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/1imbjKAqX8SsUPfXE0UioBjAilef-8JUn?usp=sharing

In [ ]:
```
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.

In [ ]:
```
################################################################################
# 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 class
                        Should be a subset of the 'classes'
    Returns:
        indices: list of indices that have labels corresponding to one of th
                 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, valida
    and testing datasets. Returns data loaders for the three preprocessed da

    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
        val_loader: iterable validation dataset organized according to batch
        test_loader: iterable testing dataset organized according to batch s
        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=transfo
    # Get the list of indices to sample from
    relevant_indices = get_relevant_indices(trainset, classes, target_classe

    # Split into train and validation
    np.random.seed(1000) # Fixed numpy random seed for reproducible shufflin
    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],
    train_sampler = SubsetRandomSampler(relevant_train_indices)
    train_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_si
                                               num_workers=1, sampler=train_
    val_sampler = SubsetRandomSampler(relevant_val_indices)
    val_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size
                                             num_workers=1, sampler=val_sam
    # Load CIFAR10 testing data
    testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                           download=True, transform=transfor
    # Get the list of indices to sample from
    relevant_test_indices = get_relevant_indices(testset, classes, target_cl
    test_sampler = SubsetRandomSampler(relevant_test_indices)
    test_loader = torch.utils.data.DataLoader(testset, batch_size=batch_size
                                              num_workers=1, sampler=test_sam
    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 v

    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
         loss: A scalar for the average loss function over the validation se
     """
    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.

In [ ]:
```
# 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
```

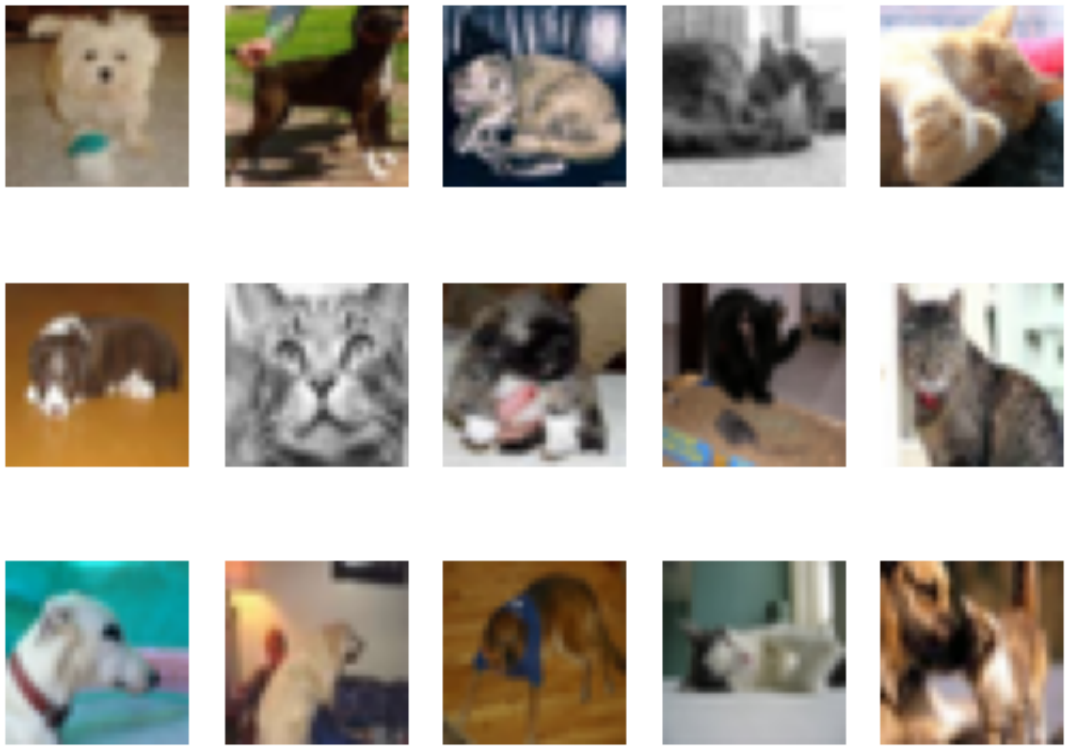
```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./dat
a/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:04<00:00, 35352727.84it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

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.)

In [ ]:
```
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?

In [ ]:
```
print('There are',len(train_loader), 'training examples for the combined cat
print('There are',len(val_loader), 'validation examples for the combined cat
print('There are',len(test_loader), 'test examples for the combined cat and
```

```
There are 8000 training examples for the combined cat and dog classes.
There are 2000 validation examples for the combined cat and dog classes.
There are 2000 test examples for the combined cat and dog classes.
```

Part (c) -- 3pt

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

Answer: We need a validation set when training our model because

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.

In [ ]:
```
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
```

In [ ]:
```
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
```

In [ ]:
```
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?)

In [ ]:
```
for param in small_net.parameters():
    print(param.shape)
```

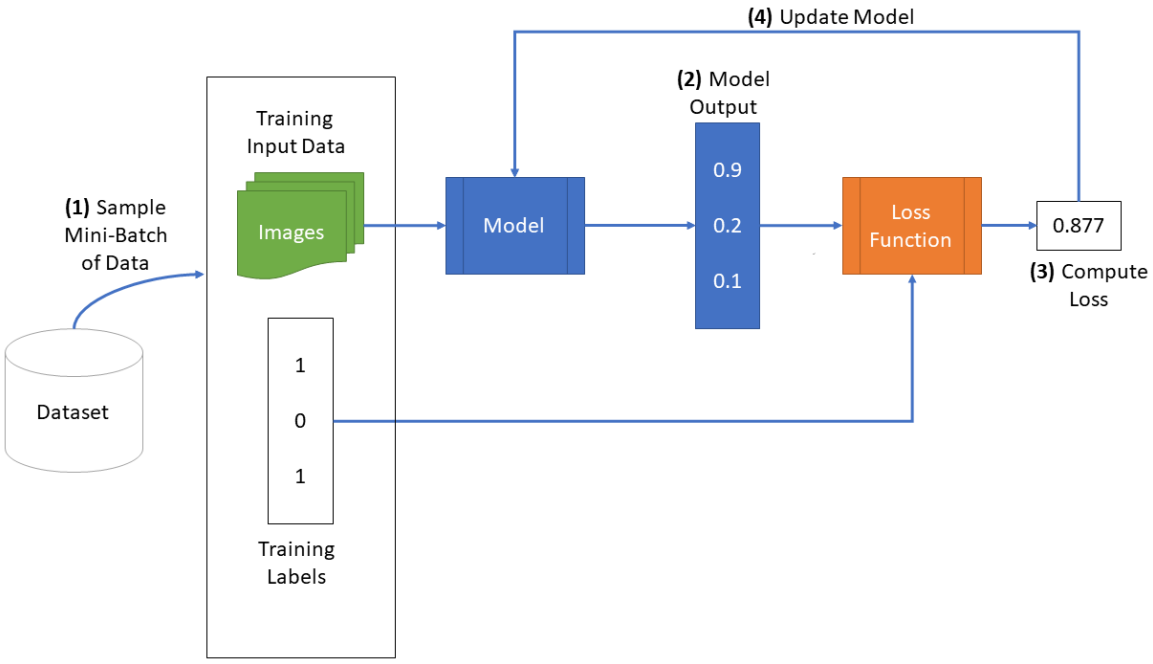
```
torch.Size([5, 3, 3, 3])
torch.Size([5])
torch.Size([1, 245])
torch.Size([1])
```

In [ ]:
```
small_total = 1
small_sum = 0

for param in small_net.parameters():
  for i in range(len(param.shape)):
    small_total *= param.shape[i]
  small_sum += small_total
  small_total = 1

print('There are', small_sum, 'parameters in small_net.')

large_total = 1
large_sum = 0

for param in large_net.parameters():
  for i in range(len(param.shape)):
    large_total *= param.shape[i]
  large_sum += large_total
  large_total = 1
print('There are', large_sum, 'parameters in small_net.')
```

```
There are 386 parameters in small_net.
There are 9705 parameters in small_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:



In [ ]:
```
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, criteric
        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, epc
        torch.save(net.state_dict(), model_path)
    print('Finished Training')
    end_time = time.time()
    elapsed_time = end_time - start_time
    print("Total time elapsed: {:.2f} seconds".format(elapsed_time))
    # Write the train/test loss/err into CSV file for plotting later
    epochs = np.arange(1, num_epochs + 1)
    np.savetxt("{}_train_err.csv".format(model_path), train_err)
    np.savetxt("{}_train_loss.csv".format(model_path), train_loss)
    np.savetxt("{}_val_err.csv".format(model_path), val_err)
    np.savetxt("{}_val_loss.csv".format(model_path), val_loss)
```

Part (b) -- 1pt

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

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

Answer: The default values of batch_size is 64, learning_rate is 0.01, num_epochs is 30 as defined in the first line of the function.

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.

In [ ]:
```
train_net(small_net, num_epochs = 5)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./dat
a/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:01<00:00, 97849366.04it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
Epoch 1: Train err: 0.418625, Train loss: 0.6711218724250794 |Validation err
: 0.372, Validation loss: 0.6551580280065536
Epoch 2: Train err: 0.35675, Train loss: 0.6371686363220215 |Validation err:
0.357, Validation loss: 0.6434999015182257
Epoch 3: Train err: 0.33725, Train loss: 0.6173642740249634 |Validation err:
0.3475, Validation loss: 0.6191077399998903
Epoch 4: Train err: 0.327, Train loss: 0.6018630278110504 |Validation err: 0
.355, Validation loss: 0.6264621298760176
Epoch 5: Train err: 0.3135, Train loss: 0.5922581877708435 |Validation err:
0.32, Validation loss: 0.6174892988055944
Finished Training
Total time elapsed: 14.89 seconds
```

Answer:

Models at 5 checkpoints are saved to 5 different files:

1. **model_small_bs64_lr0.01_epoch0 saved at checkpoint: epoch 0**
2. **model_small_bs64_lr0.01_epoch1 saved at checkpoint: epoch 1**
3. **model_small_bs64_lr0.01_epoch2 saved at checkpoint: epoch 2**
4. **model_small_bs64_lr0.01_epoch3 saved at checkpoint: epoch 3**
5. **model_small_bs64_lr0.01_epoch4 saved at checkpoint: epoch 4**

Also, the training error, training loss, validation error, and validation loss are saved to 4 different csv files for plotting.

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?

