Lab 2: Cats vs Dogs

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/1Q5nNzFjZM19GyP8BsQlJjpmbKkiRlHkO?usp=sharing

```
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):
    """ Loads images of cats and dogs, splits the data into training, validation
    and testing datasets. Returns data loaders for the three preprocessed dataset

    Args:
        target_classes: A list of strings denoting the name of the desired
                        classes. Should be a subset of the argument 'classes'
        batch_size: A int representing the number of samples per batch

    Returns:
        train_loader: iterable training dataset organized according to batch size
        val_loader: iterable validation dataset organized according to batch size
        test_loader: iterable testing dataset organized according to batch size
        classes: A list of strings denoting the name of each class
    """

    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))])
    # Load CIFAR10 training data
    trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                            download=True, transform=transform)
    # Get the list of indices to sample from
    relevant_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_indices)
```

```
    split = int(len(relevant_indices) * 0.8) #split at 80%

    # split into training and validation indices
    relevant_train_indices, relevant_val_indices = relevant_indices[:split], rele
    train_sampler = SubsetRandomSampler(relevant_train_indices)
    train_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                               num_workers=1, sampler=train_sampl
    val_sampler = SubsetRandomSampler(relevant_val_indices)
    val_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                               num_workers=1, sampler=val_sampler)
    # Load CIFAR10 testing data
    testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                           download=True, transform=transform)
    # Get the list of indices to sample from
    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.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
```

```
Files already downloaded and verified
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
```

Show hidden output

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?

```
'''Because the CIFAR-10 dataset has 50,000 training images over all classes, and
10 total classes, we get'''
total_training_images_per_class = 50000/10 #5000 training images per class
'''Out of these training images, our model splits 80% of the training data for in
the remaining 20% for validation. Therefore:'''
training_model_cats_dogs = 2 * total_training_images_per_class * 0.8 #8000 traini
validation_model_cats_dogs = 2* total_training_images_per_class - training_model_
```

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?

Double-click (or enter) to edit

```
'''We need a validation set when training our model to tune and control the model
trained, which helps in reducing bias in the model and show how well the model is
adapting. If we judge the performance of models using training set error instead 
set error, we would have a greatly biased evaluation of the model making it seem 
than it actually is because the training set is the data that the model uses to l
so the error on it will be much lower than the error when it is exposed to a bran
different data points, but the same TYPE of data.'''
```

```
'We need a validation set when training our model to tune and control the mod
el after it has been\ntrained, which helps in reducing bias in the model and 
show how well the model is learning and\nadapting. If we judge the performanc
e of models using training set error instead of the validation\nset error, we
```

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?)

```
for param in small_net.parameters():
    print(param.shape)
print("-")
for param in large_net.parameters():
    print(param.shape)
```

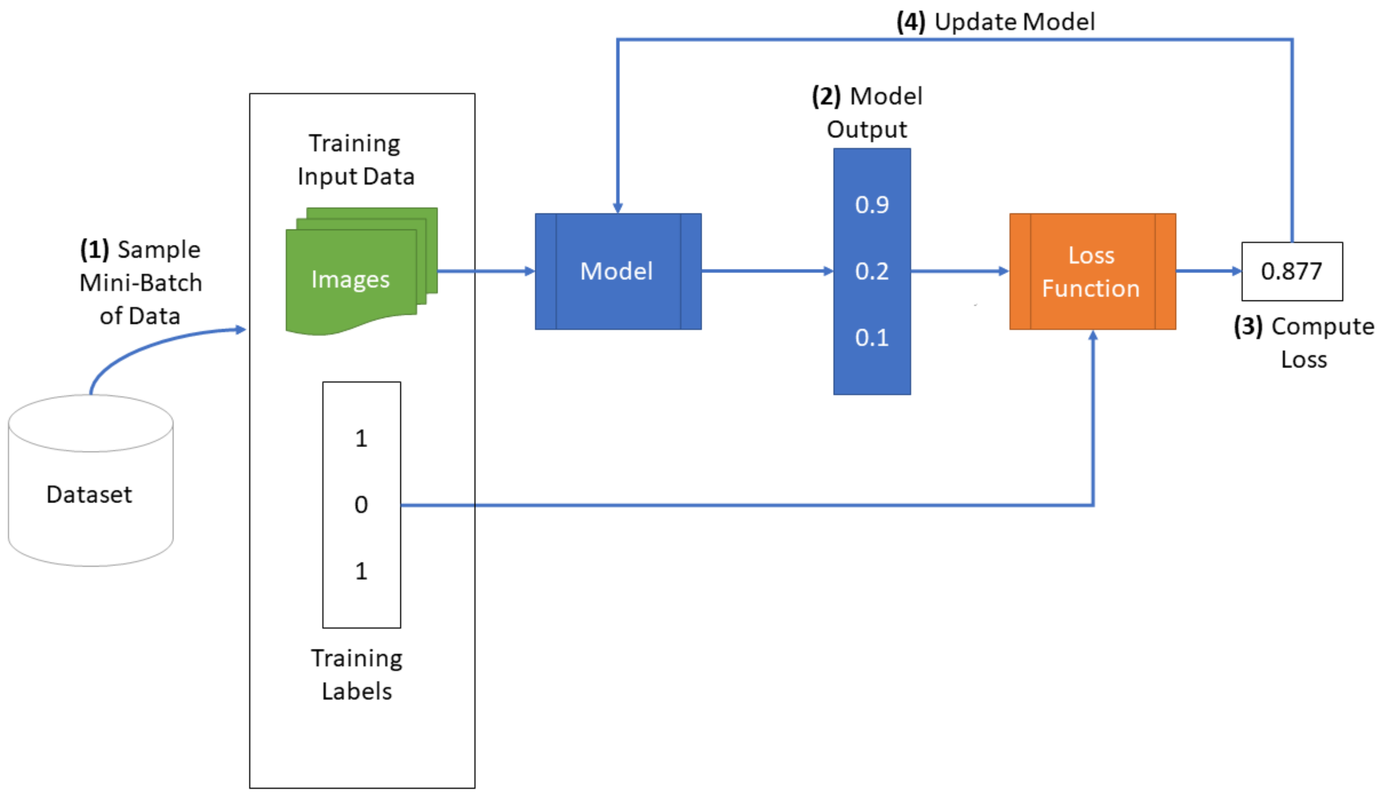
```
torch.Size([5, 3, 3, 3])
torch.Size([5])
torch.Size([1, 245])
torch.Size([1])
-
torch.Size([5, 3, 5, 5])
torch.Size([5])
torch.Size([10, 5, 5, 5])
torch.Size([10])
torch.Size([32, 250])
torch.Size([32])
torch.Size([1, 32])
torch.Size([1])
```

```
print("The total number of parameters in small_net is " + str(5*3*3*3 + 5 + 1*245
print("The total number of parameters in large_net is " + str(5*3*5*5 + 5 + 10*5*
```

```
The total number of parameters in small_net is 386
The total number of parameters in large_net is 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.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:
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(net = small_net, num_epochs = 5)

'''Files written to disk:
model_small_bs64_lr0.01_epoch0 - checkpoint for epoch 0
model_small_bs64_lr0.01_epoch1 - checkpoint for epoch 1
model_small_bs64_lr0.01_epoch2 - checkpoint for epoch 2
model_small_bs64_lr0.01_epoch3 - checkpoint for epoch 3
model_small_bs64_lr0.01_epoch4 - checkpoint for epoch 4
model_small_bs64_lr0.01_epoch4_train_err.csv - contains the training error of sma
model_small_bs64_lr0.01_epoch4_train_loss.csv - contains the training loss of sma
model_small_bs64_lr0.01_epoch4_val_err.csv - contains the validation error of sma
model_small_bs64_lr0.01_epoch4_val_loss.csv - contains the validation loss of sma
NOTE: Checkpoints contain the saved values of model weights at the end of each ep
'''
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.435625, Train loss: 0.6780269684791564 |Validation err:
Epoch 2: Train err: 0.37125, Train loss: 0.6498568053245545 |Validation err: 0
Epoch 3: Train err: 0.360125, Train loss: 0.6389743585586548 |Validation err:
Epoch 4: Train err: 0.346, Train loss: 0.6269212069511414 |Validation err: 0.3
Epoch 5: Train err: 0.343375, Train loss: 0.618819887638092 |Validation err: 0
Finished Training
Total time elapsed: 32.67 seconds
'Files written to disk:\nmodel_small_bs64_lr0.01_epoch0 - checkpoint for epoc
h 0\nmodel_small_bs64_lr0.01_epoch1 - checkpoint for epoch 1\nmodel_small_bs6
4_lr0.01_epoch2 - checkpoint for epoch 2\nmodel_small_bs64_lr0.01_epoch3 - ch
```

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')
```

```
Mounted at /content/gdrive
```

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

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.3295, Train loss: 0.6080470056533813 |Validation err:
Epoch 2: Train err: 0.32325, Train loss: 0.6009767365455627 |Validation err:
Epoch 3: Train err: 0.317625, Train loss: 0.5949181377887726 |Validation err
Epoch 4: Train err: 0.310375, Train loss: 0.585449330329895 |Validation err:
Epoch 5: Train err: 0.30775, Train loss: 0.5802813990116119 |Validation err:
Epoch 6: Train err: 0.307125, Train loss: 0.5763311166763305 |Validation err
Epoch 7: Train err: 0.30225, Train loss: 0.5728312101364136 |Validation err:
Epoch 8: Train err: 0.2975, Train loss: 0.5678883669376373 |Validation err:
Epoch 9: Train err: 0.2955, Train loss: 0.5694682185649872 |Validation err:
Epoch 10: Train err: 0.295625, Train loss: 0.5676210699081421 |Validation er
Epoch 11: Train err: 0.289125, Train loss: 0.5627483797073364 |Validation er
Epoch 12: Train err: 0.289625, Train loss: 0.5587509093284607 |Validation er
Epoch 13: Train err: 0.28675, Train loss: 0.5628734345436096 |Validation err
Epoch 14: Train err: 0.28525, Train loss: 0.5563021373748779 |Validation err
Epoch 15: Train err: 0.28525, Train loss: 0.5552288398742676 |Validation err
Epoch 16: Train err: 0.292625, Train loss: 0.5597215616703033 |Validation er
Epoch 17: Train err: 0.286375, Train loss: 0.5570372834205627 |Validation er
Epoch 18: Train err: 0.28175, Train loss: 0.5545889959335327 |Validation err
Epoch 19: Train err: 0.283, Train loss: 0.5520740718841552 |Validation err:
Epoch 20: Train err: 0.282375, Train loss: 0.5524600350856781 |Validation er
Epoch 21: Train err: 0.285, Train loss: 0.5546856572628022 |Validation err:
Epoch 22: Train err: 0.278875, Train loss: 0.5527666714191437 |Validation er
Epoch 23: Train err: 0.2785, Train loss: 0.5516463830471039 |Validation err:
Epoch 24: Train err: 0.278, Train loss: 0.5491846735477448 |Validation err:
Epoch 25: Train err: 0.28, Train loss: 0.5503250417709351 |Validation err: 0
Epoch 26: Train err: 0.2745, Train loss: 0.5468985729217529 |Validation err:
Epoch 27: Train err: 0.27725, Train loss: 0.5482517714500428 |Validation err
Epoch 28: Train err: 0.27775, Train loss: 0.5490458183288575 |Validation err
Epoch 29: Train err: 0.275625, Train loss: 0.5496381249427795 |Validation er
Epoch 30: Train err: 0.278, Train loss: 0.5486094944477081 |Validation err:
Finished Training
Total time elapsed: 180.56 seconds
Files already downloaded and verified
```

```
Files already downloaded and verified
Epoch 1: Train err: 0.458625, Train loss: 0.6897763667106629 |Validation err
Epoch 2: Train err: 0.4205, Train loss: 0.6766249690055847 |Validation err:
Epoch 3: Train err: 0.388875, Train loss: 0.6574754805564881 |Validation err
Epoch 4: Train err: 0.358375, Train loss: 0.6312700505256653 |Validation err
Epoch 5: Train err: 0.33, Train loss: 0.612538691520691 |Validation err: 0.3
Epoch 6: Train err: 0.31825, Train loss: 0.5958428113460541 |Validation err:
Epoch 7: Train err: 0.310625, Train loss: 0.5846084616184235 |Validation err
Epoch 8: Train err: 0.30425, Train loss: 0.5726539959907532 |Validation err:
Epoch 9: Train err: 0.294, Train loss: 0.5638465766906738 |Validation err: 0
Epoch 10: Train err: 0.284375, Train loss: 0.5570552878379822 |Validation er
Epoch 11: Train err: 0.2805, Train loss: 0.5491197535991669 |Validation err:
Epoch 12: Train err: 0.275625, Train loss: 0.539030620098114 |Validation err
Epoch 13: Train err: 0.27025, Train loss: 0.5299799833297729 |Validation err
Epoch 14: Train err: 0.2655, Train loss: 0.5206754057407379 |Validation err:
Epoch 15: Train err: 0.253125, Train loss: 0.512121128234864 |Validation er
Epoch 16: Train err: 0.25525, Train loss: 0.5100615527629853 |Validation err
Epoch 17: Train err: 0.2495, Train loss: 0.4994930884838104 |Validation err:
Epoch 18: Train err: 0.238875, Train loss: 0.4864052612781525 |Validation er
Epoch 19: Train err: 0.232125, Train loss: 0.48123493695259095 |Validation e
Epoch 20: Train err: 0.231, Train loss: 0.4766269898414612 |Validation err:
Epoch 21: Train err: 0.226, Train loss: 0.46736584520339963 |Validation err:
Epoch 22: Train err: 0.21975, Train loss: 0.46160859060287474 |Validation er
```

