
Epoch 28: Train err: 0.085125, Train loss: 0.21271794915758074 |Validation err: 0.3235, Validation loss: 1.275900668501854
Epoch 29: Train err: 0.0935, Train loss: 0.23916544995456934 |Validation err: 0.3275, Validation loss: 1.4214059482812882
Epoch 30: Train err: 0.08575, Train loss: 0.21191889957711102 |Validation err: 0.3325, Validation loss: 1.2688533749580384
Finished Training
Total time elapsed: 167.25 seconds
```

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



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.

```
'''
The parameters used in part (3a) produced the most well-fit model across all 4 variations in hyperparameters
    - little gap between training and validation errors and losses
    - relatively consistent values (values don't decrease or increase) as the number of epochs increased
    - parameters from part (3a): network = large_net, batch_size = 64, learning_rate = 0.001
The parameters used in part (3d) produced the model that had smallest training error value across all epochs used, but this
model showed a lot of overfitting.
    - large difference between the training and validation curves in the error plot, and the validation loss increases as
    the number of epochs increases in the loss plot
    - parameters from part (3c): network = large_net, batch_size = 16, learning_rate = 0.01

So there should exist parameters 16 <= batch_size <= 64, 0.001 <= learning_rate <= 0.01 where both the error and loss are
minimized for the training and validation datasets and the model is well-fitted. Start by testing on large_net since that
was the network used in part 3.

Try train_net(large_net, 32, 0.005, 30)
'''
```

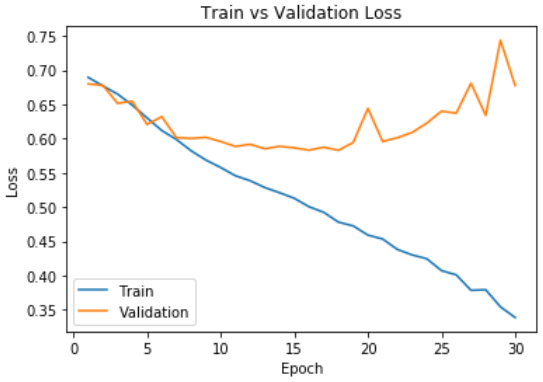
Part (b) - 1pt

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