In [ ]:
```
# 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
```

In [ ]:
```
small_net = SmallNet()
large_net = LargeNet()

train_net(small_net)
train_net(large_net)

#Time for small_net: 135.90s
#Time for large_net: 151.60s
#Training for small_net takes less time because
#it has less parameter to tune than large_net
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.446375, Train loss: 0.6813716783523559 |Validation err
: 0.3865, Validation loss: 0.6602997500449419
Epoch 2: Train err: 0.37325, Train loss: 0.6497629323005676 |Validation err:
0.3845, Validation loss: 0.6575995869934559
Epoch 3: Train err: 0.359875, Train loss: 0.6388978385925292 |Validation err
: 0.3495, Validation loss: 0.6291275043040514
Epoch 4: Train err: 0.346375, Train loss: 0.6246587996482849 |Validation err
: 0.356, Validation loss: 0.6221408396959305
Epoch 5: Train err: 0.334375, Train loss: 0.6153830280303955 |Validation err
: 0.3275, Validation loss: 0.6188967823982239
Epoch 6: Train err: 0.318, Train loss: 0.6036732516288758 |Validation err: 0
.339, Validation loss: 0.6094125052914023
Epoch 7: Train err: 0.315625, Train loss: 0.5944745948314667 |Validation err
: 0.329, Validation loss: 0.5974238961935043
Epoch 8: Train err: 0.3085, Train loss: 0.5829453563690186 |Validation err:
0.3085, Validation loss: 0.5885121468454599
Epoch 9: Train err: 0.302, Train loss: 0.5805657277107239 |Validation err: 0
.3115, Validation loss: 0.5845186104997993
Epoch 10: Train err: 0.29975, Train loss: 0.573062111377716 |Validation err:
0.309, Validation loss: 0.5785001656040549
Epoch 11: Train err: 0.287375, Train loss: 0.5632161114215851 |Validation er
r: 0.314, Validation loss: 0.5821095015853643
Epoch 12: Train err: 0.292125, Train loss: 0.5567435595989227 |Validation er
r: 0.3115, Validation loss: 0.5860895598307252
Epoch 13: Train err: 0.2885, Train loss: 0.5562505607604981 |Validation err:
0.306, Validation loss: 0.5769414035603404
Epoch 14: Train err: 0.280375, Train loss: 0.5473350758552551 |Validation er
r: 0.3115, Validation loss: 0.5721263345330954
Epoch 15: Train err: 0.285, Train loss: 0.5481121215820313 |Validation err:
0.305, Validation loss: 0.5623639700934291
Epoch 16: Train err: 0.2915, Train loss: 0.5539557900428772 |Validation err:
0.3135, Validation loss: 0.5774335078895092
Epoch 17: Train err: 0.28075, Train loss: 0.5475348830223083 |Validation err
: 0.2995, Validation loss: 0.5680588381364942
Epoch 18: Train err: 0.279625, Train loss: 0.5440063354969025 |Validation er
r: 0.319, Validation loss: 0.576342330314219
Epoch 19: Train err: 0.27575, Train loss: 0.5402116534709931 |Validation err
: 0.3295, Validation loss: 0.606647988781333
Epoch 20: Train err: 0.2715, Train loss: 0.5385935208797454 |Validation err:
0.298, Validation loss: 0.5778946885839105
Epoch 21: Train err: 0.27575, Train loss: 0.540246558189392 |Validation err:
0.302, Validation loss: 0.5672952607274055
Epoch 22: Train err: 0.279, Train loss: 0.5399930019378663 |Validation err:
0.2895, Validation loss: 0.5702174408361316
Epoch 23: Train err: 0.27325, Train loss: 0.5354620461463928 |Validation err
: 0.303, Validation loss: 0.5667499387636781
Epoch 24: Train err: 0.27275, Train loss: 0.5359286315441132 |Validation err
: 0.301, Validation loss: 0.5878297919407487
Epoch 25: Train err: 0.27325, Train loss: 0.5346703794002533 |Validation err
: 0.297, Validation loss: 0.563475382514298
Epoch 26: Train err: 0.27025, Train loss: 0.5316284673213959 |Validation err
```

```
: 0.2985, Validation loss: 0.5694020707160234
Epoch 27: Train err: 0.270375, Train loss: 0.5298305144309997 |Validation er
r: 0.301, Validation loss: 0.578824263997376
Epoch 28: Train err: 0.269625, Train loss: 0.5351403400897979 |Validation er
r: 0.3005, Validation loss: 0.5655373437330127
Epoch 29: Train err: 0.271875, Train loss: 0.5319398436546325 |Validation er
r: 0.2955, Validation loss: 0.5849009975790977
Epoch 30: Train err: 0.270875, Train loss: 0.5373601081371308 |Validation er
r: 0.3175, Validation loss: 0.5815494349226356
Finished Training
Total time elapsed: 135.90 seconds
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.454375, Train loss: 0.6898944606781006 |Validation err
: 0.4205, Validation loss: 0.6793290264904499
Epoch 2: Train err: 0.418875, Train loss: 0.6788924961090088 |Validation err
: 0.4215, Validation loss: 0.6790428329259157
Epoch 3: Train err: 0.40525, Train loss: 0.6685061388015747 |Validation err:
0.3905, Validation loss: 0.6528578028082848
Epoch 4: Train err: 0.379125, Train loss: 0.6512472186088561 |Validation err
: 0.3935, Validation loss: 0.6531118471175432
Epoch 5: Train err: 0.35525, Train loss: 0.6316115870475769 |Validation err:
0.3465, Validation loss: 0.6301526054739952
Epoch 6: Train err: 0.33675, Train loss: 0.6122225694656372 |Validation err:
0.352, Validation loss: 0.6261688079684973
Epoch 7: Train err: 0.3215, Train loss: 0.5984608561992645 |Validation err:
0.3475, Validation loss: 0.6187550257891417
Epoch 8: Train err: 0.31425, Train loss: 0.5851289410591125 |Validation err:
0.3205, Validation loss: 0.6030112085863948
Epoch 9: Train err: 0.3065, Train loss: 0.5784530780315399 |Validation err:
0.3205, Validation loss: 0.5930909719318151
Epoch 10: Train err: 0.29425, Train loss: 0.5654694662094116 |Validation err
: 0.315, Validation loss: 0.5877073928713799
Epoch 11: Train err: 0.28075, Train loss: 0.553060997247696 |Validation err:
0.316, Validation loss: 0.595877917483449
Epoch 12: Train err: 0.278625, Train loss: 0.5416067514419556 |Validation er
r: 0.31, Validation loss: 0.5843141302466393
Epoch 13: Train err: 0.273375, Train loss: 0.5358790538311005 |Validation er
r: 0.2975, Validation loss: 0.5726522607728839
Epoch 14: Train err: 0.2685, Train loss: 0.5234937779903411 |Validation err:
0.2965, Validation loss: 0.5771211478859186
Epoch 15: Train err: 0.260125, Train loss: 0.5139133958816529 |Validation er
r: 0.2955, Validation loss: 0.5627784207463264
Epoch 16: Train err: 0.259375, Train loss: 0.5125633265972137 |Validation er
r: 0.3205, Validation loss: 0.58364431373775
Epoch 17: Train err: 0.24925, Train loss: 0.5033850579261779 |Validation err
: 0.311, Validation loss: 0.5711408788338304
Epoch 18: Train err: 0.248125, Train loss: 0.4913763873577118 |Validation er
r: 0.296, Validation loss: 0.5609989166259766
Epoch 19: Train err: 0.23875, Train loss: 0.4821959674358368 |Validation err
: 0.3045, Validation loss: 0.5786248967051506
Epoch 20: Train err: 0.23125, Train loss: 0.4757481541633606 |Validation err
: 0.287, Validation loss: 0.580998913384974
```

```
Epoch 21: Train err: 0.22525, Train loss: 0.46251006150245666 |Validation er
r: 0.287, Validation loss: 0.5646722186356783
Epoch 22: Train err: 0.220375, Train loss: 0.4519791474342346 |Validation er
r: 0.2855, Validation loss: 0.5754687804728746
Epoch 23: Train err: 0.21375, Train loss: 0.44430874013900756 |Validation er
r: 0.288, Validation loss: 0.5742224156856537
Epoch 24: Train err: 0.207375, Train loss: 0.43036019349098203 |Validation e
rr: 0.309, Validation loss: 0.6110728485509753
Epoch 25: Train err: 0.20175, Train loss: 0.4197188427448273 |Validation err
: 0.2975, Validation loss: 0.5966625260189176
Epoch 26: Train err: 0.191125, Train loss: 0.4100140264034271 |Validation er
r: 0.2995, Validation loss: 0.6168392198160291
Epoch 27: Train err: 0.18725, Train loss: 0.4011841118335724 |Validation err
: 0.3035, Validation loss: 0.6397394668310881
Epoch 28: Train err: 0.177375, Train loss: 0.38736681079864504 |Validation e
rr: 0.3035, Validation loss: 0.614994109608233
Epoch 29: Train err: 0.170375, Train loss: 0.3770212644338608 |Validation er
r: 0.323, Validation loss: 0.7061460921540856
Epoch 30: Train err: 0.166125, Train loss: 0.3615760552883148 |Validation er
r: 0.3015, Validation loss: 0.6578065417706966
Finished Training
Total time elapsed: 151.60 seconds
```

Part (e) - 2pt

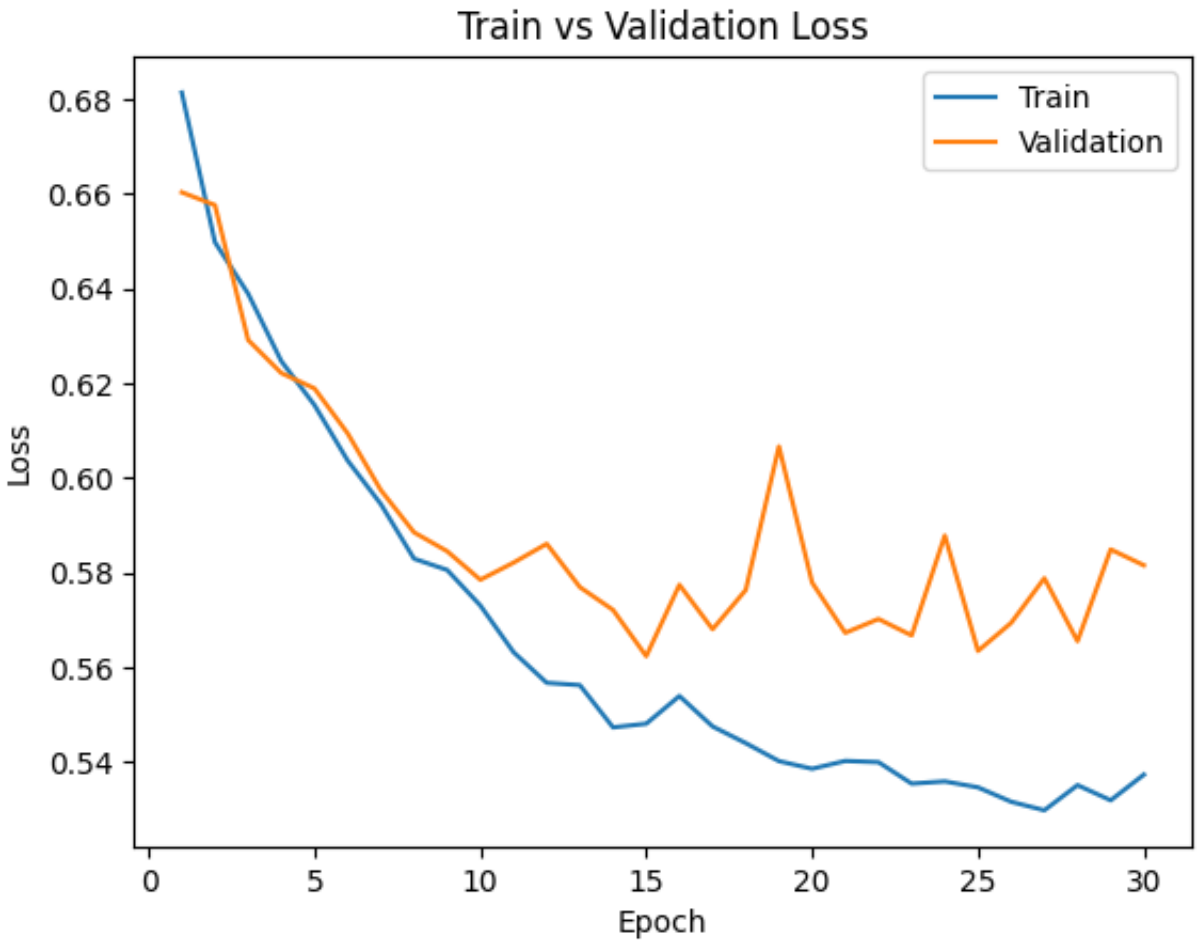
Use the function `plot_training_curve` to display the trajectory of the training/validation error and the training/validation loss. You will need to use the function `get_model_name` to generate the argument to the `plot_training_curve` function.