`small_net` reported a total time elapsed of 148.57 seconds and `large_net` reported a total time elapsed of 160.40 seconds. `large_net` took longer to train because it had more layers and thus more parameters (as reported above), so the training process required more parameters to be updated at each epoch, which increased the time taken for each individual epoch, and thus increasing the overall training time.

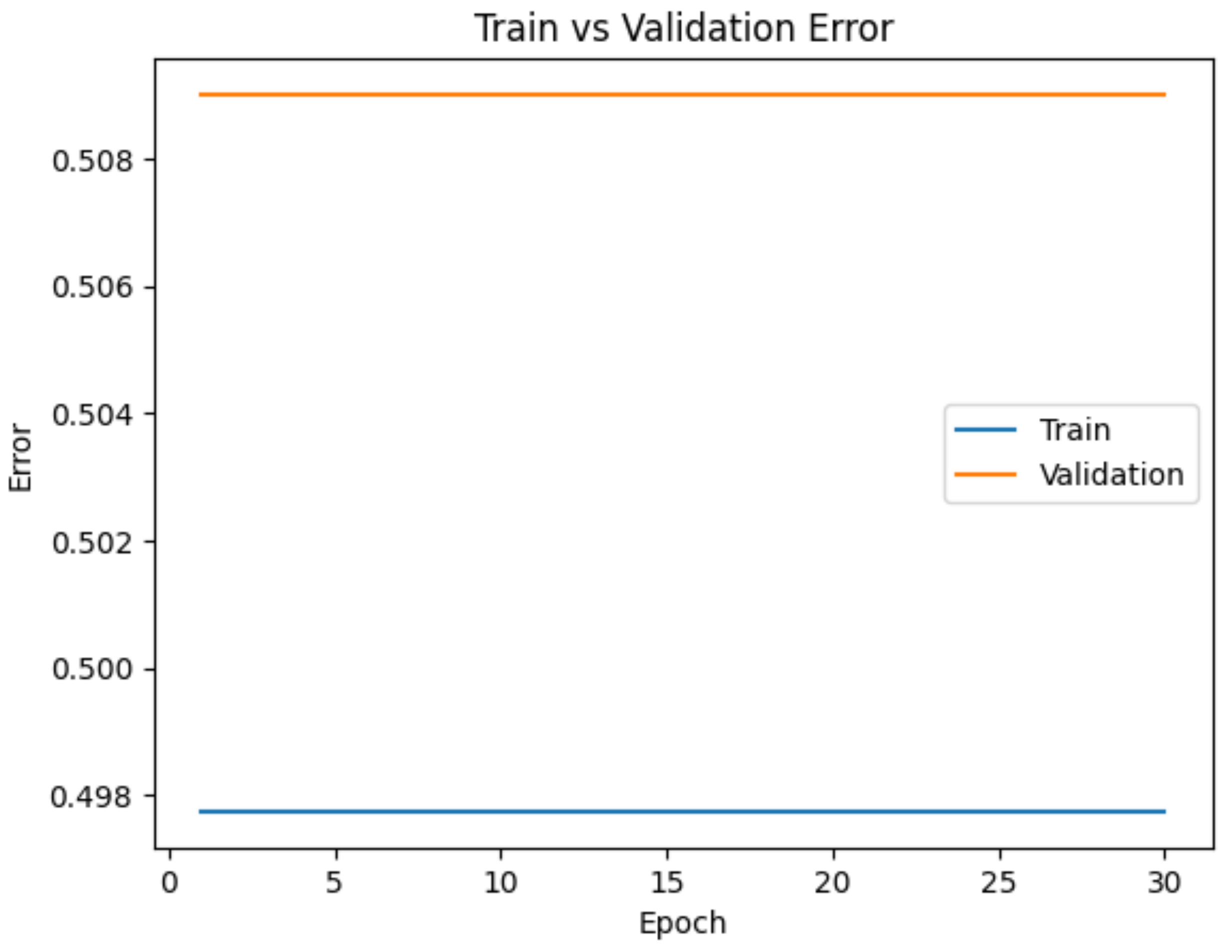
Part (e) - 2pt

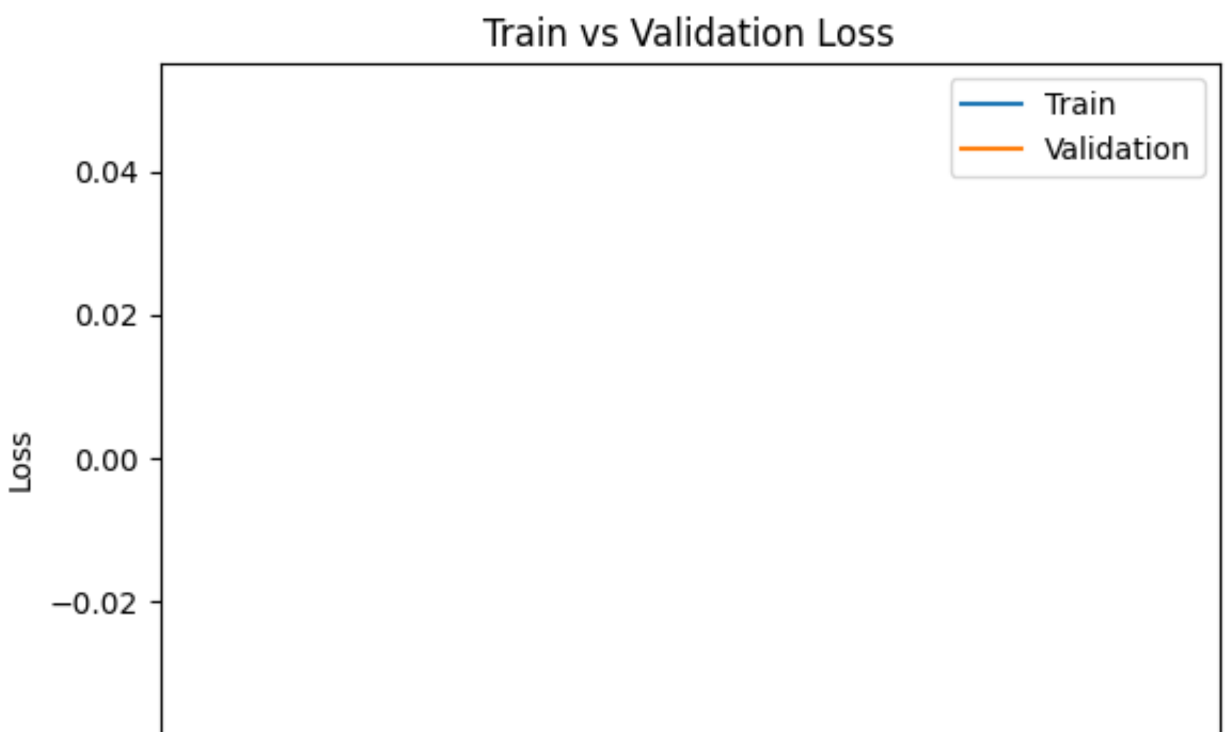
Use the function `plot_training_curve` to display the trajectory of the training/validation error and the training/validation loss. You will need to use the function `get_model_name` to generate the argument to the `plot_training_curve` function.

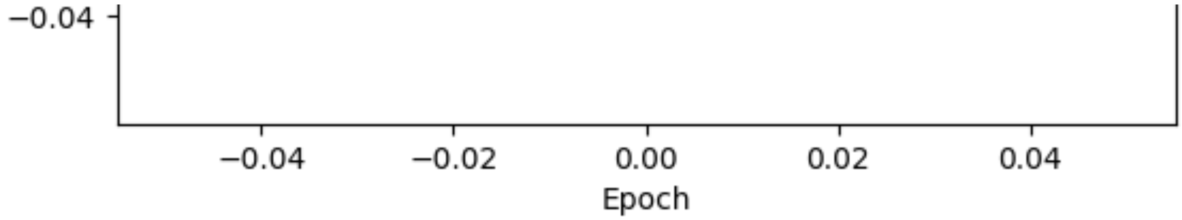
Do this for both the small network and the large network. Include both plots in your writeup.