```
large_net = LargeNet()
train_net(large_net, 32, 0.005, 30)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.451125, Train loss: 0.6895514280796051 |Validation err: 0.424, Validation loss: 0.679968071362329
Epoch 2: Train err: 0.41475, Train loss: 0.6771168055534362 |Validation err: 0.426, Validation loss: 0.6779351764255099
Epoch 3: Train err: 0.398375, Train loss: 0.6650897436141968 |Validation err: 0.3985, Validation loss: 0.6516057640787155
Epoch 4: Train err: 0.3795, Train loss: 0.6489185693264008 |Validation err: 0.393, Validation loss: 0.6546722092325725
Epoch 5: Train err: 0.356875, Train loss: 0.6303374912738801 |Validation err: 0.3405, Validation loss: 0.6206440679610722
Epoch 6: Train err: 0.33725, Train loss: 0.6118779233694076 |Validation err: 0.3475, Validation loss: 0.6321050132077838
Epoch 7: Train err: 0.324125, Train loss: 0.5986748019456863 |Validation err: 0.317, Validation loss: 0.601392797534428
Epoch 8: Train err: 0.31675, Train loss: 0.5821823552846909 |Validation err: 0.335, Validation loss: 0.6003369372042399
Epoch 9: Train err: 0.298375, Train loss: 0.5686504502296448 |Validation err: 0.3315, Validation loss: 0.601876895106028
Epoch 10: Train err: 0.294375, Train loss: 0.5578343842029572 |Validation err: 0.3055, Validation loss: 0.5958308534962791
Epoch 11: Train err: 0.284375, Train loss: 0.5458236672878265 |Validation err: 0.3005, Validation loss: 0.5886339318184626
Epoch 12: Train err: 0.279125, Train loss: 0.5384978953599929 |Validation err: 0.305, Validation loss: 0.5916959950848232
Epoch 13: Train err: 0.271375, Train loss: 0.5285226913690567 |Validation err: 0.298, Validation loss: 0.5851935900392986
Epoch 14: Train err: 0.26625, Train loss: 0.520972165465355 |Validation err: 0.297, Validation loss: 0.5887271080698285
Epoch 15: Train err: 0.256, Train loss: 0.5128746335506439 |Validation err: 0.292, Validation loss: 0.5865116431599572
Epoch 16: Train err: 0.250375, Train loss: 0.5004664026498794 |Validation err: 0.296, Validation loss: 0.583122445004327
Epoch 17: Train err: 0.2495, Train loss: 0.49232296389341357 |Validation err: 0.295, Validation loss: 0.5874926750622098
Epoch 18: Train err: 0.237375, Train loss: 0.4779584574699402 |Validation err: 0.289, Validation loss: 0.5829228725698259
Epoch 19: Train err: 0.230625, Train loss: 0.4725129836201668 |Validation err: 0.2995, Validation loss: 0.5942703934888991
Epoch 20: Train err: 0.2225, Train loss: 0.4590291562080383 |Validation err: 0.3045, Validation loss: 0.6441603231997717
Epoch 21: Train err: 0.215875, Train loss: 0.45334128630161286 |Validation err: 0.29, Validation loss: 0.5958448246357932
Epoch 22: Train err: 0.208, Train loss: 0.4381052249670029 |Validation err: 0.3, Validation loss: 0.6013416756713201
Epoch 23: Train err: 0.20575, Train loss: 0.4301115469932556 |Validation err: 0.3055, Validation loss: 0.6089236968093448
Epoch 24: Train err: 0.197875, Train loss: 0.4244694077372551 |Validation err: 0.3045, Validation loss: 0.6226789080907428
Epoch 25: Train err: 0.18975, Train loss: 0.40711965477466583 |Validation err: 0.282, Validation loss: 0.6400856541262733
Epoch 26: Train err: 0.184625, Train loss: 0.4008987776041031 |Validation err: 0.2995, Validation loss: 0.6371875405311584
Epoch 27: Train err: 0.169625, Train loss: 0.3784003840088844 |Validation err: 0.3045, Validation loss: 0.6808964222196549
Epoch 28: Train err: 0.175, Train loss: 0.3791908652186394 |Validation err: 0.304, Validation loss: 0.6340301883599114
Epoch 29: Train err: 0.1595, Train loss: 0.3541890382170677 |Validation err: 0.3065, Validation loss: 0.7437764760993776
Epoch 30: Train err: 0.15075, Train loss: 0.3385722677409649 |Validation err: 0.2945, Validation loss: 0.6775304362887428
Finished Training
Total time elapsed: 135.60 seconds
```

```
model_path = get_model_name("large", 32, 0.005, 29)
plot_training_curve(model_path)
```



Part (c) - 2pt

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

```
"""
The parameters in part (4b) caused the model to overfit.
    - the training error is decreasing as the number of epochs used increases, but the validation error flattens out after
    10 epochs
    - the validation loss keeps increasing while the training loss is decreasing
    - this could have been predicted as both the learning_rate and batch_size were low, allowing for many passes over the
    training data

To prevent overfitting, try changing the net parameter to small_net
    - harder to overfit on small_net since it has less layers and parameters to train on than large_net