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

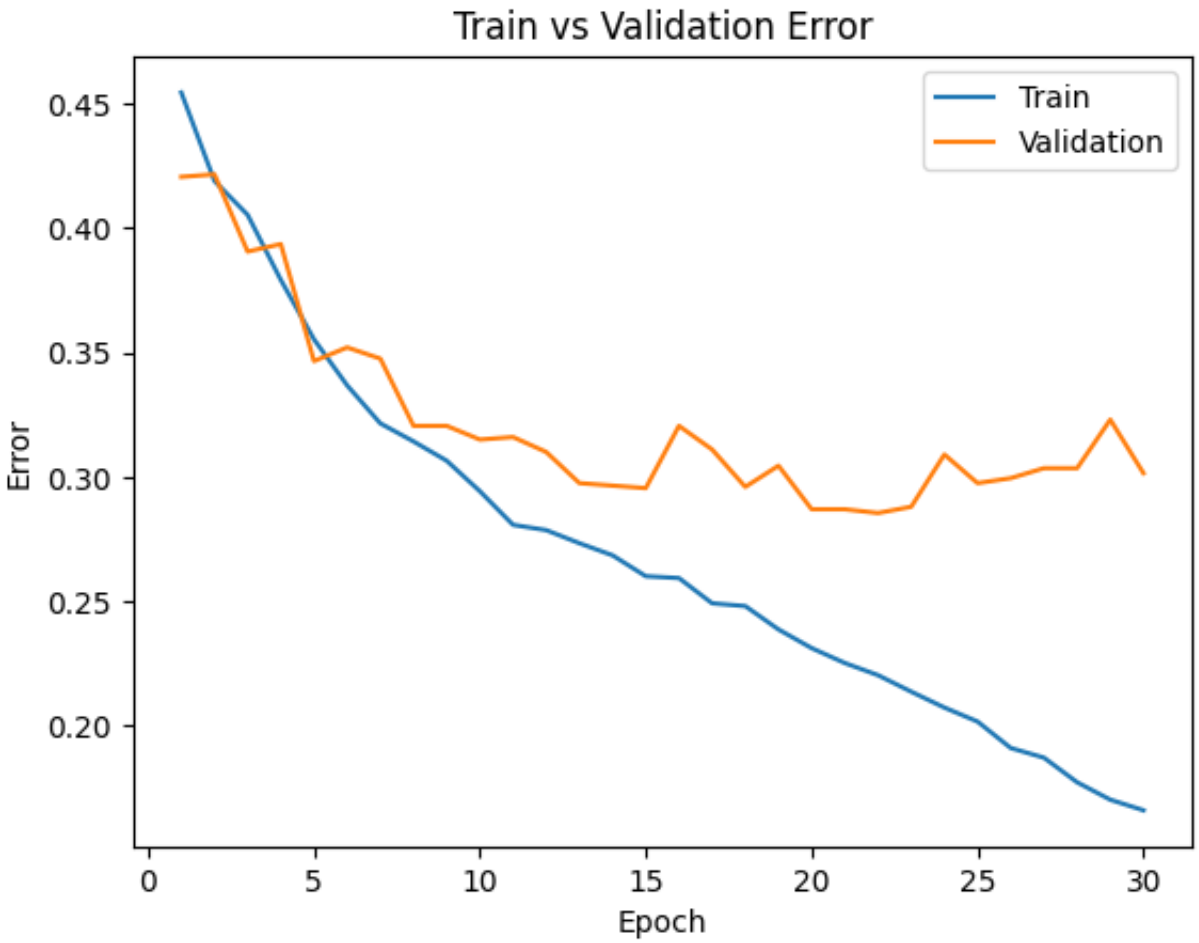
In [ ]:
```
#model_path = get_model_name("small", batch_size=??, learning_rate=??, epoch
print("Small_net training curve:")
small_model_path = get_model_name("small", batch_size = 64,
                                  learning_rate = 0.01, epoch = 29)
plot_training_curve(small_model_path)
print("Large_net training curve:")
large_model_path = get_model_name("large", batch_size = 64,
                                  learning_rate = 0.01, epoch = 29)
plot_training_curve(large_model_path)
```

```
Small_net training curve:
```


Train vs Validation Error
Train
Validation
Error
Epoch
0.450
0.425
0.400
0.375
0.350
0.325
0.300
0.275
0
5
10
15
20
25
30



Large_net training curve:


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

Part (f) - 5pt

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

Answer:

For the small_net model, the model exhibits underfitting at the beginning of the training. Both errors and losses for training and validation data are decreasing as the epoch increases. However, at the end the error and loss for the validation data becomes slightly flat, showing signs of increase, thus the small_net model may overfit if we train it for more epochs.

For the large_net model, the model also exhibits udnerfitting at the beginning of the training with errors and losses for both datasets decreasing. However, at around 15 epochs, the validation data's error and loss stop decreasing and start increase, showing signs of overfitting.

In the end, the small_net has lower error and loss, reaching higher accuracy.

Part 3. Optimization Parameters [12 pt]

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

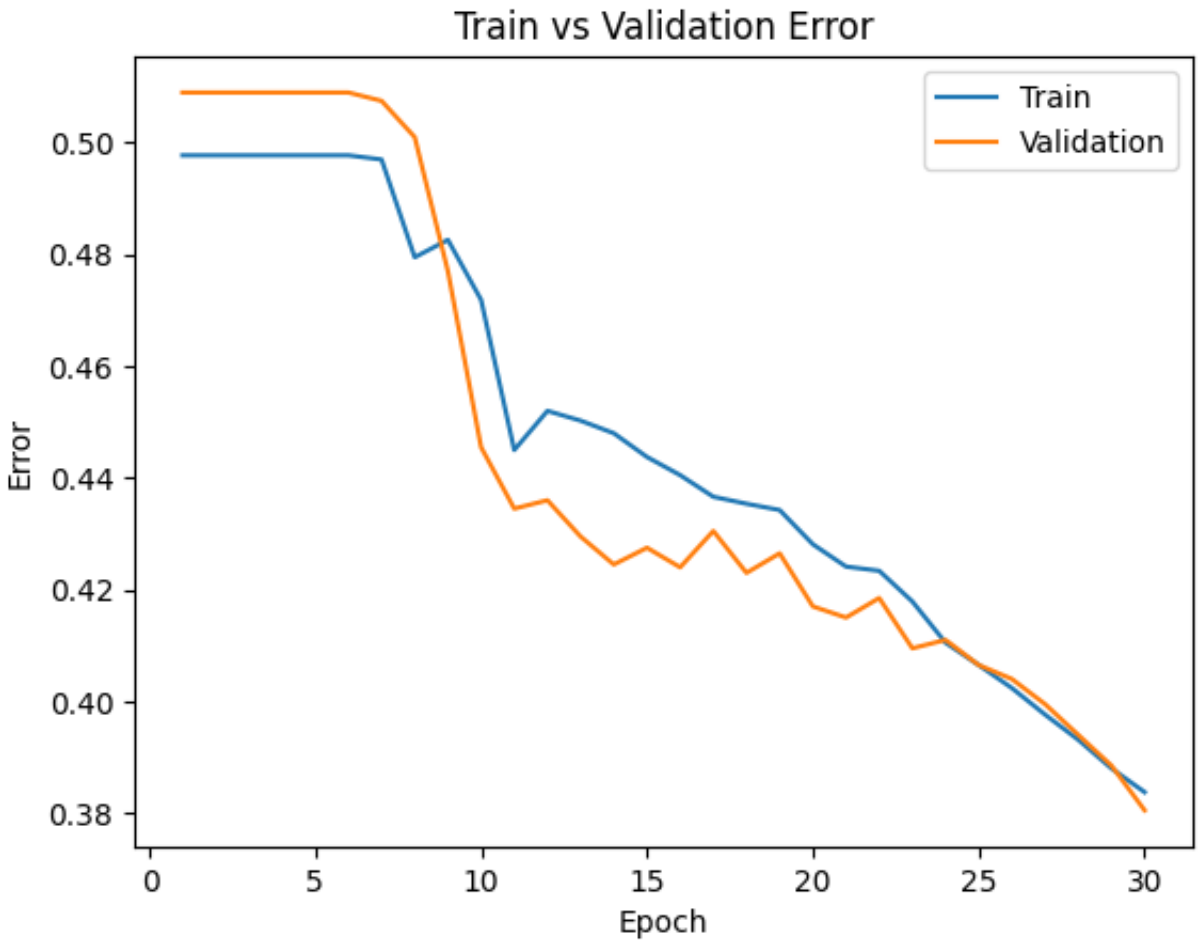
Part (a) - 3pt

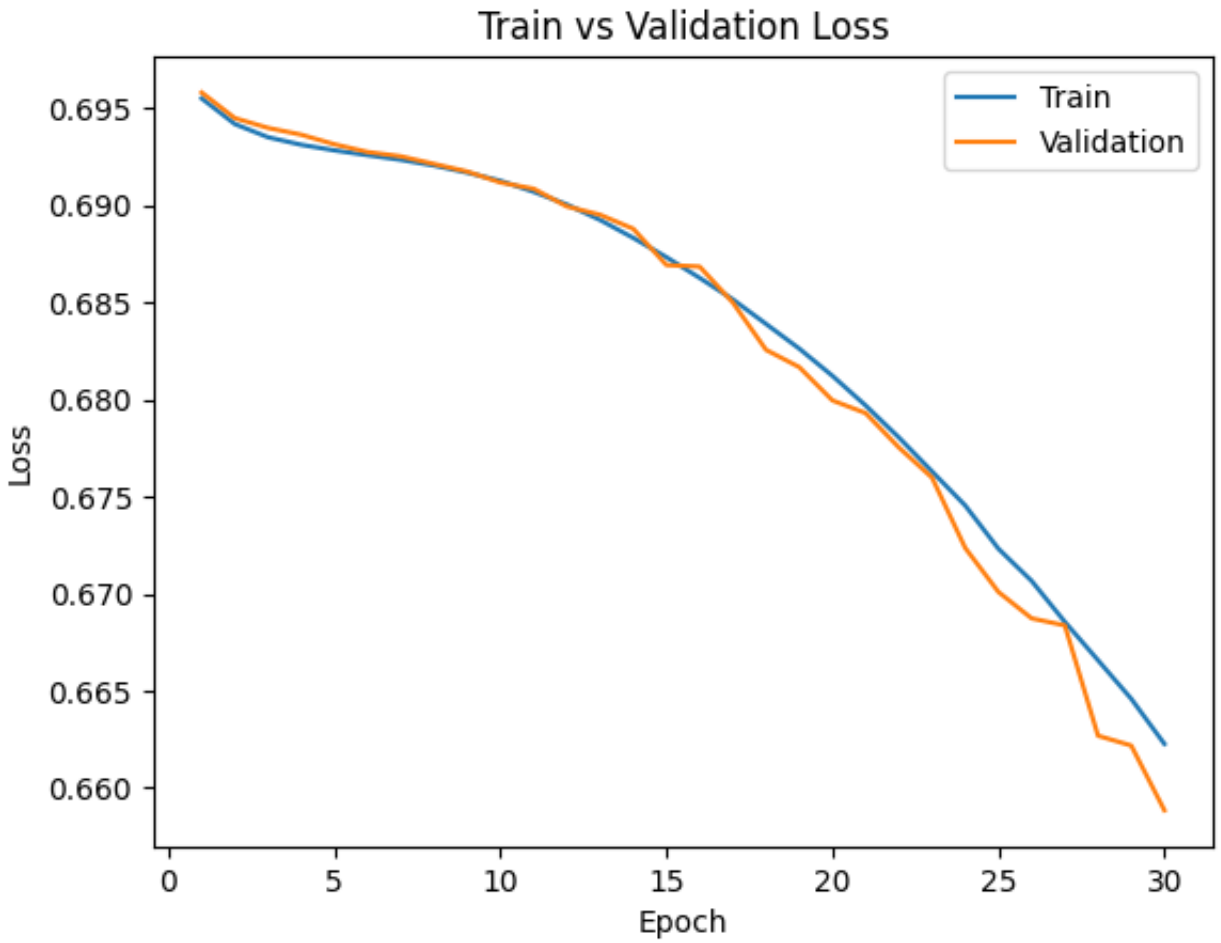
Train `large_net` with all default parameters, except set `learning_rate=0.001`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *lowering* the learning rate.