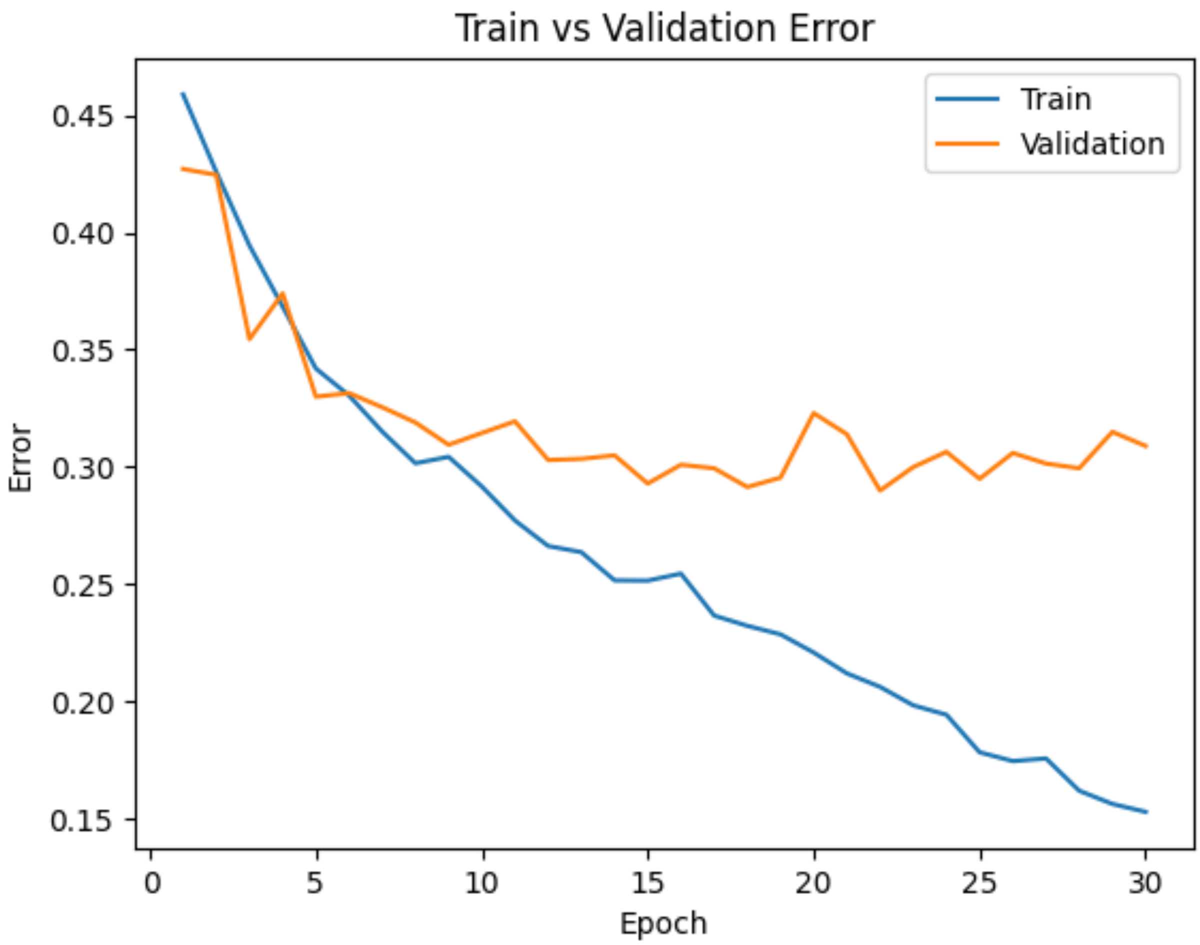
```
#model_path = get_model_name("small", batch_size=??, learning_rate=??, epoch=29)
small_path = get_model_name("small", batch_size=64, learning_rate=0.01, epoch=29)
plot_training_curve(small_path)
print("----------------------------------------------------------------------"
```

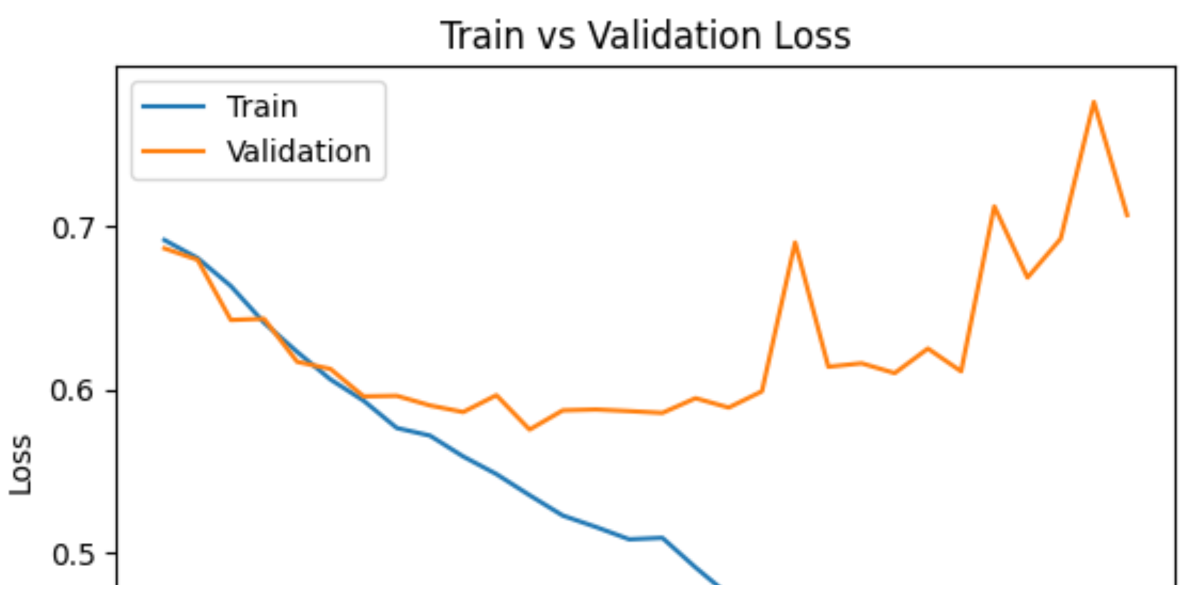
```
large_path = get_model_name("large", batch_size=64, learning_rate=0.01, epoch=29)
plot_training_curve(large_path)
```

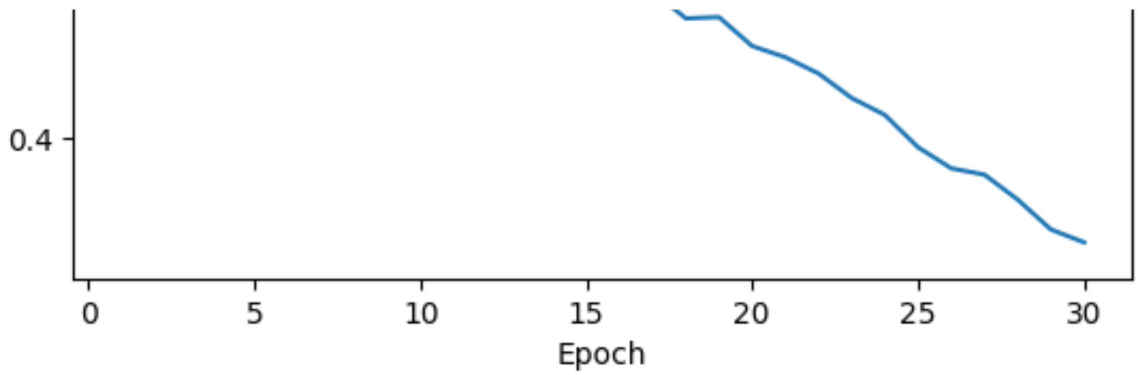





−0.04
−0.04
−0.02
0.00
0.02
0.04
Epoch


Train vs Validation Error
Train
Validation
Error
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0
5
10
15
20
25
30
Epoch


Train vs Validation Loss
Train
Validation
Loss
0.7
0.6
0.5



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.

```
'''The training curve for small_net maintains a consistent training error of 0.49
and validation errors over each epoch, although the validation error seems to beg
be computed (possibly due to the insane magnitude they may possess, as discussed
8 to 15 epochs.
This clearly demonstrates that small_net is an underfit model, as it has high tra
an underfit model (so simplistic that it returns high error). Meanwhile large_net
increase after ~15 epochs, and its validation error also appears to start to plea
overfitting.'''
```

Part 3. Optimization Parameters [12 pt]

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

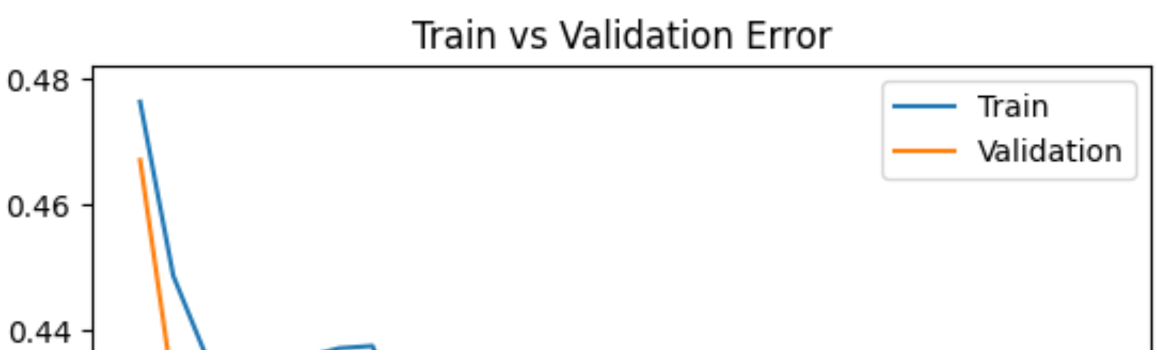
Part (a) - 3pt

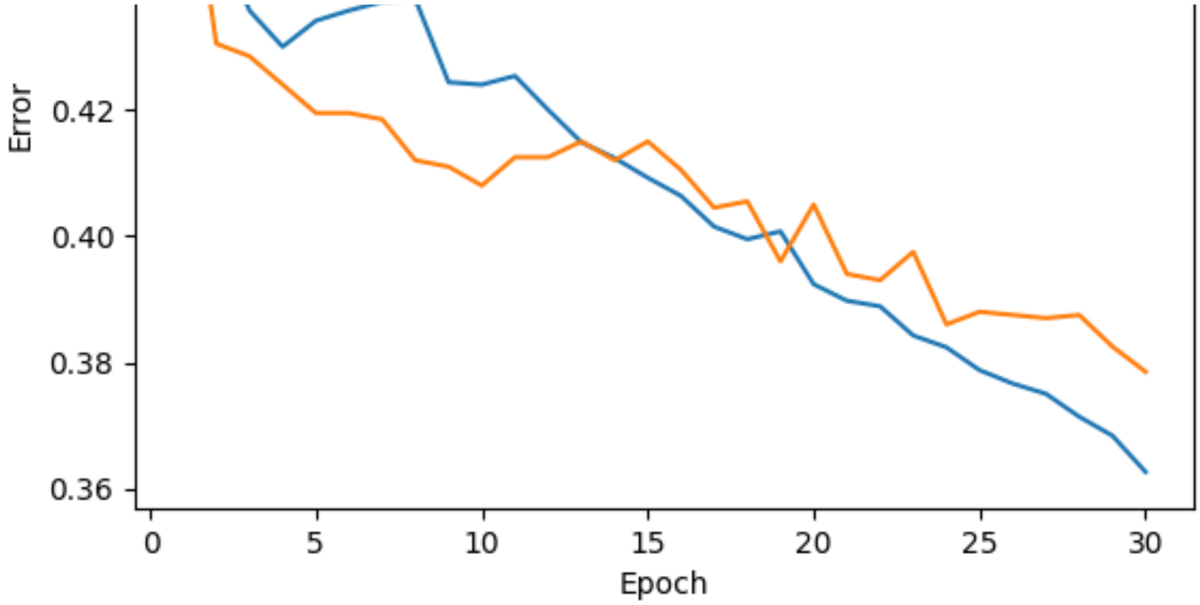
Train `large_net` with all default parameters, except set `learning_rate=0.001`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *lowering* the learning rate.

```
# 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(net=large_net, learning_rate=0.001)
large_path = get_model_name("large", batch_size=64, learning_rate=0.001, epoch=29
plot_training_curve(large_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.47625, Train loss: 0.6928360013961792 |Validation err: (
Epoch 2: Train err: 0.448625, Train loss: 0.6922589712142945 |Validation err:
Epoch 3: Train err: 0.43575, Train loss: 0.6916067280769348 |Validation err: (
Epoch 4: Train err: 0.43, Train loss: 0.690861343383789 |Validation err: 0.424
Epoch 5: Train err: 0.434125, Train loss: 0.6899195008277893 |Validation err:
Epoch 6: Train err: 0.43575, Train loss: 0.6887411961555481 |Validation err: (
Epoch 7: Train err: 0.437125, Train loss: 0.6873774147033691 |Validation err:
Epoch 8: Train err: 0.4375, Train loss: 0.6859278454780579 |Validation err: 0.
Epoch 9: Train err: 0.424375, Train loss: 0.6844058036804199 |Validation err:
Epoch 10: Train err: 0.424, Train loss: 0.6828502931594849 |Validation err: 0.
Epoch 11: Train err: 0.425375, Train loss: 0.6812348766326904 |Validation err:
Epoch 12: Train err: 0.42, Train loss: 0.6796319708824158 |Validation err: 0.4
Epoch 13: Train err: 0.414875, Train loss: 0.6777918744087219 |Validation err:
Epoch 14: Train err: 0.412375, Train loss: 0.6761112003326416 |Validation err:
Epoch 15: Train err: 0.40925, Train loss: 0.674472680568695 |Validation err: (
Epoch 16: Train err: 0.406375, Train loss: 0.6727448840141297 |Validation err:
Epoch 17: Train err: 0.4015, Train loss: 0.6713076601028443 |Validation err: (
Epoch 18: Train err: 0.3995, Train loss: 0.6696742882728577 |Validation err: (
Epoch 19: Train err: 0.40075, Train loss: 0.6679086356163025 |Validation err:
Epoch 20: Train err: 0.392375, Train loss: 0.665787980556488 |Validation err:
Epoch 21: Train err: 0.38975, Train loss: 0.6646300601959229 |Validation err:
Epoch 22: Train err: 0.388875, Train loss: 0.662373058795929 |Validation err:
Epoch 23: Train err: 0.38425, Train loss: 0.6601516346931458 |Validation err:
Epoch 24: Train err: 0.382375, Train loss: 0.6584009389877319 |Validation err:
Epoch 25: Train err: 0.37875, Train loss: 0.6554971766471863 |Validation err:
Epoch 26: Train err: 0.376625, Train loss: 0.6531173253059387 |Validation err:
Epoch 27: Train err: 0.375, Train loss: 0.6503696331977844 |Validation err: 0.
Epoch 28: Train err: 0.371375, Train loss: 0.6476435809135437 |Validation err:
Epoch 29: Train err: 0.368375, Train loss: 0.6451257643699646 |Validation err:
Epoch 30: Train err: 0.362625, Train loss: 0.6423329524993896 |Validation err:
Finished Training
Total time elapsed: 163.60 seconds
```