Try train_net(small_net, 32, 0.005, 30)
"""
```

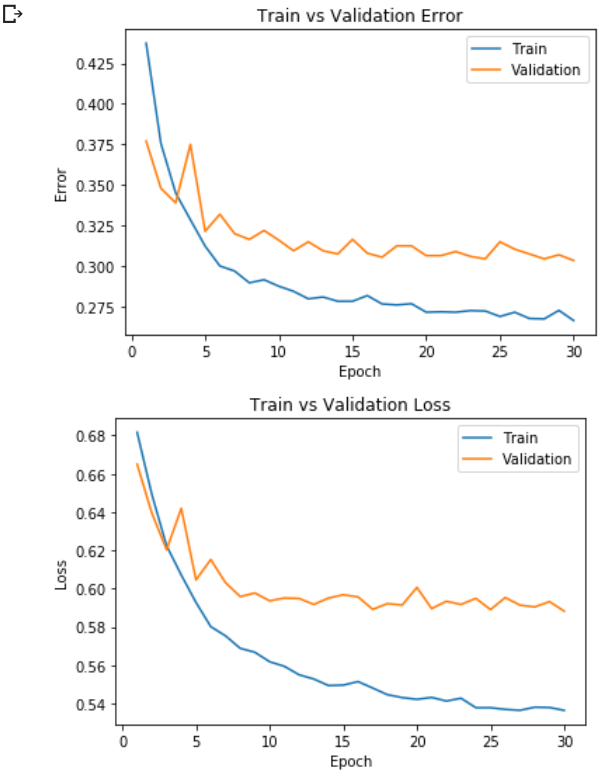
Part (d) - 1pt

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

```
small_net = SmallNet()
train_net(small_net, 32, 0.005, 30)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.437375, Train loss: 0.6816291599273682 |Validation err: 0.377, Validation loss: 0.66497712665134
Epoch 2: Train err: 0.375625, Train loss: 0.6492082698345184 |Validation err: 0.348, Validation loss: 0.6394086016549004
Epoch 3: Train err: 0.345125, Train loss: 0.622421914100647 |Validation err: 0.339, Validation loss: 0.6203346966751038
Epoch 4: Train err: 0.328625, Train loss: 0.6070742863416672 |Validation err: 0.375, Validation loss: 0.6419702210123577
Epoch 5: Train err: 0.312375, Train loss: 0.592782075881958 |Validation err: 0.3215, Validation loss: 0.6045610242419772
Epoch 6: Train err: 0.300125, Train loss: 0.5801633304357529 |Validation err: 0.332, Validation loss: 0.6151923499410115
Epoch 7: Train err: 0.297, Train loss: 0.5753675071001053 |Validation err: 0.32, Validation loss: 0.603155696675891
Epoch 8: Train err: 0.28975, Train loss: 0.5688398077487945 |Validation err: 0.3165, Validation loss: 0.5958637207273453
Epoch 9: Train err: 0.291625, Train loss: 0.5667602540254593 |Validation err: 0.322, Validation loss: 0.5977269950367156
Epoch 10: Train err: 0.287625, Train loss: 0.5618348401784897 |Validation err: 0.316, Validation loss: 0.5936325287062024
Epoch 11: Train err: 0.2845, Train loss: 0.5594247744083405 |Validation err: 0.3095, Validation loss: 0.5950875509352911
Epoch 12: Train err: 0.279875, Train loss: 0.5550178567171097 |Validation err: 0.315, Validation loss: 0.5948523306657397
Epoch 13: Train err: 0.281, Train loss: 0.5528003664016724 |Validation err: 0.3095, Validation loss: 0.5917623941860501
Epoch 14: Train err: 0.278375, Train loss: 0.5493912705183029 |Validation err: 0.3075, Validation loss: 0.595080606521122
Epoch 15: Train err: 0.278375, Train loss: 0.5496201508045196 |Validation err: 0.3165, Validation loss: 0.5967953545706612
Epoch 16: Train err: 0.281875, Train loss: 0.5514670269489288 |Validation err: 0.308, Validation loss: 0.5956861329457116
Epoch 17: Train err: 0.27675, Train loss: 0.5480822356939316 |Validation err: 0.3055, Validation loss: 0.5892035402948894
Epoch 18: Train err: 0.276125, Train loss: 0.54461756503582 |Validation err: 0.3125, Validation loss: 0.592187087687235
Epoch 19: Train err: 0.276875, Train loss: 0.5431125967502594 |Validation err: 0.3125, Validation loss: 0.5914331797569518
Epoch 20: Train err: 0.271625, Train loss: 0.5421927300691605 |Validation err: 0.3065, Validation loss: 0.6006332967016432
Epoch 21: Train err: 0.271875, Train loss: 0.5431939759254456 |Validation err: 0.3065, Validation loss: 0.5895962629999433
Epoch 22: Train err: 0.271625, Train loss: 0.5413144061565399 |Validation err: 0.309, Validation loss: 0.5933961664873456
Epoch 23: Train err: 0.272625, Train loss: 0.5427670339345932 |Validation err: 0.306, Validation loss: 0.5917511828361995
Epoch 24: Train err: 0.272375, Train loss: 0.537835864663124 |Validation err: 0.3045, Validation loss: 0.5949300702602144
Epoch 25: Train err: 0.269, Train loss: 0.5378221975564956 |Validation err: 0.315, Validation loss: 0.5890734829599895
Epoch 26: Train err: 0.271625, Train loss: 0.5369531993865967 |Validation err: 0.3105, Validation loss: 0.5953453873831128
Epoch 27: Train err: 0.26775, Train loss: 0.5364651715755463 |Validation err: 0.3075, Validation loss: 0.5913815621345763
Epoch 28: Train err: 0.2675, Train loss: 0.5381050134897232 |Validation err: 0.3045, Validation loss: 0.5904482588881538
Epoch 29: Train err: 0.27275, Train loss: 0.5379434266090393 |Validation err: 0.307, Validation loss: 0.593220715484922
Epoch 30: Train err: 0.2665, Train loss: 0.5364364982843399 |Validation err: 0.3035, Validation loss: 0.5882868468761444
Finished Training
Total time elapsed: 114.77 seconds
```