In [ ]:
```
# Note: When we re-construct the model, we start the training
# with *random weights*. If we omit this code, the values of
# the weights will still be the previously trained values.
large_net = LargeNet()
train_net(large_net, learning_rate = 0.001)
model_path1 = get_model_name("large", batch_size = 64,
                             learning_rate= 0.001, epoch = 29)
plot_training_curve(model_path1)

#Answer:

#The model took longer to train because with lower
#learning rate, the model uses more time to
#optimize the weights.
#According to the curve, there are no sign of
#overfitting, the errors losses for both training
#and validation data are decreasing through all epochs.
#The model reaches a high accuracy in the end
#with the model learn slower but more optimally
#at lower learning rate.
```

```
Files already downloaded and verified
Files already downloaded and verified
Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x
7f5a7f6f2830>
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.
py", line 1478, in __del__
    self._shutdown_workers()
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.
py", line 1461, in _shutdown_workers
    if w.is_alive():
  File "/usr/lib/python3.10/multiprocessing/process.py", line 160, in is_ali
ve
    assert self._parent_pid == os.getpid(), 'can only test a child process'
AssertionError: can only test a child process
Epoch 1: Train err: 0.49775, Train loss: 0.6955151119232178 |Validation err:
0.509, Validation loss: 0.6958169657737017
Epoch 2: Train err: 0.49775, Train loss: 0.6941926474571228 |Validation err:
0.509, Validation loss: 0.6944976653903723
Epoch 3: Train err: 0.49775, Train loss: 0.6935119123458863 |Validation err:
0.509, Validation loss: 0.6939931735396385
Epoch 4: Train err: 0.49775, Train loss: 0.6931246285438537 |Validation err:
0.509, Validation loss: 0.6936436239629984
Epoch 5: Train err: 0.49775, Train loss: 0.6928350310325623 |Validation err:
0.509, Validation loss: 0.6931494977325201
Epoch 6: Train err: 0.49775, Train loss: 0.6925876278877259 |Validation err:
0.509, Validation loss: 0.6927513647824526
Epoch 7: Train err: 0.497, Train loss: 0.6923485560417175 |Validation err: 0
.5075, Validation loss: 0.6925282552838326
Epoch 8: Train err: 0.4795, Train loss: 0.6920534367561341 |Validation err:
0.501, Validation loss: 0.6921367514878511
```

```
Epoch 9: Train err: 0.482625, Train loss: 0.6916898331642151 |Validation err
: 0.477, Validation loss: 0.6917457524687052
Epoch 10: Train err: 0.471875, Train loss: 0.6912688236236573 |Validation er
r: 0.4455, Validation loss: 0.691182030364871
Epoch 11: Train err: 0.445, Train loss: 0.6907034521102905 |Validation err:
0.4345, Validation loss: 0.6908601280301809
Epoch 12: Train err: 0.452, Train loss: 0.6900478286743164 |Validation err:
0.436, Validation loss: 0.6899487059563398
Epoch 13: Train err: 0.45025, Train loss: 0.6892529673576355 |Validation err
: 0.4295, Validation loss: 0.6895196251571178
Epoch 14: Train err: 0.448, Train loss: 0.6883297729492187 |Validation err:
0.4245, Validation loss: 0.6887994688004255
Epoch 15: Train err: 0.44375, Train loss: 0.6873349986076355 |Validation err
: 0.4275, Validation loss: 0.6869303584098816
Epoch 16: Train err: 0.4405, Train loss: 0.6862681117057801 |Validation err:
0.424, Validation loss: 0.6868501417338848
Epoch 17: Train err: 0.436625, Train loss: 0.6851591882705689 |Validation er
r: 0.4305, Validation loss: 0.6849991548806429
Epoch 18: Train err: 0.435375, Train loss: 0.6838960800170898 |Validation er
r: 0.423, Validation loss: 0.6825617998838425
Epoch 19: Train err: 0.43425, Train loss: 0.6826265134811401 |Validation err
: 0.4265, Validation loss: 0.6816726867109537
Epoch 20: Train err: 0.428125, Train loss: 0.6812234616279602 |Validation er
r: 0.417, Validation loss: 0.6799570601433516
Epoch 21: Train err: 0.424125, Train loss: 0.6796952705383301 |Validation er
r: 0.415, Validation loss: 0.6793038621544838
Epoch 22: Train err: 0.423375, Train loss: 0.6780511221885681 |Validation er
r: 0.4185, Validation loss: 0.6775354780256748
Epoch 23: Train err: 0.417875, Train loss: 0.6762975811958313 |Validation er
r: 0.4095, Validation loss: 0.6759823802858591
Epoch 24: Train err: 0.4105, Train loss: 0.6745438990592957 |Validation err:
0.411, Validation loss: 0.6723580192774534
Epoch 25: Train err: 0.4065, Train loss: 0.6723178811073304 |Validation err:
0.4065, Validation loss: 0.6700910720974207
Epoch 26: Train err: 0.402375, Train loss: 0.6706597013473511 |Validation er
r: 0.404, Validation loss: 0.6687308996915817
Epoch 27: Train err: 0.397625, Train loss: 0.6685513954162597 |Validation er
r: 0.3995, Validation loss: 0.6683666333556175
Epoch 28: Train err: 0.393125, Train loss: 0.6665741100311279 |Validation er
r: 0.394, Validation loss: 0.6626860611140728
Epoch 29: Train err: 0.388, Train loss: 0.6645897102355957 |Validation err:
0.3885, Validation loss: 0.662179147824645
Epoch 30: Train err: 0.38375, Train loss: 0.6622627830505371 |Validation err
: 0.3805, Validation loss: 0.6588452607393265
Finished Training
Total time elapsed: 168.90 seconds
```


Train vs Validation Error
Train
Validation
Error
0.50
0.48
0.46
0.44
0.42
0.40
0.38
0
5
10
15
20
25
30
Epoch



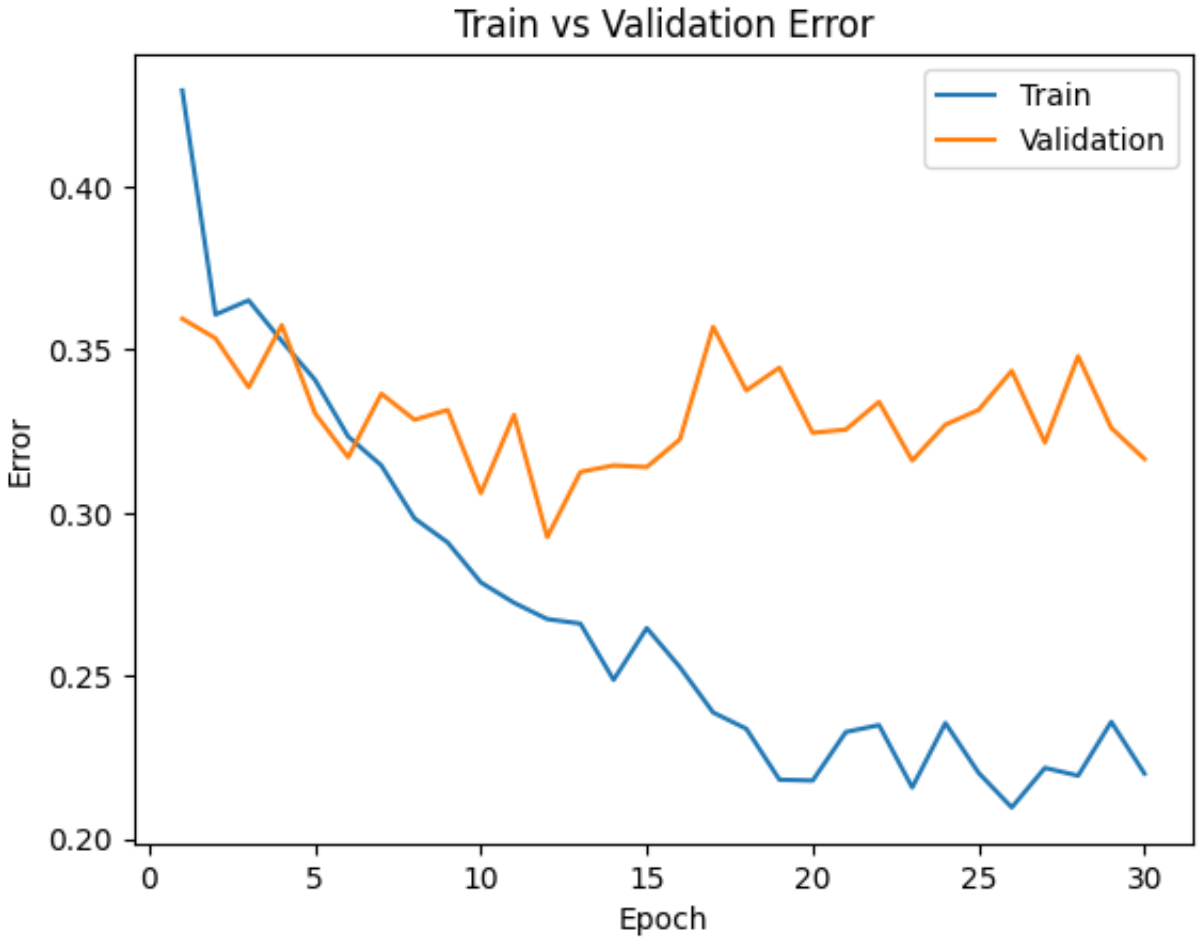
Part (b) - 3pt

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.

In [ ]:
```
train_net(large_net, learning_rate = 0.1)
model_path = get_model_name("large", batch_size = 64,
                            learning_rate= 0.1, epoch = 29)
plot_training_curve(model_path)

#Answer:

#The model took longer to train.
#Looking at the train curve, the model exhibits
#signs of overfitting at early stages in the training
#period. This is because the weights are modified
#too much at each epoch, so the model does not
#converge to its optimal accuracy, where validation
#data's loss is at its lowest. Increasing learning rate
#will let the model to lose accuracy.
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.4295, Train loss: 0.67437779712677 |Validation err: 0.
3595, Validation loss: 0.6350857093930244
Epoch 2: Train err: 0.36075, Train loss: 0.6411805458068848 |Validation err:
0.3535, Validation loss: 0.6361209936439991
Epoch 3: Train err: 0.365125, Train loss: 0.6321813461780548 |Validation err
: 0.3385, Validation loss: 0.6056603882461786
Epoch 4: Train err: 0.352625, Train loss: 0.6233456182479858 |Validation err
: 0.3575, Validation loss: 0.6362800188362598
Epoch 5: Train err: 0.34075, Train loss: 0.6108013873100281 |Validation err:
0.3305, Validation loss: 0.6064918786287308
Epoch 6: Train err: 0.323375, Train loss: 0.5921835997104645 |Validation err
: 0.317, Validation loss: 0.5967769594863057
Epoch 7: Train err: 0.3145, Train loss: 0.5817317583560944 |Validation err:
0.3365, Validation loss: 0.6204487886279821
Epoch 8: Train err: 0.29825, Train loss: 0.5660300073623658 |Validation err:
0.3285, Validation loss: 0.5983372200280428
Epoch 9: Train err: 0.290875, Train loss: 0.552809501171112 |Validation err:
0.3315, Validation loss: 0.6084455158561468
Epoch 10: Train err: 0.278625, Train loss: 0.539032607793808 |Validation err
: 0.306, Validation loss: 0.5918631898239255
Epoch 11: Train err: 0.272375, Train loss: 0.5236025826931 |Validation err:
0.33, Validation loss: 0.6430060230195522
Epoch 12: Train err: 0.267375, Train loss: 0.5220149435997009 |Validation er
r: 0.2925, Validation loss: 0.6413561534136534
Epoch 13: Train err: 0.266, Train loss: 0.5160510110855102 |Validation err:
0.3125, Validation loss: 0.6349832843989134
Epoch 14: Train err: 0.24875, Train loss: 0.4951590054035187 |Validation err
: 0.3145, Validation loss: 0.7193072671070695
Epoch 15: Train err: 0.264625, Train loss: 0.519231944322586 |Validation err
: 0.314, Validation loss: 0.6381420725956559
Epoch 16: Train err: 0.252625, Train loss: 0.5020012385845184 |Validation er
r: 0.3225, Validation loss: 0.6551959458738565
Epoch 17: Train err: 0.23875, Train loss: 0.481714787364006 |Validation err:
0.357, Validation loss: 0.6440742611885071
Epoch 18: Train err: 0.23375, Train loss: 0.47645506453514097 |Validation er
r: 0.3375, Validation loss: 0.6777342790737748
Epoch 19: Train err: 0.218125, Train loss: 0.45134368968009947 |Validation e
rr: 0.3445, Validation loss: 0.7232250478118658
Epoch 20: Train err: 0.217875, Train loss: 0.45516350817680357 |Validation e
rr: 0.3245, Validation loss: 0.6354950983077288
Epoch 21: Train err: 0.23275, Train loss: 0.47897080445289614 |Validation er
r: 0.3255, Validation loss: 0.8348110988736153
Epoch 22: Train err: 0.234875, Train loss: 0.4808810565471649 |Validation er
r: 0.334, Validation loss: 0.7191346418112516
Epoch 23: Train err: 0.21575, Train loss: 0.4563647754192352 |Validation err
: 0.316, Validation loss: 0.7083508176729083
Epoch 24: Train err: 0.2355, Train loss: 0.47718250966072084 |Validation err
: 0.327, Validation loss: 0.7333047650754452
Epoch 25: Train err: 0.22025, Train loss: 0.4583414270877838 |Validation err
: 0.3315, Validation loss: 0.7806987538933754
Epoch 26: Train err: 0.209625, Train loss: 0.4519626965522766 |Validation er
```

```
r: 0.3435, Validation loss: 0.7715998776257038
Epoch 27: Train err: 0.22175, Train loss: 0.4636160457134247 |Validation err
: 0.3215, Validation loss: 0.7656293725594878
Epoch 28: Train err: 0.219375, Train loss: 0.46314777398109436 |Validation e
rr: 0.348, Validation loss: 0.8202023077756166
Epoch 29: Train err: 0.235875, Train loss: 0.49053542733192446 |Validation e
rr: 0.326, Validation loss: 0.8150460105389357
Epoch 30: Train err: 0.22, Train loss: 0.4623157248497009 |Validation err: 0
.3165, Validation loss: 0.7585078496485949
Finished Training
Total time elapsed: 203.63 seconds
```