Error
0.42
0.40
0.38
0.36
0
5
10
15
20
25
30
Epoch


Train vs Validation Loss
Train
Validation
Loss
0.69
0.68
0.67
0.66
0.65
0.64
0
5
10
15
20
25
30
Epoch

Lowering the learning rate increases the time taken to 163.60 seconds (from 160.40 seconds with `learning_rate = 0.01`); however, it also significantly decreases the amount of overfitting occurring in the model and creates a better trained model, with both the Train and Validation errors and losses decreasing rapidly, and overall much more improvement of fitting noticed with drastic decreases in validation error and loss. Thus, in sum, lowering the learning rate better fits the model at the cost of a few seconds.

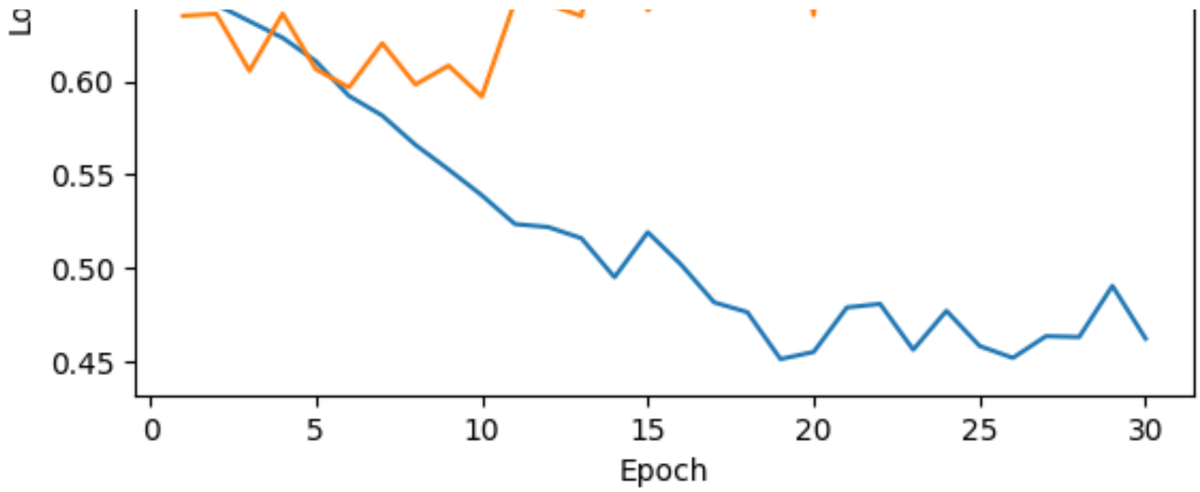
Part (b) - 3pt

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(net=large_net, learning_rate=0.1)
large_path = get_model_name("large", batch_size=64, learning_rate=0.1, epoch=29)
plot_training_curve(large_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.4295, Train loss: 0.67437779712677 |Validation err: 0.35
Epoch 2: Train err: 0.36075, Train loss: 0.6411805458068848 |Validation err: 0
Epoch 3: Train err: 0.365125, Train loss: 0.6321813461780548 |Validation err:
Epoch 4: Train err: 0.352625, Train loss: 0.6233456182479858 |Validation err:
Epoch 5: Train err: 0.34075, Train loss: 0.6108013873100281 |Validation err: 0
Epoch 6: Train err: 0.323375, Train loss: 0.5921835997104645 |Validation err:
Epoch 7: Train err: 0.3145, Train loss: 0.5817317583560944 |Validation err: 0.
Epoch 8: Train err: 0.29825, Train loss: 0.5660300073623658 |Validation err: 0
Epoch 9: Train err: 0.290875, Train loss: 0.552809501171112 |Validation err: 0
Epoch 10: Train err: 0.278625, Train loss: 0.539032607793808 |Validation err:
Epoch 11: Train err: 0.272375, Train loss: 0.5236025826931 |Validation err: 0.
Epoch 12: Train err: 0.267375, Train loss: 0.5220149435997009 |Validation err:
Epoch 13: Train err: 0.266, Train loss: 0.5160510110855102 |Validation err: 0.
Epoch 14: Train err: 0.24875, Train loss: 0.4951590054035187 |Validation err:
Epoch 15: Train err: 0.264625, Train loss: 0.519231944322586 |Validation err:
Epoch 16: Train err: 0.252625, Train loss: 0.5020012385845184 |Validation err:
Epoch 17: Train err: 0.23875, Train loss: 0.481714787364006 |Validation err: 0
Epoch 18: Train err: 0.23375, Train loss: 0.47645506453514097 |Validation err:
Epoch 19: Train err: 0.218125, Train loss: 0.45134368968009947 |Validation err
Epoch 20: Train err: 0.217875, Train loss: 0.45516350817680357 |Validation err
Epoch 21: Train err: 0.23275, Train loss: 0.47897080445289614 |Validation err:
```

```
Epoch 22: Train err: 0.234875, Train loss: 0.48088105654 71649 |Validation err:
Epoch 23: Train err: 0.21575, Train loss: 0.4563647754192352 |Validation err:
Epoch 24: Train err: 0.2355, Train loss: 0.47718250966072084 |Validation err:
Epoch 25: Train err: 0.22025, Train loss: 0.4583414270877838 |Validation err:
Epoch 26: Train err: 0.209625, Train loss: 0.4519626965522766 |Validation err:
Epoch 27: Train err: 0.22175, Train loss: 0.4636160457134247 |Validation err:
Epoch 28: Train err: 0.219375, Train loss: 0.46314777398109436 |Validation err
Epoch 29: Train err: 0.235875, Train loss: 0.49053542733192446 |Validation err
Epoch 30: Train err: 0.22, Train loss: 0.4623157248497009 |Validation err: 0.3
Finished Training
Total time elapsed: 161.74 seconds
```



The model now takes slightly more time to train than the original learning rate of 0.01, but less time to train than the learning rate of 0.001, reporting a total time of 161.74 seconds. However, it also performs significantly worse than both the models with learning rates of 0.01 and 0.001, demonstrating an *increasing* validation loss for progressing epochs. Thus, increasing the learning rate decreases the accuracy of the model, and still takes more time than the learning rate of 0.01.

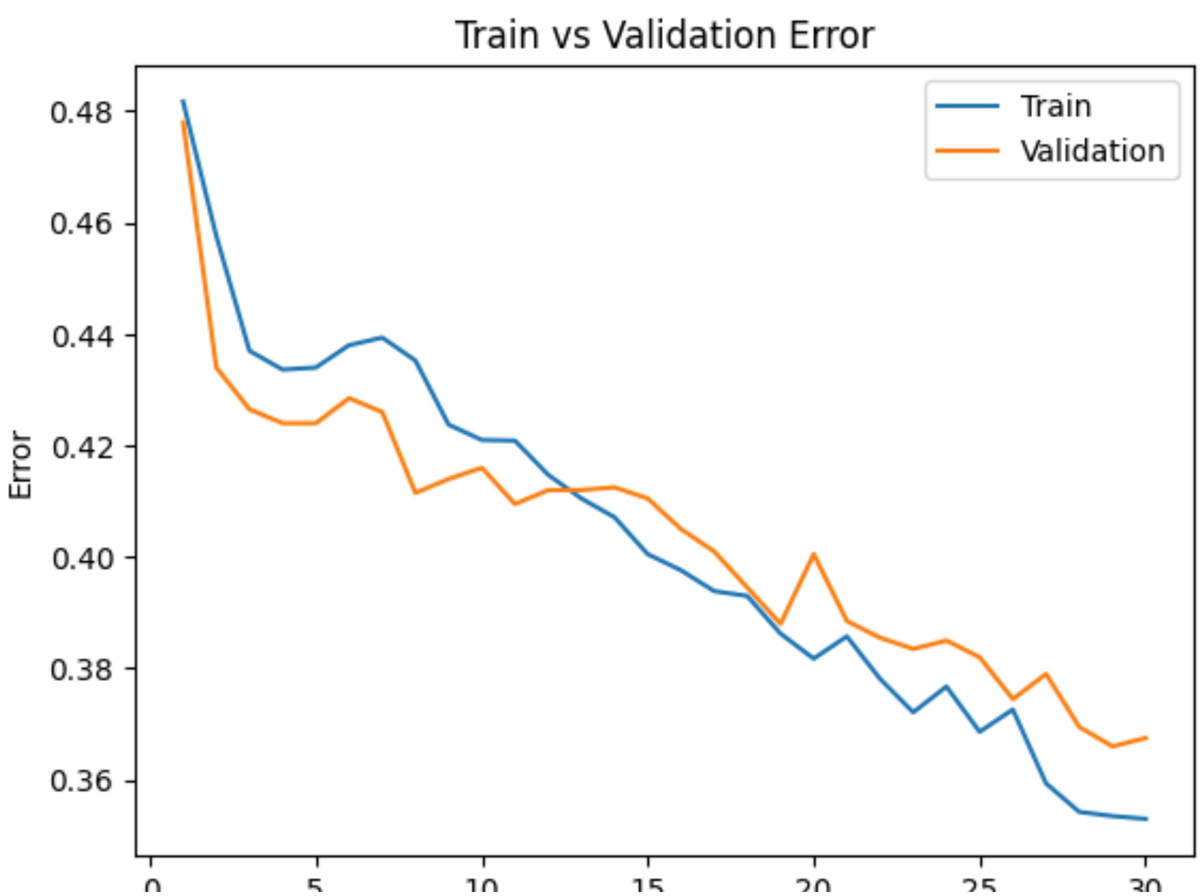
Part (c) - 3pt

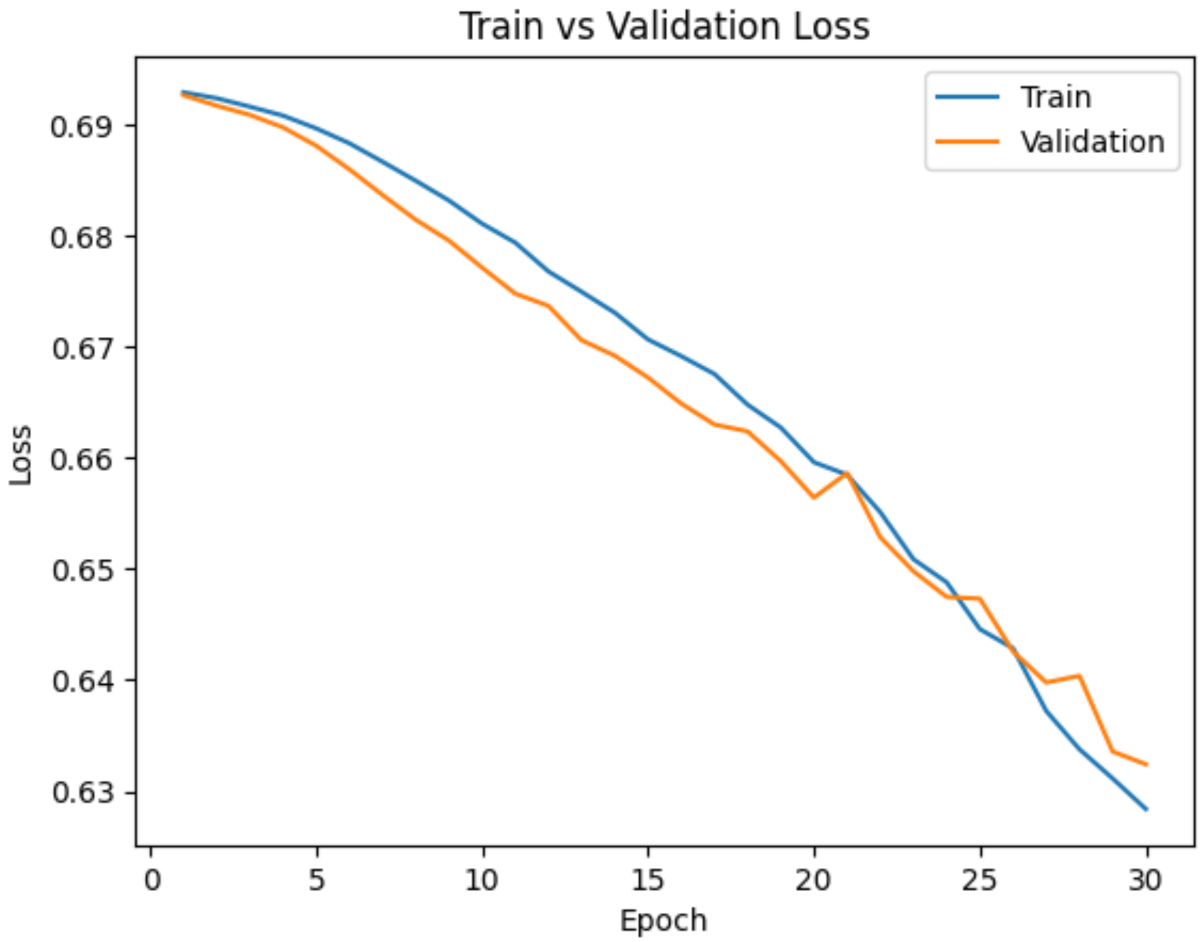
Train `large_net` with all default parameters, including with `learning_rate=0.01`. Now, set `batch_size=512`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *increasing* the batch size.