```
model_path = get_model_name("small", 32, 0.005, 29)
plot_training_curve(model_path)
```



Part 5. 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`.

```
net = SmallNet()
model_path = get_model_name(net.name, batch_size=32, learning_rate=0.005, epoch=29)
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).

```
"""
- small_net was chosen because it is less likely to overfit compared to large_net when the learning_rate is low
- batch_size was set to 32 because it is between the batch_size used in part (3a) (where the loss function is
optimized) and (3c) (where the error had the smallest values)
- learning_rate was set to 0.005 because it produced the smallest range of errors
- the number of epochs was set to 30 to keep it consistent with part 3 where the model was trained with
different hyperparameters
"""
```

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=64)

error, loss = evaluate(net, test_loader, nn.BCEWithLogitsLoss())
print("\nThe test classification error is {}".format(error))
#print("The test classification loss is {}".format(loss))
```

```
Files already downloaded and verified
Files already downloaded and verified

The test classification error is 0.289
```

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.

```
error, loss = evaluate(net, val_loader, nn.BCEWithLogitsLoss())
print("The validation classification error is {}".format(error))
#print("The validation classification loss is {}".format(loss))

#error, loss = evaluate(net, train_loader, nn.BCEWithLogitsLoss())
#print("\nThe training classification error is {}".format(error))
#print("The training classification loss is {}".format(loss))
```

```
The validation classification error is 0.3035
```

```
"""
The test classification error is less than the validation classification error.

It is expected that the test error is greater than the validation error because the model's parameters were tuned to
the validation dataset, so the model should fit the validation data better.
"""
```

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?

```
"""
We use the test data at the end to judge the performance of the model on classifying accurate,
real-world data, after we have trained the model using the training data and chosen the best
hyperparameters using the validation data.