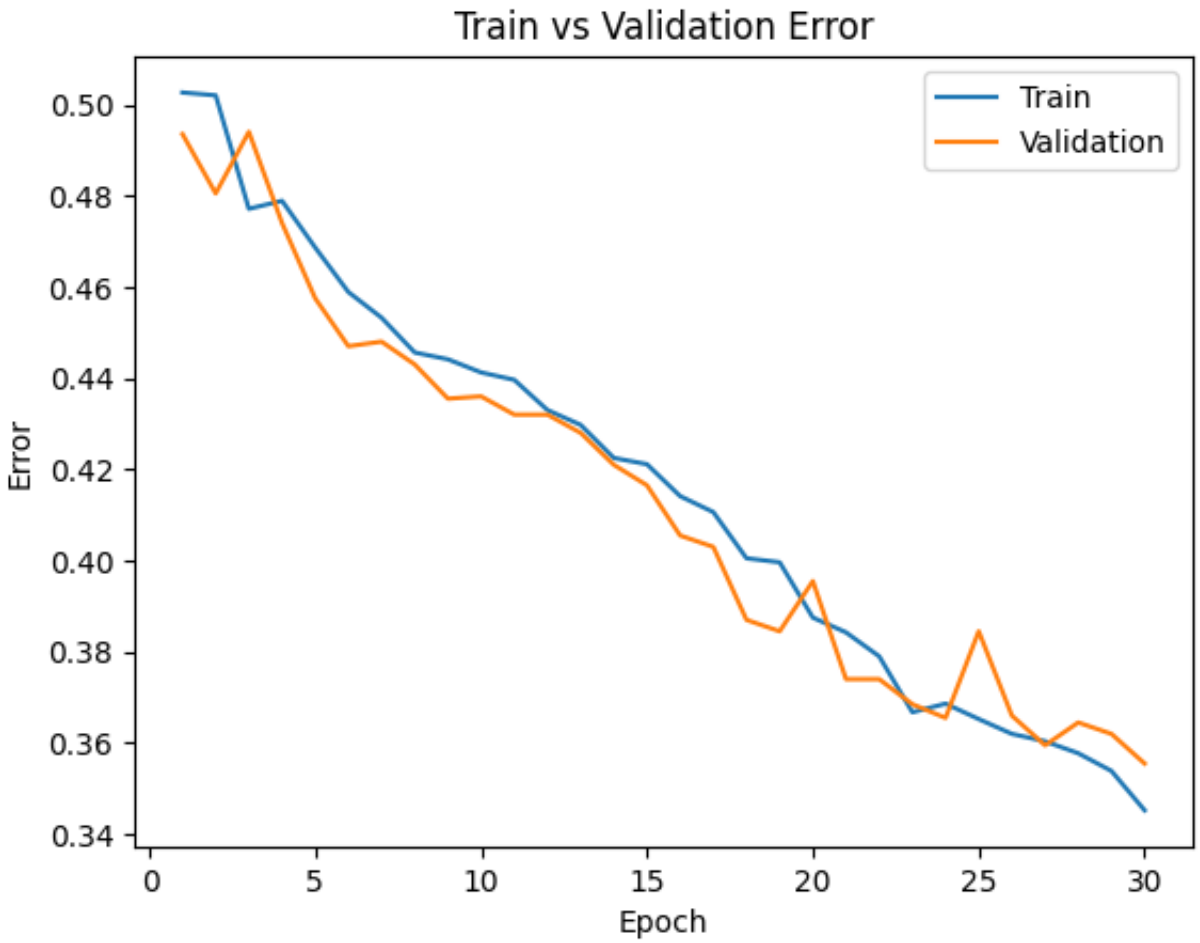
Part (c) - 3pt

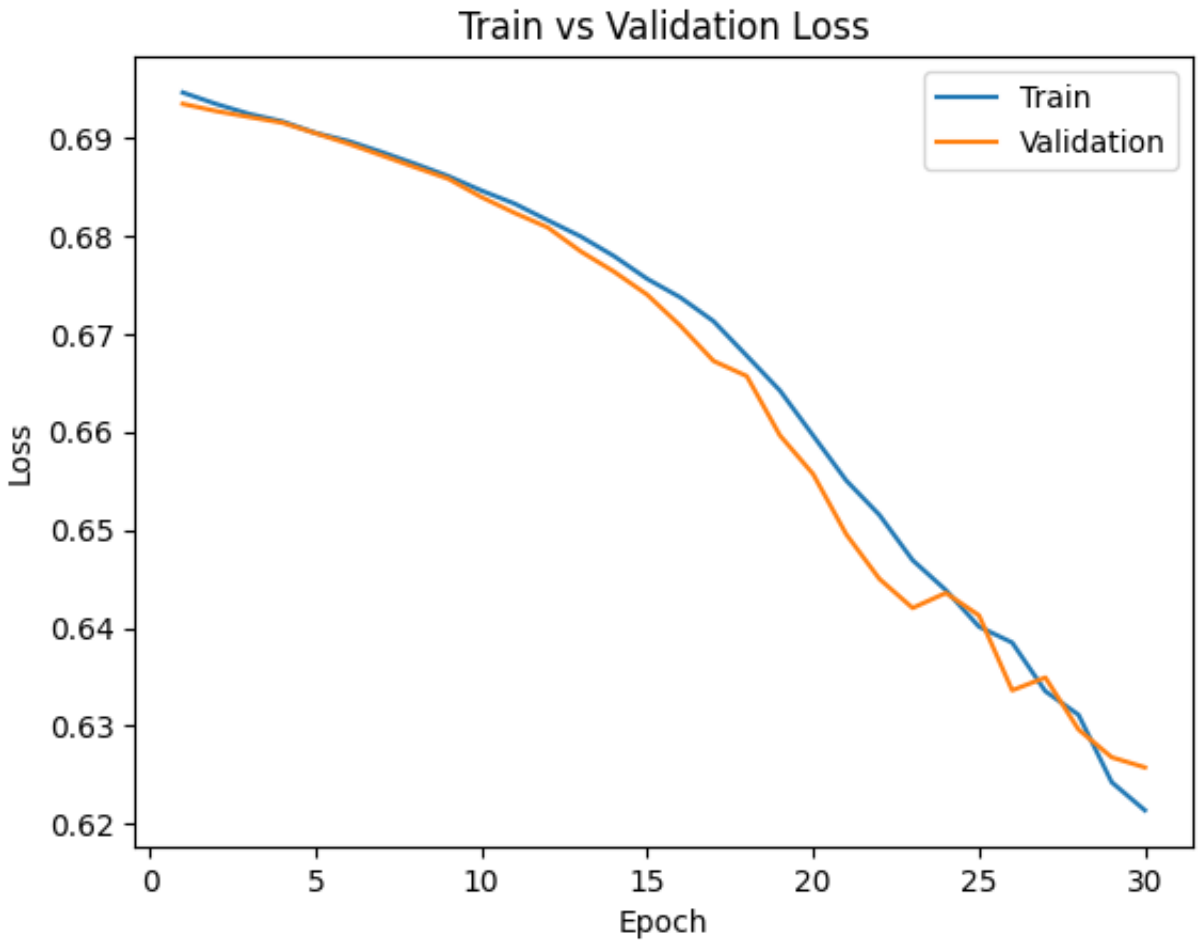
Train `large_net` with all default parameters, including with `learning_rate=0.01`. Now, set `batch_size=512`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *increasing* the batch size.

In [ ]:
```
large_net = LargeNet()
train_net(large_net, batch_size = 512)
model_path2 = get_model_name("large", batch_size = 512,
                             learning_rate = 0.01, epoch = 29)
plot_training_curve(model_path2)

#The model took less time to train because the
#batch size is large, and for each epoch,
#the data is broken into fewer groups, so it took
#less time to train. Compare to the default parameters,
#increasing batch size will make the model have higher
#errors and losses on both training and validation datasets.
#However, even though the large batch size causes the
#model to learn less, likely resulting in loss
#of accuracy in the end, there are no signs of overfitting
#after training.
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.502625, Train loss: 0.694613516330719 |Validation err:
0.4935, Validation loss: 0.693454310297966
Epoch 2: Train err: 0.502, Train loss: 0.6934484839439392 |Validation err: 0
.4805, Validation loss: 0.6927193701267242
Epoch 3: Train err: 0.477125, Train loss: 0.6924241930246353 |Validation err
: 0.494, Validation loss: 0.6921168267726898
Epoch 4: Train err: 0.478875, Train loss: 0.6916632652282715 |Validation err
: 0.474, Validation loss: 0.6915290355682373
Epoch 5: Train err: 0.468625, Train loss: 0.6904933676123619 |Validation err
: 0.4575, Validation loss: 0.6904552429914474
Epoch 6: Train err: 0.458875, Train loss: 0.6896506026387215 |Validation err
: 0.447, Validation loss: 0.689381331205368
Epoch 7: Train err: 0.45325, Train loss: 0.6885304935276508 |Validation err:
0.448, Validation loss: 0.6881940066814423
Epoch 8: Train err: 0.445625, Train loss: 0.6873400807380676 |Validation err
: 0.443, Validation loss: 0.686993733048439
Epoch 9: Train err: 0.444125, Train loss: 0.686088215559721 |Validation err:
0.4355, Validation loss: 0.685817763209343
Epoch 10: Train err: 0.44125, Train loss: 0.6846113316714764 |Validation err
: 0.436, Validation loss: 0.6839505881071091
Epoch 11: Train err: 0.439625, Train loss: 0.6832805164158344 |Validation er
r: 0.432, Validation loss: 0.6823321580886841
Epoch 12: Train err: 0.433, Train loss: 0.6815880872309208 |Validation err:
0.432, Validation loss: 0.680852398276329
Epoch 13: Train err: 0.42975, Train loss: 0.6799208149313927 |Validation err
: 0.428, Validation loss: 0.6783946454524994
Epoch 14: Train err: 0.4225, Train loss: 0.6779238805174828 |Validation err:
0.421, Validation loss: 0.676338717341423
Epoch 15: Train err: 0.421125, Train loss: 0.6756023988127708 |Validation er
r: 0.4165, Validation loss: 0.6739865839481354
Epoch 16: Train err: 0.414125, Train loss: 0.673709973692894 |Validation err
: 0.4055, Validation loss: 0.6708182692527771
Epoch 17: Train err: 0.410625, Train loss: 0.6712822429835796 |Validation er
r: 0.403, Validation loss: 0.66722771525383
```

```
Epoch 18: Train err: 0.4005, Train loss: 0.6677621155977249 |Validation err:
0.387, Validation loss: 0.665687158703804
Epoch 19: Train err: 0.399625, Train loss: 0.664214726537466 |Validation err
: 0.3845, Validation loss: 0.6596384793519974
Epoch 20: Train err: 0.3875, Train loss: 0.6596253775060177 |Validation err:
0.3955, Validation loss: 0.655694454908371
Epoch 21: Train err: 0.38425, Train loss: 0.6550370417535305 |Validation err
: 0.374, Validation loss: 0.649547815322876
Epoch 22: Train err: 0.379, Train loss: 0.6515219211578369 |Validation err:
0.374, Validation loss: 0.6449964195489883
Epoch 23: Train err: 0.36675, Train loss: 0.6469098553061485 |Validation err
: 0.3685, Validation loss: 0.642036646604538
Epoch 24: Train err: 0.368625, Train loss: 0.6438647955656052 |Validation er
r: 0.3655, Validation loss: 0.6435673534870148
Epoch 25: Train err: 0.36525, Train loss: 0.6401287950575352 |Validation err
: 0.3845, Validation loss: 0.6412773281335831
Epoch 26: Train err: 0.362, Train loss: 0.6385088674724102 |Validation err:
0.366, Validation loss: 0.6336372792720795
Epoch 27: Train err: 0.360375, Train loss: 0.6335447728633881 |Validation er
r: 0.3595, Validation loss: 0.6349429935216904
Epoch 28: Train err: 0.35775, Train loss: 0.631140697747469 |Validation err:
0.3645, Validation loss: 0.6296398341655731
Epoch 29: Train err: 0.353875, Train loss: 0.6242633946239948 |Validation er
r: 0.362, Validation loss: 0.6268091648817062
Epoch 30: Train err: 0.34525, Train loss: 0.6214020065963268 |Validation err
: 0.3555, Validation loss: 0.6257748752832413
Finished Training
Total time elapsed: 141.96 seconds
```


Train vs Validation Error
Train
Validation
Error
Epoch
0.50
0.48
0.46
0.44
0.42
0.40
0.38
0.36
0.34
0
5
10
15
20
25
30



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.