```
large_net = LargeNet()

train_net(net=large_net, batch_size=512)
large_path = get_model_name("large", batch_size=512, learning_rate=0.01, epoch=29
plot_training_curve(large_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.48175, Train loss: 0.6929379552602768 |Validation err: 0
Epoch 2: Train err: 0.457625, Train loss: 0.6924104019999504 |Validation err:
Epoch 3: Train err: 0.437, Train loss: 0.6916500590741634 |Validation err: 0.4
Epoch 4: Train err: 0.433625, Train loss: 0.6908449940383434 |Validation err:
Epoch 5: Train err: 0.434, Train loss: 0.6896935552358627 |Validation err: 0.4
Epoch 6: Train err: 0.438, Train loss: 0.688353206962347 |Validation err: 0.42
```

```
Epoch 7: Train err: 0.439375, Train loss: 0.6866871677339077 |Validation err:
Epoch 8: Train err: 0.43525, Train loss: 0.6849770769476891 |Validation err: (
Epoch 9: Train err: 0.42375, Train loss: 0.6832009293138981 |Validation err: (
Epoch 10: Train err: 0.421, Train loss: 0.6811089366674423 |Validation err: 0.
Epoch 11: Train err: 0.420875, Train loss: 0.6794026419520378 |Validation err:
Epoch 12: Train err: 0.41475, Train loss: 0.6768048219382763 |Validation err:
Epoch 13: Train err: 0.4105, Train loss: 0.6749702803790569 |Validation err: (
Epoch 14: Train err: 0.407125, Train loss: 0.6730880849063396 |Validation err:
Epoch 15: Train err: 0.4005, Train loss: 0.6706806942820549 |Validation err: (
Epoch 16: Train err: 0.397625, Train loss: 0.6691771410405636 |Validation err:
Epoch 17: Train err: 0.393875, Train loss: 0.6675694733858109 |Validation err:
Epoch 18: Train err: 0.393, Train loss: 0.6648042872548103 |Validation err: 0.
Epoch 19: Train err: 0.38625, Train loss: 0.662746611982584 |Validation err: (
Epoch 20: Train err: 0.38175, Train loss: 0.6596181839704514 |Validation err:
Epoch 21: Train err: 0.38575, Train loss: 0.6584899798035622 |Validation err:
Epoch 22: Train err: 0.378125, Train loss: 0.655123382806778 |Validation err:
Epoch 23: Train err: 0.372125, Train loss: 0.6508794128894806 |Validation err:
Epoch 24: Train err: 0.37675, Train loss: 0.6488028429448605 |Validation err:
Epoch 25: Train err: 0.368625, Train loss: 0.6445869170129299 |Validation err:
Epoch 26: Train err: 0.372625, Train loss: 0.6428566053509712 |Validation err:
Epoch 27: Train err: 0.359375, Train loss: 0.6372117549180984 |Validation err:
Epoch 28: Train err: 0.35425, Train loss: 0.6337667480111122 |Validation err:
Epoch 29: Train err: 0.3535, Train loss: 0.6311353109776974 |Validation err: (
Epoch 30: Train err: 0.353, Train loss: 0.6283832415938377 |Validation err: 0.
Finished Training
Total time elapsed: 142.88 seconds
```





The model now takes shorter to train, completing training in 142.88 seconds. It also exhibits better fitting than the previous batch_size of 64, demonstrating rapidly decreasing training and validation errors and losses, and thus is not too overfitted or underfitted. Thus, increasing the batch size creates a huge improvement on the accuracy and efficiency of the model.

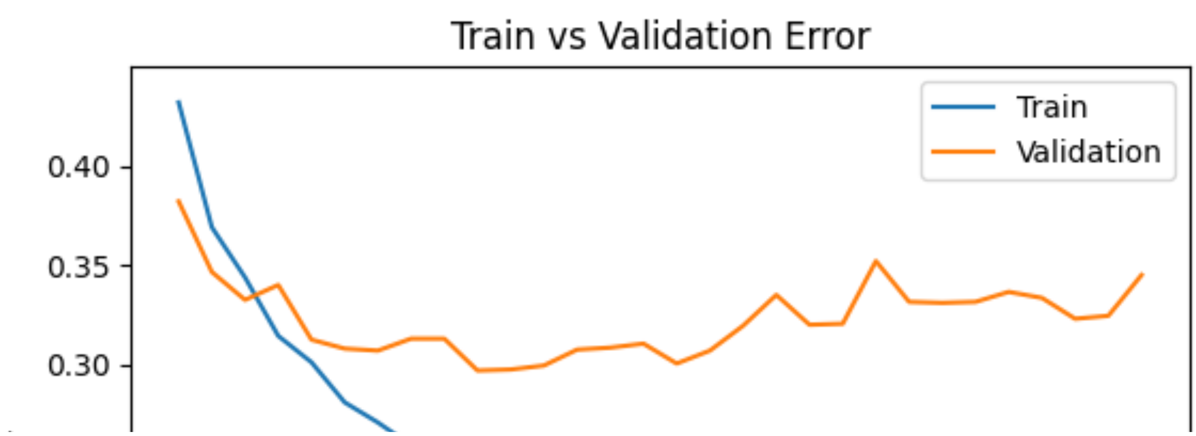
Part (d) - 3pt

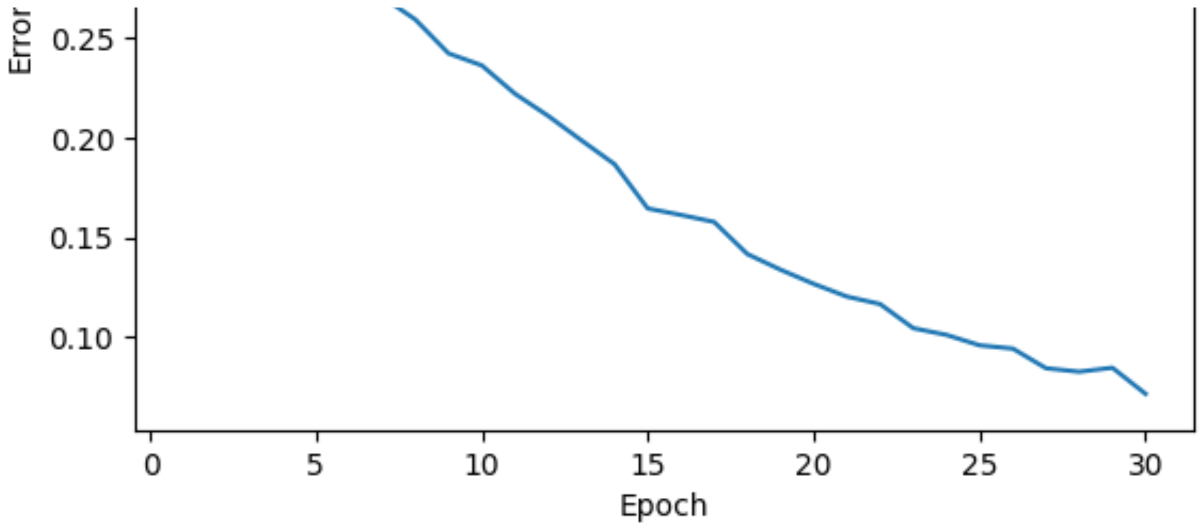
Train `large_net` with all default parameters, including with `learning_rate=0.01`. Now, set `batch_size=16`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *decreasing* the batch size.