If the test data is used frequently, then the model will overfit to the test data. This means that
the error in test classification will be low, but not because the model does a good job at
classifying the data samples. Rather, the model learns the patterns in the test data to artificially
inflate the accuracy.
"""
```

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?

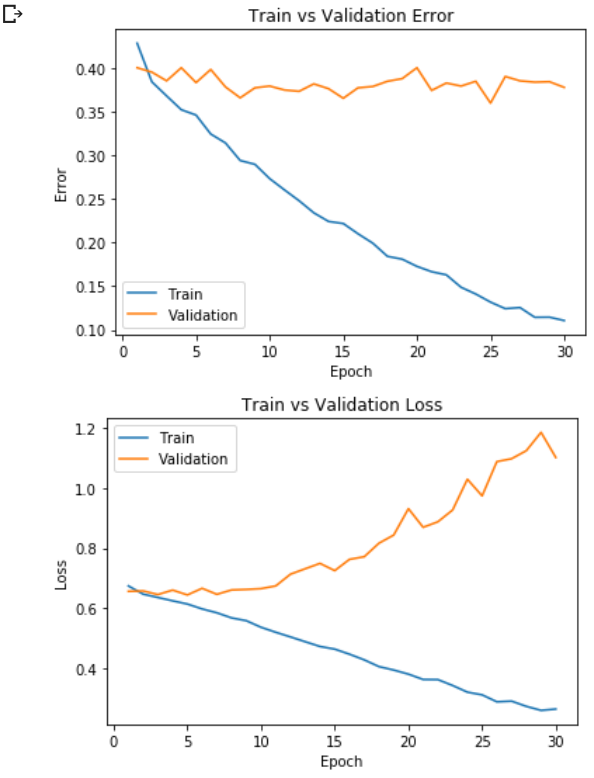
```
# ANN from Lab 1
# define a 2-layer artificial neural network
class Pigeon(nn.Module):
    def __init__(self):
        super(Pigeon, self).__init__()
        self.name = "pigeon"
        self.fc1 = nn.Linear(3 * 32 * 32, 30)
        self.fc3 = nn.Linear(30, 30)
        self.fc2 = nn.Linear(30, 1)

    def forward(self, x):
        x = x.view(-1, 3 * 32 * 32)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc3(x))
        x = self.fc2(x)
        x = x.squeeze(1)
        return x
pigeon = Pigeon()
```