In [ ]:
```
large_net = LargeNet()
train_net(large_net, batch_size = 16)
model_path3 = get_model_name("large", batch_size = 16,
                             learning_rate = 0.01, epoch = 29)
plot_training_curve(model_path3)

#Answer:

#The model took longer to train because the training
#dataset is divied into smaller group with a batch size
#of 16 (each group has 16 training data elements). Therefore,
#for each epoch, the model is trained with a larger number
#of data groups, taking more time. Smaller batch size allow the
#model to have better accuracy on the training data. However,
#compare to the default parameter, the error and loss on validation
#data was about the same.Also, since model learned so well
#on the training data due to small batch size,
#overfitting appeared at early stages of the training.
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.43175, Train loss: 0.6774994022846222 |Validation err:
0.382, Validation loss: 0.6513170118331909
Epoch 2: Train err: 0.369, Train loss: 0.639639899969101 |Validation err: 0.
3465, Validation loss: 0.6161113576889038
Epoch 3: Train err: 0.34375, Train loss: 0.6098222947120666 |Validation err:
0.3325, Validation loss: 0.6260210764408112
Epoch 4: Train err: 0.314375, Train loss: 0.5849691489338875 |Validation err
: 0.34, Validation loss: 0.6044013917446136
Epoch 5: Train err: 0.301125, Train loss: 0.5689119303822517 |Validation err
: 0.3125, Validation loss: 0.576918310880661
Epoch 6: Train err: 0.281, Train loss: 0.5452213581204415 |Validation err: 0
.308, Validation loss: 0.5708447456359863
Epoch 7: Train err: 0.270875, Train loss: 0.5272981298565864 |Validation err
: 0.307, Validation loss: 0.5854293291568756
Epoch 8: Train err: 0.259375, Train loss: 0.5070905526578426 |Validation err
: 0.313, Validation loss: 0.5877130818367005
Epoch 9: Train err: 0.242375, Train loss: 0.4968344421982765 |Validation err
: 0.313, Validation loss: 0.5922425072193146
Epoch 10: Train err: 0.236375, Train loss: 0.4756101597249508 |Validation er
r: 0.297, Validation loss: 0.5718690166473389
Epoch 11: Train err: 0.222125, Train loss: 0.4599769461452961 |Validation er
r: 0.2975, Validation loss: 0.6376970833539963
Epoch 12: Train err: 0.211, Train loss: 0.4454492371380329 |Validation err:
0.2995, Validation loss: 0.609202565908432
Epoch 13: Train err: 0.19875, Train loss: 0.4245421719551086 |Validation err
: 0.3075, Validation loss: 0.6494987765550614
Epoch 14: Train err: 0.18675, Train loss: 0.4007472907453775 |Validation err
: 0.3085, Validation loss: 0.6610016552209854
Epoch 15: Train err: 0.1645, Train loss: 0.3759974058121443 |Validation err:
0.3105, Validation loss: 0.7106090537309646
Epoch 16: Train err: 0.16125, Train loss: 0.3591455406397581 |Validation err
: 0.3005, Validation loss: 0.7310364942550659
```

```
Epoch 17: Train err: 0.15775, Train loss: 0.3463234790861607 |Validation err
: 0.307, Validation loss: 0.7263009325265884
Epoch 18: Train err: 0.141625, Train loss: 0.32175366275012496 |Validation e
rr: 0.3195, Validation loss: 0.7913952842950821
Epoch 19: Train err: 0.13375, Train loss: 0.30618105667084455 |Validation er
r: 0.335, Validation loss: 0.8032052783966065
Epoch 20: Train err: 0.126625, Train loss: 0.3029071792438626 |Validation er
r: 0.32, Validation loss: 0.8106685240268707
Epoch 21: Train err: 0.12025, Train loss: 0.28682796490937473 |Validation er
r: 0.3205, Validation loss: 0.8259474284648896
Epoch 22: Train err: 0.1165, Train loss: 0.27489088076353074 |Validation err
: 0.352, Validation loss: 0.8937610774040222
Epoch 23: Train err: 0.104375, Train loss: 0.2467898527495563 |Validation er
r: 0.3315, Validation loss: 1.0021928198337555
Epoch 24: Train err: 0.101, Train loss: 0.23970085787773132 |Validation err:
0.331, Validation loss: 1.1290796399116516
Epoch 25: Train err: 0.09575, Train loss: 0.23643119425699116 |Validation er
r: 0.3315, Validation loss: 1.1338514368534087
Epoch 26: Train err: 0.094125, Train loss: 0.2325953512713313 |Validation er
r: 0.3365, Validation loss: 1.1414263204336166
Epoch 27: Train err: 0.08425, Train loss: 0.21040759468451142 |Validation er
r: 0.3335, Validation loss: 1.1823678107261657
Epoch 28: Train err: 0.0825, Train loss: 0.20643112615589052 |Validation err
: 0.323, Validation loss: 1.266836181640625
Epoch 29: Train err: 0.0845, Train loss: 0.21273409337876364 |Validation err
: 0.3245, Validation loss: 1.406717705130577
Epoch 30: Train err: 0.071375, Train loss: 0.18387044295761734 |Validation e
rr: 0.345, Validation loss: 1.4871552000045776
Finished Training
Total time elapsed: 215.22 seconds
```


Train vs Validation Error
Train
Validation
Error
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
5
10
15
20
25
30
Epoch

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.

In [ ]:
```
#The set of parameter I chose is:

#network: Large_net, because large_net has two filters,
#it can capture higher level features that distinguish
#cat and dog.

#bacth_size: 128, because 64 is still a little bit
#slow for large_net to train, and having a large batch
#size may prevent the model's focus on learning too much,
#which result in overfitting. Although a batch size
#of 512 is a good fit as we tested in part 3(c), it will
#result in a higher validation error and loss.

#learning_rate: 0.005, the 0.001 learning rate
#performed very well on the model in part 3, but
#consumed too much time on tunning the weights,
#so I decide to increase it.
```

Part (b) - 1pt

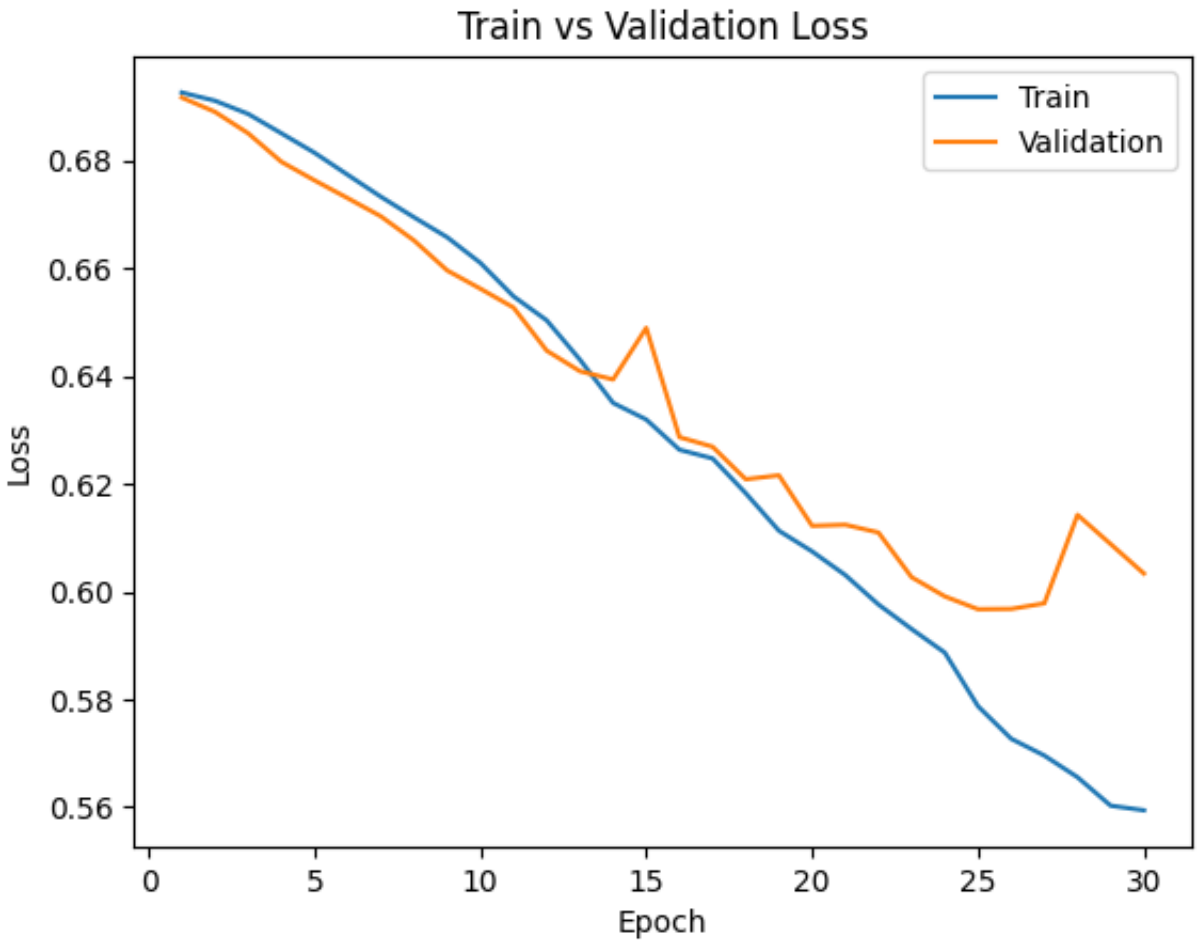
Train the model with the hyperparameters you chose in part(a), and include the training curve.

In [ ]:
```
large_net = LargeNet()
train_net(large_net, batch_size = 128, learning_rate = 0.005)
my_model_path = get_model_name("large", batch_size = 128,
                               learning_rate = 0.005, epoch = 29)
plot_training_curve(my_model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.466625, Train loss: 0.6925613236805749 |Validation err
: 0.4305, Validation loss: 0.6916250362992287
Epoch 2: Train err: 0.45025, Train loss: 0.6910346017943488 |Validation err:
0.4295, Validation loss: 0.6889704614877701
Epoch 3: Train err: 0.4305, Train loss: 0.6885886050405956 |Validation err:
0.4165, Validation loss: 0.6850010603666306
Epoch 4: Train err: 0.430625, Train loss: 0.6850024423901997 |Validation err
: 0.4125, Validation loss: 0.6797147057950497
Epoch 5: Train err: 0.420875, Train loss: 0.6813881851377941 |Validation err
: 0.411, Validation loss: 0.6762721762061119
Epoch 6: Train err: 0.415875, Train loss: 0.6773001438095456 |Validation err
: 0.4125, Validation loss: 0.672969501465559
Epoch 7: Train err: 0.405, Train loss: 0.6732292080682422 |Validation err: 0
.4055, Validation loss: 0.6696001030504704
Epoch 8: Train err: 0.4, Train loss: 0.6694153556748043 |Validation err: 0.4
02, Validation loss: 0.6651207581162453
Epoch 9: Train err: 0.3905, Train loss: 0.6657110272891937 |Validation err:
0.3965, Validation loss: 0.6595559529960155
Epoch 10: Train err: 0.38475, Train loss: 0.6609666309659443 |Validation err
```

```
: 0.398, Validation loss: 0.6561728976666927
Epoch 11: Train err: 0.374625, Train loss: 0.6547484369505019 |Validation er
r: 0.3895, Validation loss: 0.6526741832494736
Epoch 12: Train err: 0.379125, Train loss: 0.6503025397421822 |Validation er
r: 0.382, Validation loss: 0.644683513790369
Epoch 13: Train err: 0.3685, Train loss: 0.6430244833704025 |Validation err:
0.377, Validation loss: 0.6408736705780029
Epoch 14: Train err: 0.35375, Train loss: 0.634985539648268 |Validation err:
0.3715, Validation loss: 0.6393358670175076
Epoch 15: Train err: 0.35375, Train loss: 0.6319066002255395 |Validation err
: 0.385, Validation loss: 0.6489016860723495
Epoch 16: Train err: 0.35125, Train loss: 0.6263072131172059 |Validation err
: 0.357, Validation loss: 0.6286376938223839
Epoch 17: Train err: 0.349875, Train loss: 0.6246873547160436 |Validation er
r: 0.351, Validation loss: 0.6268594488501549
Epoch 18: Train err: 0.343, Train loss: 0.6182309228276449 |Validation err:
0.3415, Validation loss: 0.6208269745111465
Epoch 19: Train err: 0.33675, Train loss: 0.6112786872046334 |Validation err
: 0.348, Validation loss: 0.6215575821697712
Epoch 20: Train err: 0.334875, Train loss: 0.6074399777821132 |Validation er
r: 0.3385, Validation loss: 0.6121908687055111
Epoch 21: Train err: 0.328375, Train loss: 0.603059540665339 |Validation err
: 0.337, Validation loss: 0.6123869866132736
Epoch 22: Train err: 0.327375, Train loss: 0.5975976425503927 |Validation er
r: 0.333, Validation loss: 0.6109451837837696
Epoch 23: Train err: 0.322375, Train loss: 0.5930433736907111 |Validation er
r: 0.3245, Validation loss: 0.6026575490832329
Epoch 24: Train err: 0.317875, Train loss: 0.5886693076481895 |Validation er
r: 0.327, Validation loss: 0.5991024523973465
Epoch 25: Train err: 0.306375, Train loss: 0.5787286261717478 |Validation er
r: 0.315, Validation loss: 0.596717856824398
Epoch 26: Train err: 0.30725, Train loss: 0.572699801316337 |Validation err:
0.321, Validation loss: 0.5967751257121563
Epoch 27: Train err: 0.302875, Train loss: 0.5695816690013522 |Validation er
r: 0.3185, Validation loss: 0.5978134162724018
Epoch 28: Train err: 0.300875, Train loss: 0.5655642464047387 |Validation er
r: 0.334, Validation loss: 0.6142176277935505
Epoch 29: Train err: 0.294125, Train loss: 0.5602825222507356 |Validation er
r: 0.3275, Validation loss: 0.6087971664965153
Epoch 30: Train err: 0.291875, Train loss: 0.5594303224767957 |Validation er
r: 0.328, Validation loss: 0.6033484600484371
Finished Training
Total time elapsed: 150.55 seconds
```