```
large_net = LargeNet()
```

```
train_net(net=large_net, batch_size=16)
large_path = get_model_name("large", batch_size=16, learning_rate=0.01, epoch=29)
plot_training_curve(large_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.43175, Train loss: 0.6774994022846222 |Validation err: (
Epoch 2: Train err: 0.369, Train loss: 0.639639899969101 |Validation err: 0.34
Epoch 3: Train err: 0.34375, Train loss: 0.6098222947120666 |Validation err: (
Epoch 4: Train err: 0.314375, Train loss: 0.5849691489338875 |Validation err:
Epoch 5: Train err: 0.301125, Train loss: 0.5689119303822517 |Validation err:
Epoch 6: Train err: 0.281, Train loss: 0.5452213581204415 |Validation err: 0.3
Epoch 7: Train err: 0.270875, Train loss: 0.5272981298565864 |Validation err:
Epoch 8: Train err: 0.259375, Train loss: 0.5070905526578426 |Validation err:
Epoch 9: Train err: 0.242375, Train loss: 0.4968344421982765 |Validation err:
Epoch 10: Train err: 0.236375, Train loss: 0.4756101597249508 |Validation err:
Epoch 11: Train err: 0.222125, Train loss: 0.4599769461452961 |Validation err:
Epoch 12: Train err: 0.211, Train loss: 0.4454492371380329 |Validation err: 0.
Epoch 13: Train err: 0.19875, Train loss: 0.4245421719551086 |Validation err:
Epoch 14: Train err: 0.18675, Train loss: 0.4007472907453775 |Validation err:
Epoch 15: Train err: 0.1645, Train loss: 0.3759974058121443 |Validation err: (
Epoch 16: Train err: 0.16125, Train loss: 0.3591455406397581 |Validation err:
Epoch 17: Train err: 0.15775, Train loss: 0.3463234790861607 |Validation err:
Epoch 18: Train err: 0.141625, Train loss: 0.32175366275012496 |Validation err
Epoch 19: Train err: 0.13375, Train loss: 0.30618105667084455 |Validation err:
Epoch 20: Train err: 0.126625, Train loss: 0.3029071792438626 |Validation err:
Epoch 21: Train err: 0.12025, Train loss: 0.28682796490937473 |Validation err:
Epoch 22: Train err: 0.1165, Train loss: 0.27489088076353074 |Validation err:
Epoch 23: Train err: 0.104375, Train loss: 0.2467898527495563 |Validation err:
Epoch 24: Train err: 0.101, Train loss: 0.23970085787773132 |Validation err: (
Epoch 25: Train err: 0.09575, Train loss: 0.23643119425699116 |Validation err:
Epoch 26: Train err: 0.094125, Train loss: 0.2325953512713313 |Validation err:
Epoch 27: Train err: 0.08425, Train loss: 0.21040759468451142 |Validation err:
Epoch 28: Train err: 0.0825, Train loss: 0.20643112615589052 |Validation err:
Epoch 29: Train err: 0.0845, Train loss: 0.21273409337876364 |Validation err:
Epoch 30: Train err: 0.071375, Train loss: 0.18387044295761734 |Validation err
Finished Training
Total time elapsed: 217.53 seconds
```




Error
0.25
0.20
0.15
0.10
0
5
10
15
20
25
30
Epoch


Train vs Validation Loss
Train
Validation
Loss
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
5
10
15
20
25
30
Epoch

Decreasing the batch size made it take longer to train, increasing the total time elapsed to 217.53 seconds (the highest so far), while also decreasing the accuracy of the model and demonstrating severe overfitting, since the training error and loss decrease rapidly for increasing epochs, but the validation error decreases slightly, plateaus, and then begins to increase again, and the validation loss increases rapidly during the training process. Thus, decreasing the batch size makes the model severely inaccurate, overfitted, and inefficient.

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 = LargeNet()
batch_size = 700
learning_rate = 0.001
'''I picked these values because based on the plots above, large_net is more accu
batch size increased both accuracy and thus the fitting of the model and efficien
theoretically by the time decrease from the increased batch size, so these values
```

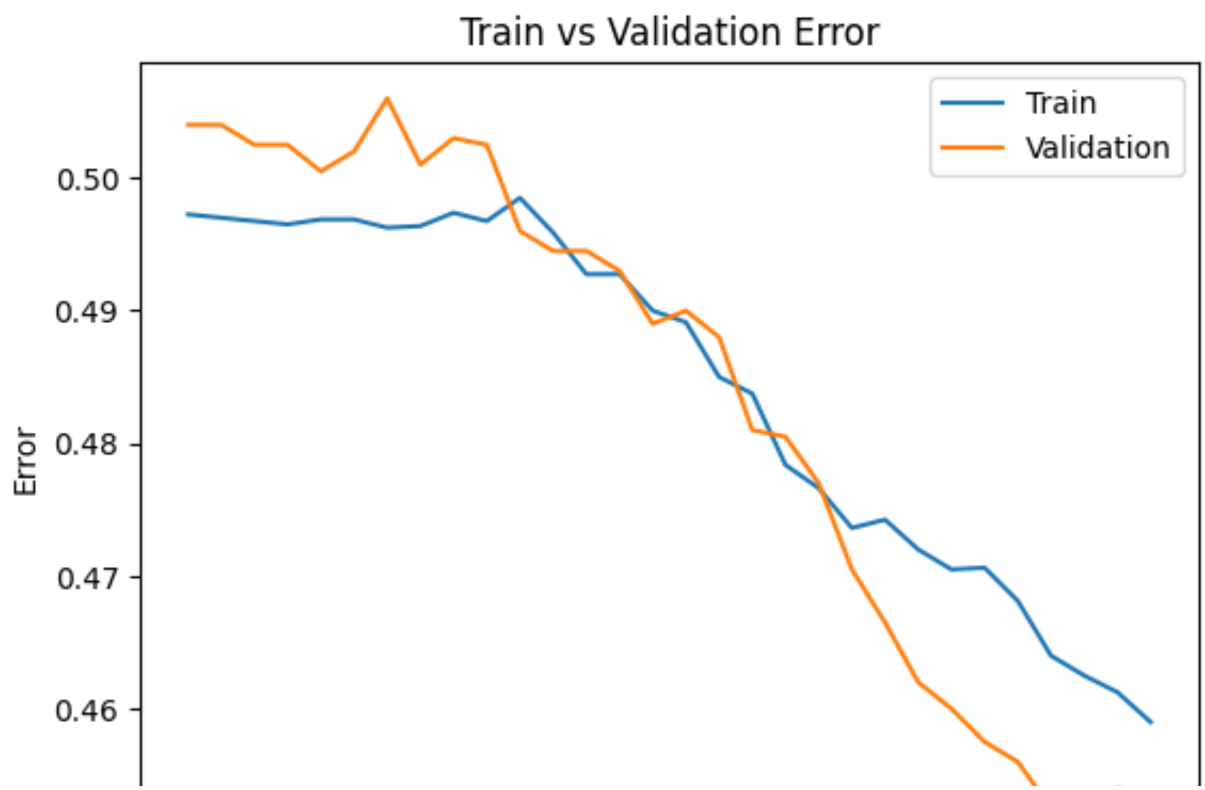
Part (b) - 1pt

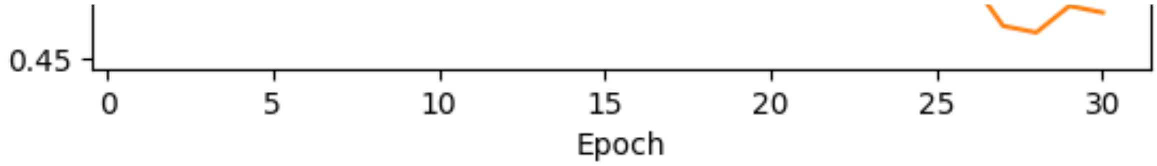
Train the model with the hyperparameters you chose in part(a), and include the training curve.

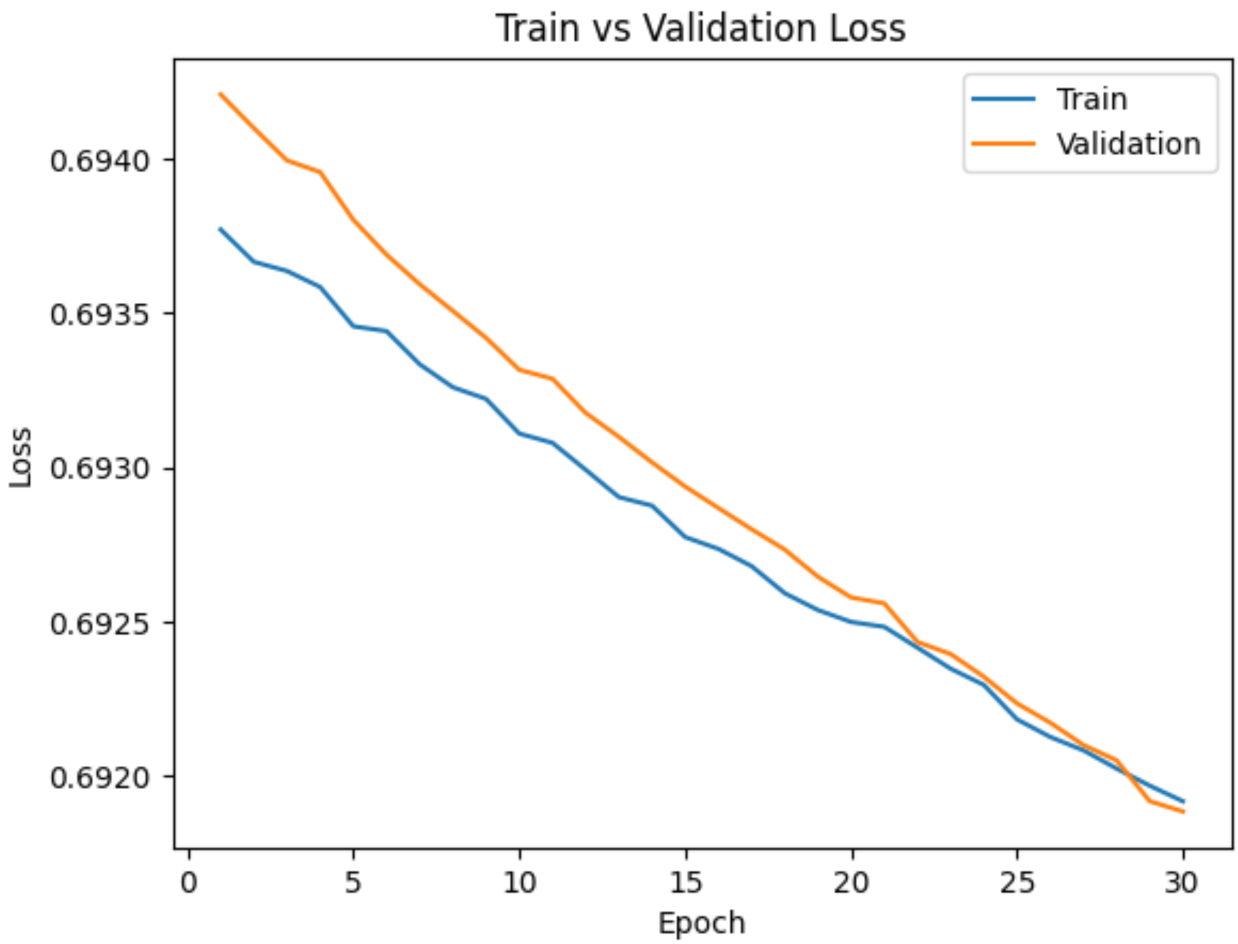
```
large_net = LargeNet()

train_net(net=large_net, batch_size=700, learning_rate=0.001)
large_path = get_model_name("large", batch_size=700, learning_rate=0.001, epoch=2
plot_training_curve(large_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.49725, Train loss: 0.6937709798415502 |Validation err: 0
Epoch 2: Train err: 0.497, Train loss: 0.6936662693818411 |Validation err: 0.5
Epoch 3: Train err: 0.49675, Train loss: 0.6936365564664205 |Validation err: 0
```

```
Epoch 4: Train err: 0.4965, Train loss: 0.6935847202936808 |Validation err: 0.
Epoch 5: Train err: 0.496875, Train loss: 0.6934573849042257 |Validation err:
Epoch 6: Train err: 0.496875, Train loss: 0.6934417535861334 |Validation err:
Epoch 7: Train err: 0.49625, Train loss: 0.6933341572682062 |Validation err: 0
Epoch 8: Train err: 0.496375, Train loss: 0.6932598501443863 |Validation err:
Epoch 9: Train err: 0.497375, Train loss: 0.6932220856348673 |Validation err:
Epoch 10: Train err: 0.49675, Train loss: 0.6931097457806269 |Validation err:
Epoch 11: Train err: 0.4985, Train loss: 0.6930797646443049 |Validation err: 0
Epoch 12: Train err: 0.495875, Train loss: 0.6929926623900732 |Validation err:
Epoch 13: Train err: 0.49275, Train loss: 0.692904050151507 |Validation err: 0
Epoch 14: Train err: 0.49275, Train loss: 0.6928762743870417 |Validation err:
Epoch 15: Train err: 0.49, Train loss: 0.6927744150161743 |Validation err: 0.4
Epoch 16: Train err: 0.489125, Train loss: 0.6927362432082494 |Validation err:
Epoch 17: Train err: 0.485, Train loss: 0.6926803886890411 |Validation err: 0.
Epoch 18: Train err: 0.48375, Train loss: 0.692592610915502 |Validation err: 0
Epoch 19: Train err: 0.478375, Train loss: 0.6925380229949951 |Validation err:
Epoch 20: Train err: 0.476625, Train loss: 0.6924994687239329 |Validation err:
Epoch 21: Train err: 0.473625, Train loss: 0.6924836933612823 |Validation err:
Epoch 22: Train err: 0.47425, Train loss: 0.692416210969289 2 |Validation err:
Epoch 23: Train err: 0.472, Train loss: 0.6923480580250422 |Validation err: 0.
Epoch 24: Train err: 0.4705, Train loss: 0.6922955214977264 |Validation err: 0
Epoch 25: Train err: 0.470625, Train loss: 0.6921838571627935 |Validation err:
Epoch 26: Train err: 0.468125, Train loss: 0.6921269049247106 |Validation err:
Epoch 27: Train err: 0.464, Train loss: 0.6920836518208185 |Validation err: 0.
Epoch 28: Train err: 0.4625, Train loss: 0.6920246183872223 |Validation err: 0
Epoch 29: Train err: 0.46125, Train loss: 0.6919689526160558 |Validation err:
Epoch 30: Train err: 0.459, Train loss: 0.69191841284434 |Validation err: 0.45
Finished Training
Total time elapsed: 145.09 seconds
```