```
pigeon = Pigeon()
train_net(pigeon, 32, 0.005, 30)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.428625, Train loss: 0.6747044415473938 |Validation err: 0.4005, Validation loss: 0.6568544194811866
Epoch 2: Train err: 0.384375, Train loss: 0.647522987127304 |Validation err: 0.3955, Validation loss: 0.6581749480868143
Epoch 3: Train err: 0.36825, Train loss: 0.6365154120922089 |Validation err: 0.3855, Validation loss: 0.646224458066244
Epoch 4: Train err: 0.352375, Train loss: 0.6254205241203308 |Validation err: 0.4005, Validation loss: 0.6609205471144782
Epoch 5: Train err: 0.34625, Train loss: 0.6147860208749771 |Validation err: 0.3835, Validation loss: 0.6449282964070638
Epoch 6: Train err: 0.3245, Train loss: 0.5985808489322663 |Validation err: 0.3985, Validation loss: 0.6668390423532516
Epoch 7: Train err: 0.31425, Train loss: 0.5857539578676224 |Validation err: 0.3785, Validation loss: 0.6469652047232975
Epoch 8: Train err: 0.294125, Train loss: 0.5688342607021332 |Validation err: 0.366, Validation loss: 0.6613867150412666
Epoch 9: Train err: 0.28975, Train loss: 0.559494670510292 |Validation err: 0.3775, Validation loss: 0.6631318742320651
Epoch 10: Train err: 0.273375, Train loss: 0.5377255365848541 |Validation err: 0.3795, Validation loss: 0.6657828083114018
Epoch 11: Train err: 0.2605, Train loss: 0.5211802898645401 |Validation err: 0.375, Validation loss: 0.6748781308295235
Epoch 12: Train err: 0.248, Train loss: 0.5056772075891495 |Validation err: 0.3735, Validation loss: 0.7137904588192229
Epoch 13: Train err: 0.234125, Train loss: 0.48980536532402036 |Validation err: 0.382, Validation loss: 0.7317158286533658
Epoch 14: Train err: 0.22425, Train loss: 0.47410700809955597 |Validation err: 0.3765, Validation loss: 0.7495038244459364
Epoch 15: Train err: 0.221875, Train loss: 0.4653646719455719 |Validation err: 0.3655, Validation loss: 0.7254905095176091
Epoch 16: Train err: 0.210125, Train loss: 0.4485363901853561 |Validation err: 0.3775, Validation loss: 0.7628295837886749
Epoch 17: Train err: 0.199375, Train loss: 0.4297655017375946 |Validation err: 0.379, Validation loss: 0.7715125684700315
Epoch 18: Train err: 0.18425, Train loss: 0.4074015765786171 |Validation err: 0.385, Validation loss: 0.81634050560376
Epoch 19: Train err: 0.181, Train loss: 0.39601739156246185 |Validation err: 0.388, Validation loss: 0.843327289062833
Epoch 20: Train err: 0.17275, Train loss: 0.38245126795768736 |Validation err: 0.4005, Validation loss: 0.9308577624578325
Epoch 21: Train err: 0.166625, Train loss: 0.36471832323074344 |Validation err: 0.3745, Validation loss: 0.8690491105828967
Epoch 22: Train err: 0.163, Train loss: 0.3642842833995819 |Validation err: 0.383, Validation loss: 0.8874295368081048
Epoch 23: Train err: 0.14875, Train loss: 0.3447896727323532 |Validation err: 0.3795, Validation loss: 0.9264571784980713
Epoch 24: Train err: 0.141, Train loss: 0.32274877232313154 |Validation err: 0.385, Validation loss: 1.0283710171305944
Epoch 25: Train err: 0.13175, Train loss: 0.31378499779105185 |Validation err: 0.36, Validation loss: 0.9731962406446063
Epoch 26: Train err: 0.12425, Train loss: 0.29078270846605303 |Validation err: 0.3905, Validation loss: 1.0868386806003631
Epoch 27: Train err: 0.1255, Train loss: 0.29297015303373336 |Validation err: 0.3855, Validation loss: 1.0963344971338909
Epoch 28: Train err: 0.114375, Train loss: 0.27564783856272695 |Validation err: 0.384, Validation loss: 1.123046198534587
Epoch 29: Train err: 0.1145, Train loss: 0.26234547287225723 |Validation err: 0.3845, Validation loss: 1.1837979200340452
Epoch 30: Train err: 0.110625, Train loss: 0.26666715782880784 |Validation err: 0.378, Validation loss: 1.1004831710505107
Finished Training
Total time elapsed: 87.02 seconds
```

```
model_path = get_model_name("pigeon", 32, 0.005, 30-1)
plot_training_curve(model_path)
```




```
net = Pigeon()
model_path = get_model_name(net.name, batch_size=32, learning_rate=0.005, epoch=30-1)
state = torch.load(model_path)
net.load_state_dict(state)

error, loss = evaluate(net, test_loader, nn.BCEWithLogitsLoss())
print("The test classification error is {}".format(error))
```

```

error, loss = evaluate(net, val_loader, nn.BCEWithLogitsLoss())
print("The validation classification error is {}".format(error))
```

```
The test classification error is 0.3735
The validation classification error is 0.378
```

```
"""
With the same batch_size, learning_rate, number of epochs, and number of layers as the model chosen in part 4, the ANN
has a higher test classification error and is more overfitted to the training data. This indicates that the ANN performs
worse than the CNN on this dataset.

- the CNN is better at gathering local information in an image, whereas the ANN gathers all the information in an image
    - the CNN later combines all local information by taking, for example, the maximum or average of all local information
- the CNN contains less fully-connected layers than the ANN to reduce computation time and complexity
"""
```