Train vs Validation Error
Train
Validation
Error
Epoch
0.475
0.450
0.425
0.400
0.375
0.350
0.325
0.300
0
5
10
15
20
25
30



Part (c) - 2pt

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

In [ ]:
```
#Another set of hyperparamter I would try is:

#network: Large_net, because large_net
#has two filters, it can capture
#higher level features that distinguish cat and dog.

#bacth_size: 128, staying the same, because
#it seems to be a good fit.

#learning_rate: 0.007, because the model seems to
#have similar training time as the default
#parameter. Thus, I would increase the learning
#rate to see if the time reduces.
```

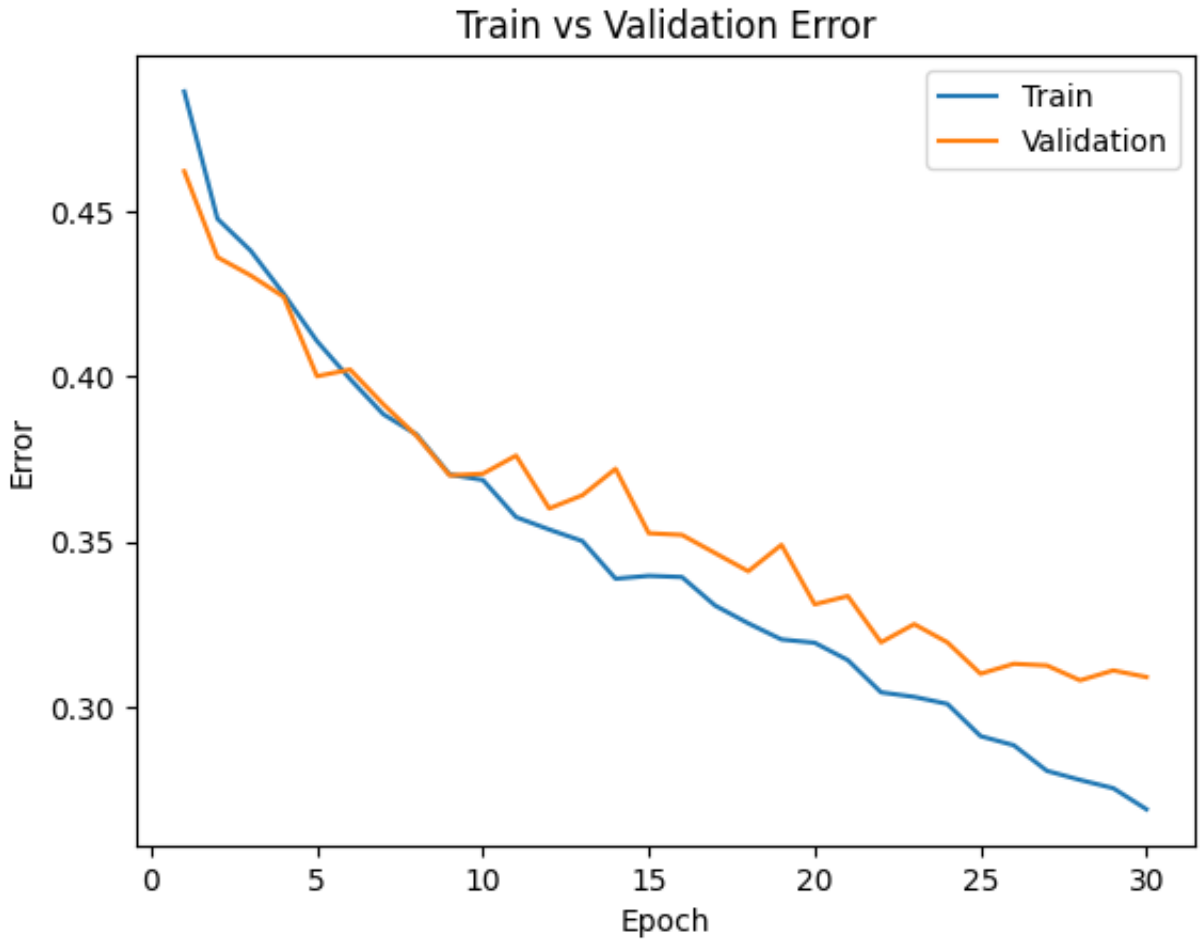
Part (d) - 1pt

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

In [ ]:
```
large_net = LargeNet()
train_net(large_net, batch_size = 128, learning_rate = 0.007)
my_model_path2 = get_model_name("large", batch_size = 128,
                                learning_rate = 0.007, epoch = 29)
plot_training_curve(my_model_path2)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.486125, Train loss: 0.6929923277052622 |Validation err
: 0.462, Validation loss: 0.691496804356575
Epoch 2: Train err: 0.447625, Train loss: 0.6898457143041823 |Validation err
: 0.436, Validation loss: 0.6869067177176476
Epoch 3: Train err: 0.438, Train loss: 0.6855275839094132 |Validation err: 0
.4305, Validation loss: 0.6815641149878502
Epoch 4: Train err: 0.424875, Train loss: 0.6788790330054268 |Validation err
: 0.424, Validation loss: 0.6740341857075691
Epoch 5: Train err: 0.410625, Train loss: 0.6723156069952344 |Validation err
: 0.4, Validation loss: 0.6672771535813808
Epoch 6: Train err: 0.399125, Train loss: 0.664779712283422 |Validation err:
0.402, Validation loss: 0.6627656742930412
Epoch 7: Train err: 0.3885, Train loss: 0.6593310568067763 |Validation err:
0.3915, Validation loss: 0.6555960960686207
Epoch 8: Train err: 0.382375, Train loss: 0.6516290004291232 |Validation err
: 0.382, Validation loss: 0.6483094692230225
Epoch 9: Train err: 0.37025, Train loss: 0.6463586432593209 |Validation err:
0.37, Validation loss: 0.6411598920822144
Epoch 10: Train err: 0.368625, Train loss: 0.6428424933600048 |Validation er
r: 0.3705, Validation loss: 0.6401523053646088
Epoch 11: Train err: 0.357375, Train loss: 0.6348766939980643 |Validation er
r: 0.376, Validation loss: 0.6367527991533279
Epoch 12: Train err: 0.353625, Train loss: 0.6284563569795518 |Validation er
r: 0.36, Validation loss: 0.6303407028317451
Epoch 13: Train err: 0.350125, Train loss: 0.6261693390588912 |Validation er
r: 0.364, Validation loss: 0.6261237040162086
Epoch 14: Train err: 0.33875, Train loss: 0.6181833280457391 |Validation err
: 0.372, Validation loss: 0.6287984065711498
Epoch 15: Train err: 0.339625, Train loss: 0.6148951734815326 |Validation er
r: 0.3525, Validation loss: 0.6365647688508034
Epoch 16: Train err: 0.33925, Train loss: 0.6088614454345097 |Validation err
: 0.352, Validation loss: 0.6181153208017349
Epoch 17: Train err: 0.330625, Train loss: 0.6039005631492251 |Validation er
r: 0.3465, Validation loss: 0.6129559874534607
Epoch 18: Train err: 0.32525, Train loss: 0.5966078733641004 |Validation err
: 0.341, Validation loss: 0.6029507741332054
Epoch 19: Train err: 0.320375, Train loss: 0.5881873416522193 |Validation er
r: 0.349, Validation loss: 0.6267595551908016
```

```
Epoch 20: Train err: 0.319375, Train loss: 0.5842690155619666 |Validation er
r: 0.331, Validation loss: 0.6079521216452122
Epoch 21: Train err: 0.314125, Train loss: 0.5813383024836344 |Validation er
r: 0.3335, Validation loss: 0.5973181687295437
Epoch 22: Train err: 0.304375, Train loss: 0.5732384598444379 |Validation er
r: 0.3195, Validation loss: 0.5890955664217472
Epoch 23: Train err: 0.303, Train loss: 0.5689865445333814 |Validation err:
0.325, Validation loss: 0.5860350355505943
Epoch 24: Train err: 0.300875, Train loss: 0.5666398836506737 |Validation er
r: 0.3195, Validation loss: 0.5924878232181072
Epoch 25: Train err: 0.291125, Train loss: 0.5538880432408954 |Validation er
r: 0.31, Validation loss: 0.5819087289273739
Epoch 26: Train err: 0.288375, Train loss: 0.5506627252177586 |Validation er
r: 0.313, Validation loss: 0.5807468295097351
Epoch 27: Train err: 0.280625, Train loss: 0.54737716865918 |Validation err:
0.3125, Validation loss: 0.5856056362390518
Epoch 28: Train err: 0.277875, Train loss: 0.5383775688353039 |Validation er
r: 0.308, Validation loss: 0.5820190422236919
Epoch 29: Train err: 0.275375, Train loss: 0.5328794949584537 |Validation er
r: 0.311, Validation loss: 0.584551926702261
Epoch 30: Train err: 0.269, Train loss: 0.5298422934517028 |Validation err:
0.309, Validation loss: 0.5883184187114239
Finished Training
Total time elapsed: 151.98 seconds
```



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`.

In [ ]:
```
net = large_net
model_path = get_model_name('large', batch_size=128,
                            learning_rate=0.007, epoch=29)
state = torch.load(model_path)
net.load_state_dict(state)
```

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

Part (b) - 2pt

Justify your choice of model from part (a).

In [ ]:
```
#network: Large_net: because large_net has
#two filters, it can capture higher level
#features that distinguish cat and dog than small_net.

#batch_size: 128, learning_rate : 0.007,
#and epoch: 29, they provide the best validation
#error: 0.309, and loss: 0.588 among all tested
#hyperparameter sets.
#Even though validation error is slightly higher
#than the default parameter, the validation
#loss is significantly lower.
```

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.

In [ ]:
```
# 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=128)

criterion = nn.BCEWithLogitsLoss()
error, loss = evaluate(net, test_loader, criterion)
print('The test classification error is:', error)
print('The test classfication loss is:',loss)
```

```
Files already downloaded and verified
Files already downloaded and verified
The test classification error is: 0.3135
The test classfication loss is: 0.5896048024296761
```

Part (d) - 3pt

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

Answer: Compare the validation error: 0.309, the test classification error is similar but a little higher: 0.3135. This is expected because during the training and hyperparameter testings, the model has seen the validation dataset many time, so its prediction will be slightly accurate for the validation data. While, the model's prediction on new dataset -- the test data, which has only been shown the model once, may have a slightly higher error.

Part (e) - 2pt

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

Answer: the reason why we use the test data in the very end is because it is used to provide an unbiased accuracy of our model. Unlike the validation set, the test data should only be shown to the model in the end and as fewer times as possible, so that it can give us a realistic prerdiction and the accuracy of the model on the objectives we try to achieve in real world scenario, in this case, the objective is to accurately classifiying cats and dogs.