Part (c) - 2pt

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

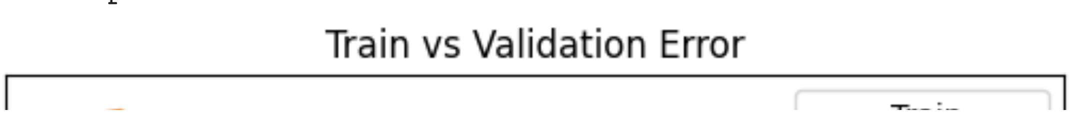
```
network = LargeNet()
batch_size = 900
learning_rate = 0.001
'''The results above seem positive, so this is now doubling down on everything do
```

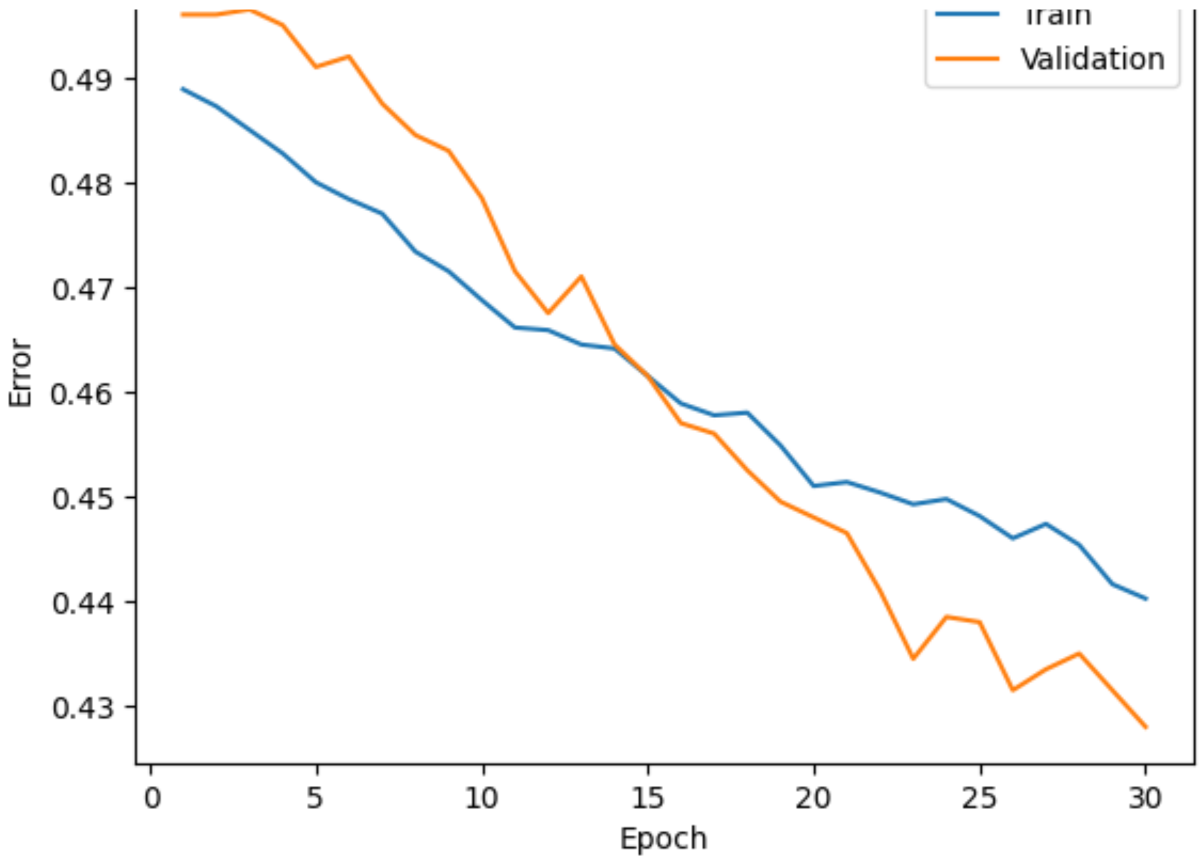
Part (d) - 1pt

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

```
large_net = LargeNet()

train_net(net=large_net, batch_size=900, learning_rate=0.001)
large_path = get_model_name("large", batch_size=900, learning_rate=0.001, epoch=2
plot_training_curve(large_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.488875, Train loss: 0.6930699414677091 |Validation err:
Epoch 2: Train err: 0.48725, Train loss: 0.6930334236886766 |Validation err: 0
Epoch 3: Train err: 0.485, Train loss: 0.693001925945282 |Validation err: 0.49
Epoch 4: Train err: 0.48275, Train loss: 0.6929701699150933 |Validation err: 0
Epoch 5: Train err: 0.48, Train loss: 0.6929284201727973 |Validation err: 0.49
Epoch 6: Train err: 0.478375, Train loss: 0.6928779416614108 |Validation err:
Epoch 7: Train err: 0.477, Train loss: 0.6928358674049377 |Validation err: 0.4
Epoch 8: Train err: 0.473375, Train loss: 0.6927957468562655 |Validation err:
Epoch 9: Train err: 0.4715, Train loss: 0.6927448047531976 |Validation err: 0.
Epoch 10: Train err: 0.46875, Train loss: 0.6927042735947503 |Validation err:
Epoch 11: Train err: 0.466125, Train loss: 0.6926594773928324 |Validation err:
Epoch 12: Train err: 0.465875, Train loss: 0.6926183236969842 |Validation err:
Epoch 13: Train err: 0.4645, Train loss: 0.6925761699676514 |Validation err: 0
Epoch 14: Train err: 0.464125, Train loss: 0.6925377051035563 |Validation err:
Epoch 15: Train err: 0.4615, Train loss: 0.6924908690982394 |Validation err: 0
Epoch 16: Train err: 0.458875, Train loss: 0.6924456291728549 |Validation err:
Epoch 17: Train err: 0.45775, Train loss: 0.6924138996336195 |Validation err:
Epoch 18: Train err: 0.458, Train loss: 0.6923718253771464 |Validation err: 0.
Epoch 19: Train err: 0.454875, Train loss: 0.6923299299346076 |Validation err:
Epoch 20: Train err: 0.451, Train loss: 0.6922861470116509 |Validation err: 0.
Epoch 21: Train err: 0.451375, Train loss: 0.6922404103808932 |Validation err:
Epoch 22: Train err: 0.450375, Train loss: 0.6921975480185615 |Validation err:
Epoch 23: Train err: 0.44925, Train loss: 0.6921489768558078 |Validation err:
Epoch 24: Train err: 0.44975, Train loss: 0.6921018428272672 |Validation err:
Epoch 25: Train err: 0.448125, Train loss: 0.6920628415213691 |Validation err:
Epoch 26: Train err: 0.446, Train loss: 0.6920124159918891 |Validation err: 0.
Epoch 27: Train err: 0.447375, Train loss: 0.6919745802879333 |Validation err:
Epoch 28: Train err: 0.445375, Train loss: 0.6919245786137052 |Validation err:
Epoch 29: Train err: 0.441625, Train loss: 0.6918752259678311 |Validation err:
Epoch 30: Train err: 0.44025, Train loss: 0.6918278204070197 |Validation err:
Finished Training
Total time elapsed: 147.48 seconds
```




Train
Validation
0.49
0.48
0.47
0.46
0.45
0.44
0.43
Error
0
5
10
15
20
25
30
Epoch


Train vs Validation Loss
Train
Validation
0.69325
0.69300
0.69275
0.69250
0.69225
0.69200
0.69175
0.69150
Loss
0
5
10
15
20
25
30
Epoch

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`.

```
net = SmallNet()
model_path = get_model_name(net.name, batch_size=64, learning_rate=0.01, epoch=10
state = torch.load(model_path)
net.load_state_dict(state)
```

```
#Best model:

net = LargeNet()
model_path = get_model_name(net.name, batch_size=900, learning_rate=0.001, 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).

```
'''This model had the lowest training and validation errors and losses, and there
```

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=900)

test_err, test_loss = evaluate(net, test_loader, nn.BCEWithLogitsLoss())
print("Test Error: ", test_err, "Test Loss: ", test_loss)
```

```
Files already downloaded and verified
Files already downloaded and verified
Test Error:  0.432 Test Loss:  0.691737155119578
```

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.

```
'''The test error is slightly higher than the validation error, being 0.4335 comp
once, so it is likely to make more errors than with the validation data, which it
revise questions you got wrong in the first attempt and keep attempting until you
either too different from the practice test or you couldn't make the connection,
```

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?

```
'''The test set was used at the very end to test how well the best version of our
It is crucial to use the test data as little as possible because the more the tes
test vs. practice test" analogy from before, the more times a student retakes a t
of course always possible that the student's generalization ability has also impr
```

Part (f) - 5pt

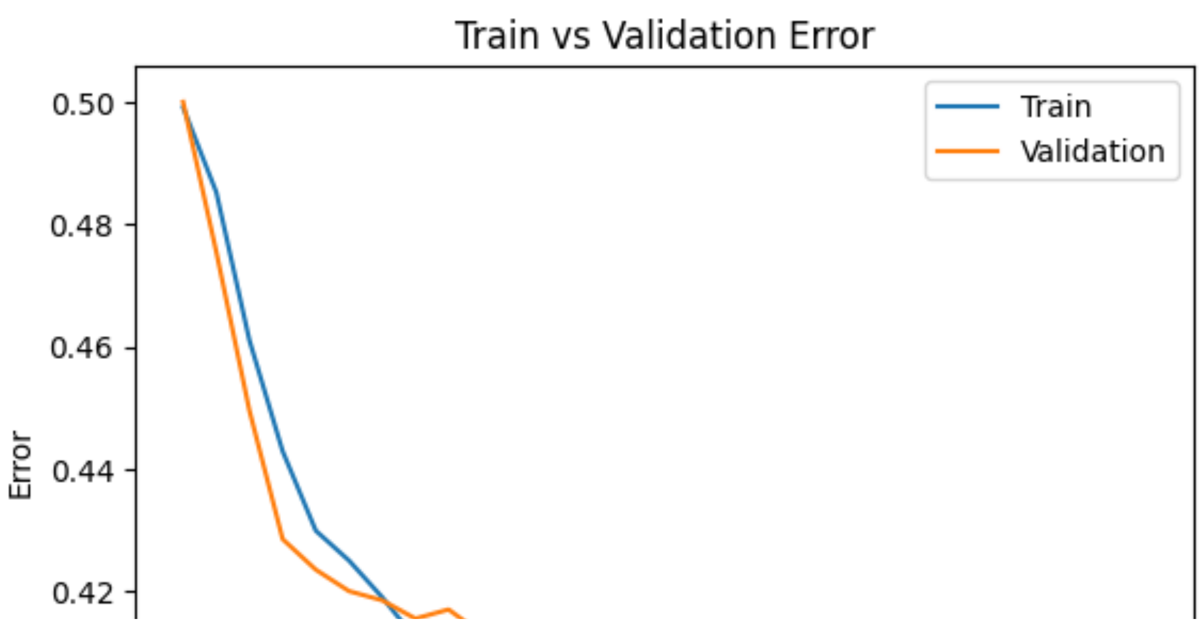
How does the your best CNN model compare with an 2-layer ANN model (no convolutional layers) on classifying cat and dog images. You can use a 2-layer ANN architecture similar to what you used in Lab 1. You should explore different hyperparameter settings to determine how well you can do on the validation dataset. Once satisified with the performance, you may test it out on the test data.