Part (f) - 5pt

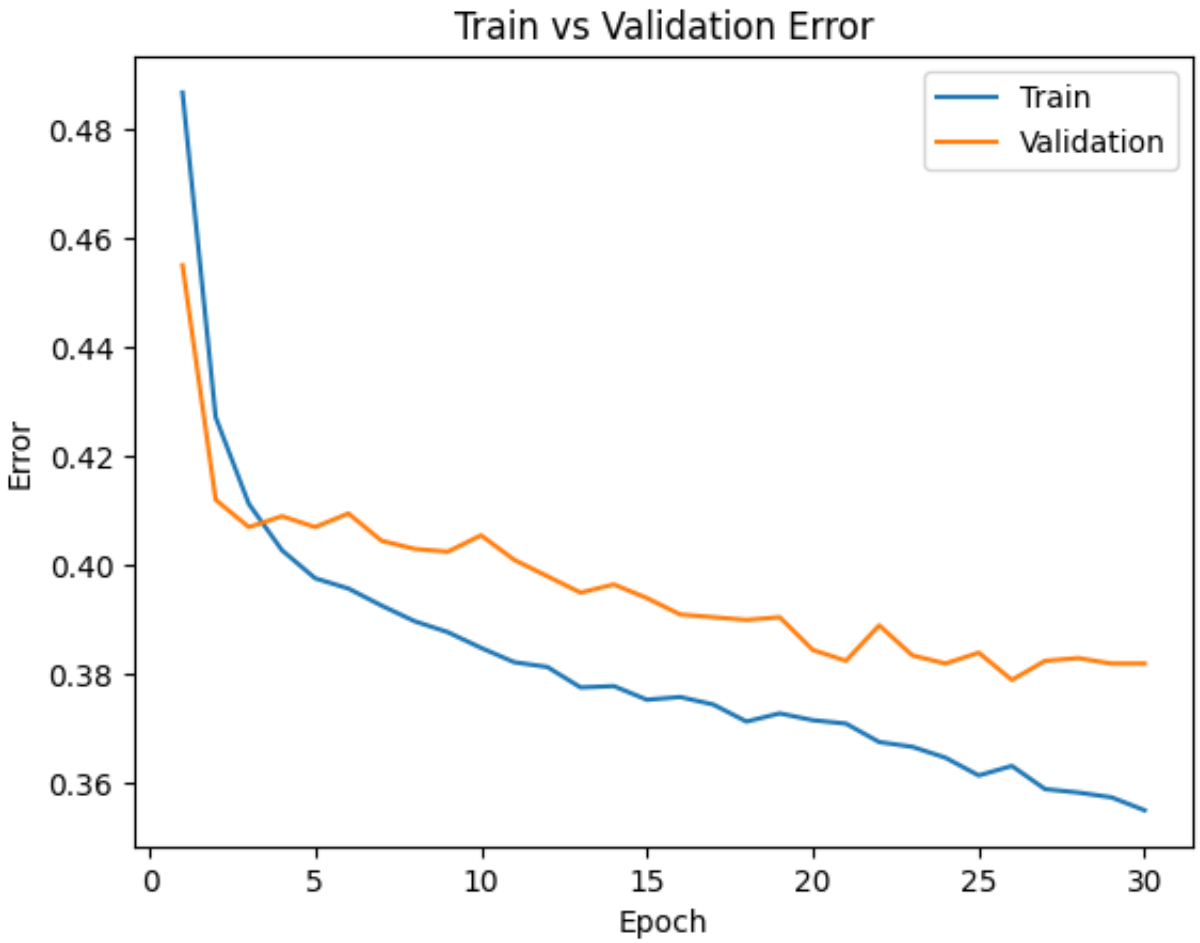
How does the your best CNN model compare with an 2-layer ANN model (no convolutional layers) on classifying cat and dog images. You can use a 2-layer ANN architecture similar to what you used in Lab 1. You should explore different hyperparameter settings to determine how well you can do on the validation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

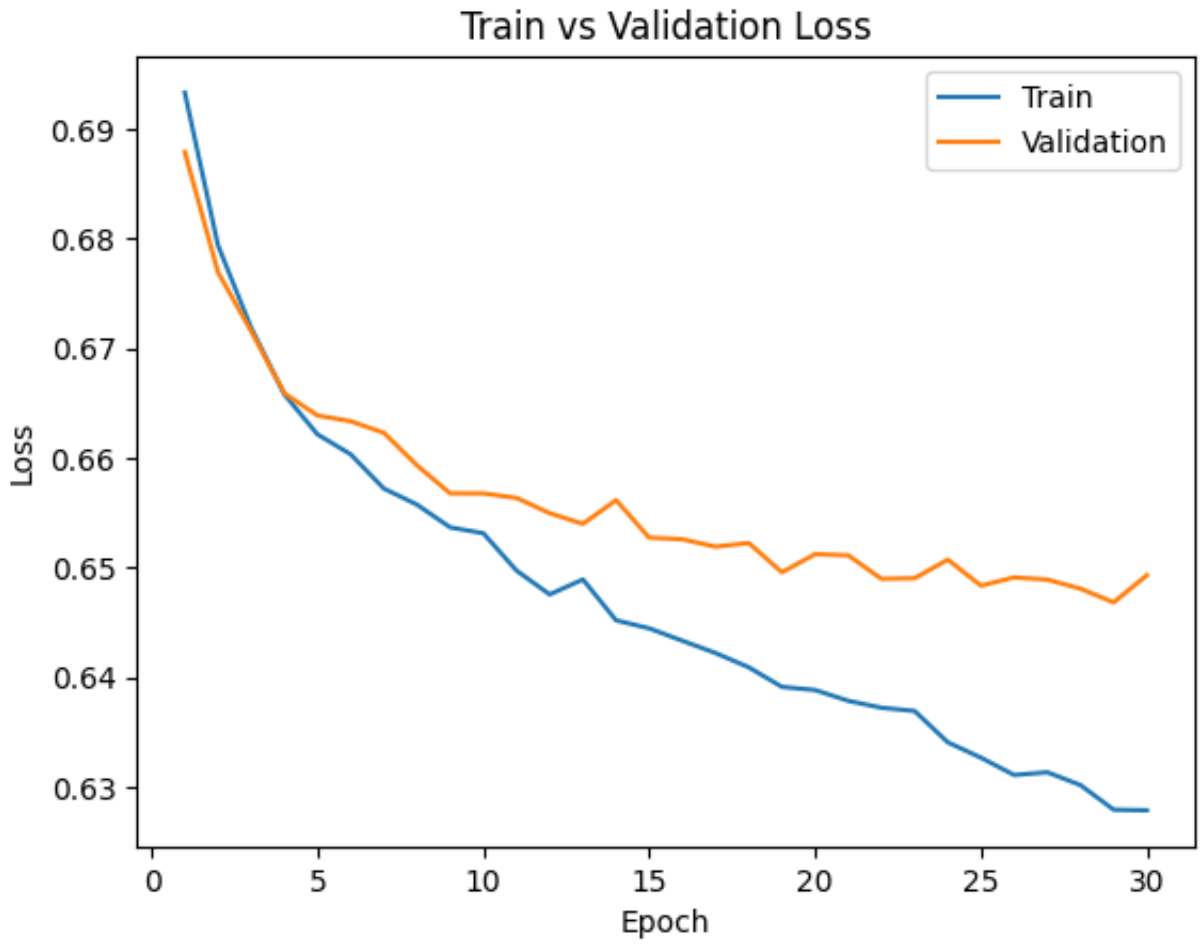
Hint: The ANN in lab 1 was applied on greyscale images. The cat and dog images are colour (RGB) and so you will need to flatted and concatinate all three colour layers before feeding them into an ANN.

In [ ]:
```
class Pigeon(nn.Module):
    def __init__(self):
        super(Pigeon, self).__init__()
        self.layer1 = nn.Linear(32 * 32 * 3, 30)
        self.layer2 = nn.Linear(30, 1)
        self.name = 'ANN'
    def forward(self, img):
        flattened = img.view(-1, 32 * 32 * 3)
        activation1 = self.layer1(flattened)
        activation1 = F.relu(activation1)
        activation2 = self.layer2(activation1)
        activation2 = activation2.squeeze(1)
        return activation2

pigeon = Pigeon()

#Training
train_net(pigeon, batch_size = 264, learning_rate = 0.001)
pigeon_path = get_model_name("ANN", batch_size = 264, learning_rate= 0.001,
plot_training_curve(pigeon_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.486625, Train loss: 0.6933326029008434 |Validation err
: 0.455, Validation loss: 0.6879132390022278
Epoch 2: Train err: 0.427125, Train loss: 0.67939692735672 |Validation err:
0.412, Validation loss: 0.6769351065158844
Epoch 3: Train err: 0.41125, Train loss: 0.6718621023239628 |Validation err:
0.407, Validation loss: 0.6715616956353188
Epoch 4: Train err: 0.40275, Train loss: 0.665788181366459 |Validation err:
0.409, Validation loss: 0.6658869609236717
Epoch 5: Train err: 0.397625, Train loss: 0.6621389600538439 |Validation err
: 0.407, Validation loss: 0.6638624146580696
Epoch 6: Train err: 0.39575, Train loss: 0.6603204531054343 |Validation err:
0.4095, Validation loss: 0.663334883749485
Epoch 7: Train err: 0.392625, Train loss: 0.6571945086602242 |Validation err
: 0.4045, Validation loss: 0.6622826680541039
Epoch 8: Train err: 0.38975, Train loss: 0.6557257809946614 |Validation err:
0.403, Validation loss: 0.6593106985092163
Epoch 9: Train err: 0.38775, Train loss: 0.6536499377219908 |Validation err:
0.4025, Validation loss: 0.6567518338561058
Epoch 10: Train err: 0.384875, Train loss: 0.6531144842024772 |Validation er
r: 0.4055, Validation loss: 0.6567492932081223
Epoch 11: Train err: 0.38225, Train loss: 0.6497075615390655 |Validation err
: 0.401, Validation loss: 0.65633824467659
Epoch 12: Train err: 0.381375, Train loss: 0.647529746255567 |Validation err
: 0.398, Validation loss: 0.6549452543258667
Epoch 13: Train err: 0.377625, Train loss: 0.6489002916120714 |Validation er
r: 0.395, Validation loss: 0.6539705023169518
Epoch 14: Train err: 0.377875, Train loss: 0.6451720922223984 |Validation er
r: 0.3965, Validation loss: 0.6561417356133461
Epoch 15: Train err: 0.375375, Train loss: 0.6444370554339501 |Validation er
r: 0.394, Validation loss: 0.6527101024985313
```

```
Epoch 16: Train err: 0.375875, Train loss: 0.6433041980189662 |Validation er
r: 0.391, Validation loss: 0.6525653228163719
Epoch 17: Train err: 0.3745, Train loss: 0.6421823386223086 |Validation err:
0.3905, Validation loss: 0.6518913432955742
Epoch 18: Train err: 0.371375, Train loss: 0.6408840014088538 |Validation er
r: 0.39, Validation loss: 0.6522262990474701
Epoch 19: Train err: 0.372875, Train loss: 0.6390952910146406 |Validation er
r: 0.3905, Validation loss: 0.649550274014473
Epoch 20: Train err: 0.371625, Train loss: 0.638811876696925 |Validation err
: 0.3845, Validation loss: 0.6512059643864632
Epoch 21: Train err: 0.371, Train loss: 0.6378232529086452 |Validation err:
0.3825, Validation loss: 0.6510981246829033
Epoch 22: Train err: 0.367625, Train loss: 0.63719029003574 |Validation err:
0.389, Validation loss: 0.6489558294415474
Epoch 23: Train err: 0.36675, Train loss: 0.6368980119305272 |Validation err
: 0.3835, Validation loss: 0.6490115523338318
Epoch 24: Train err: 0.36475, Train loss: 0.6340390193846918 |Validation err
: 0.382, Validation loss: 0.6507077440619469
Epoch 25: Train err: 0.3615, Train loss: 0.6326327285458965 |Validation err:
0.384, Validation loss: 0.6483332812786102
Epoch 26: Train err: 0.36325, Train loss: 0.6310545417570299 |Validation err
: 0.379, Validation loss: 0.6490855365991592
Epoch 27: Train err: 0.359, Train loss: 0.6313030085256023 |Validation err:
0.3825, Validation loss: 0.6488847956061363
Epoch 28: Train err: 0.358375, Train loss: 0.6301331750808223 |Validation er
r: 0.383, Validation loss: 0.6480538547039032
Epoch 29: Train err: 0.3575, Train loss: 0.6278889813730794 |Validation err:
0.382, Validation loss: 0.646805614233017
Epoch 30: Train err: 0.355125, Train loss: 0.627836617731279 |Validation err
: 0.382, Validation loss: 0.6492830440402031
Finished Training
Total time elapsed: 119.40 seconds
```


Train vs Validation Error
Train
Validation
Error
0.48
0.46
0.44
0.42
0.40
0.38
0.36
0
5
10
15
20
25
30
Epoch



In [ ]:
```
train_loader, val_loader, test_loader, classes = get_data_loader(
    target_classes=["cat", "dog"],
    batch_size=128)
net = pigeon
criterion = nn.BCEWithLogitsLoss()
error, loss = evaluate(net, test_loader, criterion)
print('The test classification error is:', error)
print('The test classfication loss is:',loss)

#Compared to the CNN network, ANN has a much lower
#accuracy in classfiying cats and dogs.
#It has a test classification error of 0.37, and a
#test loss of 0.64. Therefore, using an ANN model,
#only flattening the image will lose many important
#distinguishing features for cats and dogs, whereas
# a CNN network will capture both low level and high
#level features in cats and dogs for more accurate
#classification.
```

```
Files already downloaded and verified
Files already downloaded and verified
The test classification error is: 0.37
The test classfication loss is: 0.6419178508222103
```