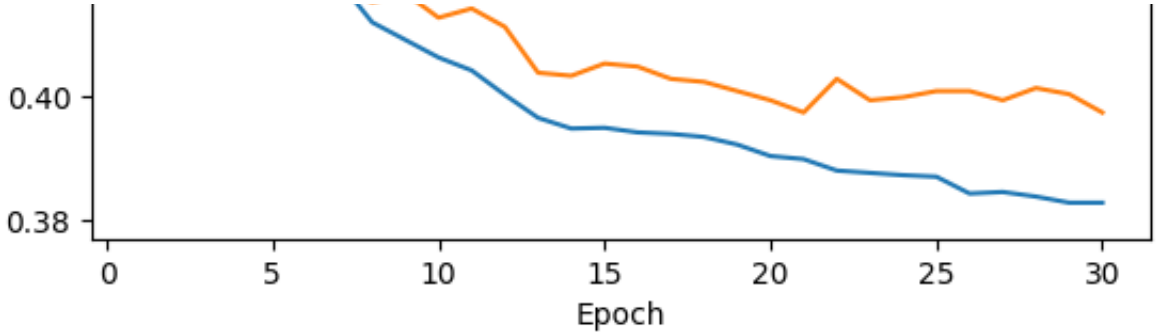
Hint: The ANN in lab 1 was applied on greyscale images. The cat and dog images are colour (RGB) and so you will need to flatted and concatinate all three colour layers before feeding them into an ANN.

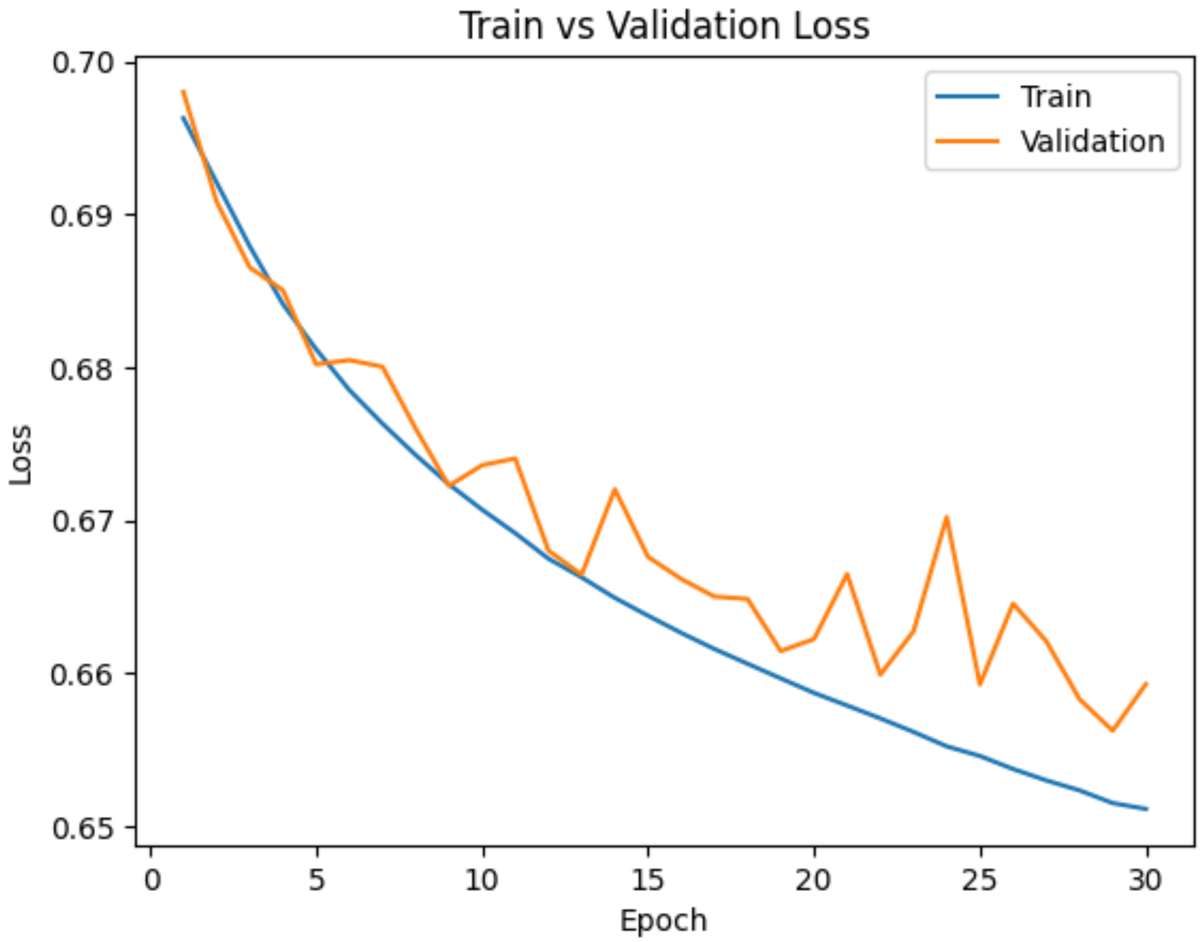
```
torch.manual_seed(1000) # set the random seed

# define a 2-layer artificial neural network
class ANN(nn.Module):
    def __init__(self):
        super(ANN, self).__init__()
        self.name = "ann"
        self.layer1 = nn.Linear(3 * 32 * 32, 32)
        self.layer2 = nn.Linear(32, 1)
    def forward(self, x):
        flattened = x.view(-1, 3 * 32 * 32)
        activation1 = self.layer1(flattened)
        activation1 = F.relu(activation1)
        activation2 = self.layer2(activation1)
        activation2 = activation2.squeeze(1)
        return activation2

ann = ANN()

train_net(net=ann, batch_size=900, learning_rate=0.001)
ann_path = get_model_name("ann", batch_size=900, learning_rate=0.001, epoch=29)
plot_training_curve(ann_path)
```

```
Files already downloaded and verified
```

```
Files already downloaded and verified
Epoch 1: Train err: 0.499125, Train loss: 0.6962915990087721 |Validation err:
Epoch 2: Train err: 0.48525, Train loss: 0.692088438404931 |Validation err: 0.
Epoch 3: Train err: 0.461, Train loss: 0.6879125369919671 |Validation err: 0.4
Epoch 4: Train err: 0.442875, Train loss: 0.6841277943717109 |Validation err:
Epoch 5: Train err: 0.429875, Train loss: 0.681184622976515 |Validation err: 0
Epoch 6: Train err: 0.425, Train loss: 0.6785153746604919 |Validation err: 0.4
Epoch 7: Train err: 0.419, Train loss: 0.6763131552272372 |Validation err: 0.4
Epoch 8: Train err: 0.41225, Train loss: 0.6742619077364603 |Validation err: 0
Epoch 9: Train err: 0.409375, Train loss: 0.672358062532213 |Validation err: 0
Epoch 10: Train err: 0.4065, Train loss: 0.6706896490520902 |Validation err: 0
Epoch 11: Train err: 0.404375, Train loss: 0.6691503259870741 |Validation err:
Epoch 12: Train err: 0.400375, Train loss: 0.6674857007132636 |Validation err:
Epoch 13: Train err: 0.396625, Train loss: 0.6662647591696845 |Validation err:
Epoch 14: Train err: 0.394875, Train loss: 0.6649220983187357 |Validation err:
Epoch 15: Train err: 0.395, Train loss: 0.6637577083375719 |Validation err: 0.
Epoch 16: Train err: 0.39425, Train loss: 0.662631438838111 |Validation err: 0
Epoch 17: Train err: 0.394, Train loss: 0.6615797678629557 |Validation err: 0.
Epoch 18: Train err: 0.3935, Train loss: 0.6606135699484084 |Validation err: 0
Epoch 19: Train err: 0.39225, Train loss: 0.6596558623843722 |Validation err:
Epoch 20: Train err: 0.390375, Train loss: 0.6587084598011441 |Validation err:
Epoch 21: Train err: 0.389875, Train loss: 0.6578803592258029 |Validation err:
Epoch 22: Train err: 0.388, Train loss: 0.6570409735043844 |Validation err: 0.
Epoch 23: Train err: 0.387625, Train loss: 0.6561560299661424 |Validation err:
Epoch 24: Train err: 0.38725, Train loss: 0.6552125612894694 |Validation err:
Epoch 25: Train err: 0.387, Train loss: 0.6545802884631686 |Validation err: 0.
Epoch 26: Train err: 0.38425, Train loss: 0.6537306110064188 |Validation err:
Epoch 27: Train err: 0.3845, Train loss: 0.652985950311025 |Validation err: 0.
Epoch 28: Train err: 0.38375, Train loss: 0.6523342198795743 |Validation err:
Epoch 29: Train err: 0.38275, Train loss: 0.6515019734700521 |Validation err:
Epoch 30: Train err: 0.38275, Train loss: 0.6511197487513224 |Validation err:
Finished Training
Total time elapsed: 117.21 seconds
```




0.40
0.38
0
5
10
15
20
25
30
Epoch


Train vs Validation Loss
Train
Validation
Loss
0.70
0.69
0.68
0.67
0.66
0.65
0
5
10
15
20
25
30
Epoch

```
net = ANN()
model_path = get_model_name(net.name, batch_size=900, learning_rate=0.001, epoch=
state = torch.load(model_path)
net.load_state_dict(state)

train_loader, val_loader, test_loader, classes = get_data_loader(
    target_classes=["cat", "dog"],
    batch_size=900)

test_err, test_loss = evaluate(net, test_loader, nn.BCEWithLogitsLoss())
print("Test Error: ", test_err, "Test Loss: ", test_loss)
```

```
Files already downloaded and verified
Files already downloaded and verified
Test Error:  0.379 Test Loss:  0.6517025232315063
```

The CNN fares better than the ANN because while the ANN takes less time than the CNN to train, the CNN has a much lower test error and loss than the ANN, reporting better accuracy for the same parameters, despite the ANN being the best possible version of the model. In fact, it is only after ~5 reruns of the test set (as an experiment to see how fast the model stops generalizing) that I was able to get the above test error and loss lower than that of the CNN.