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:

In [2]:
```
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 [4]:
```
################################################################################
# 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, validatio
    and testing datasets. Returns data loaders for the three preprocessed datas

    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 s
        val_loader: iterable validation dataset organized according to batch s
        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], re
    train_sampler = SubsetRandomSampler(relevant_train_indices)
    train_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size
                                               num_workers=1, sampler=train_sa
    val_sampler = SubsetRandomSampler(relevant_val_indices)
    val_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                             num_workers=1, sampler=val_sample
    # 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_clas
    test_sampler = SubsetRandomSampler(relevant_test_indices)
    test_loader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                              num_workers=1, sampler=test_sample
    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 valu

    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 se
         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

        # labels = labels.unsqueeze(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 [3]:
```
# 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
```

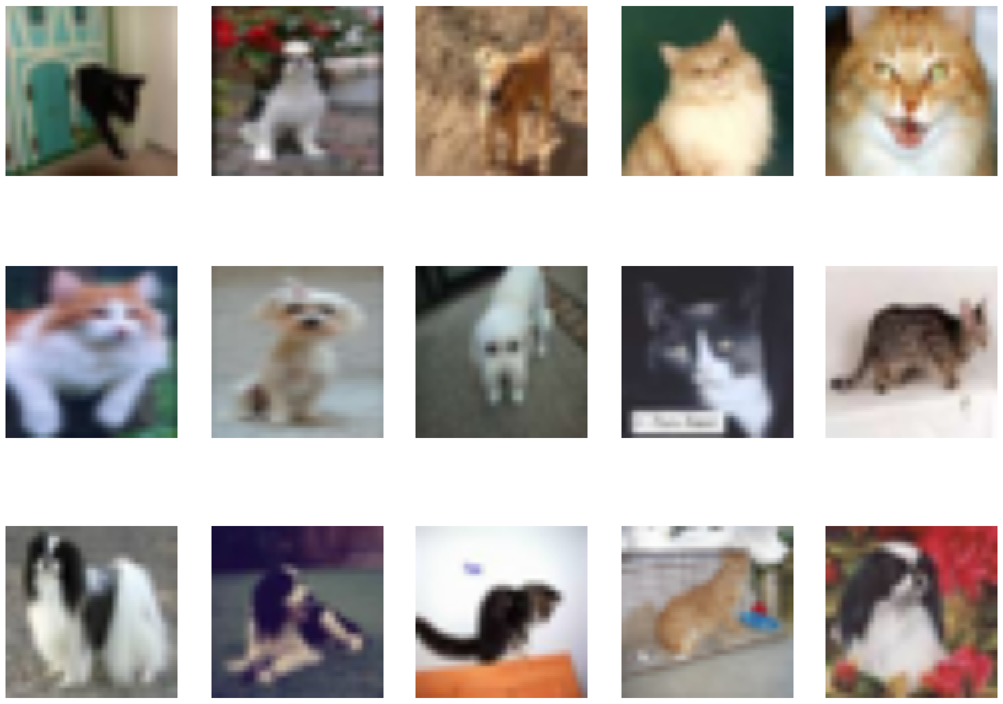
```
0.9%
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/
cifar-10-python.tar.gz
100.0%
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

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 [4]:
```
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 [6]:
```
num_train = len(train_loader.sampler)
num_val = len(val_loader.sampler)
num_test = len(test_loader.sampler)

print(f"Number of training examples: {num_train}")
print(f"Number of validation examples: {num_val}")
print(f"Number of test example: {num_test}")
```

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

Part (c) -- 3pt

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

We need a valiation set when training our model bacause, firstly, it provides an unbiased evaluation of the model fit during training. It helps in assessing the model's performing on the data it hasn't seen before, which gives a better indication of how well the model will perform in reality. Secondly, it helps implementing early stopping to prevent overfitting. Lastly, it can be used for tuning hyperparameters such as learning rate or batch size.

If we judge the performance of our models using the training set loss/error instead, the model may overfit the training data, meaning it will perform well on the training data but poorly on unseen one. Without validation, it's challenging to know if the model generalizes well to new, unseen data. Lastly, the training loss/error may be significantly lower than the validation loss/error due to overfitting.

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 [7]:
```
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 [8]:
```
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 [9]:
```
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.

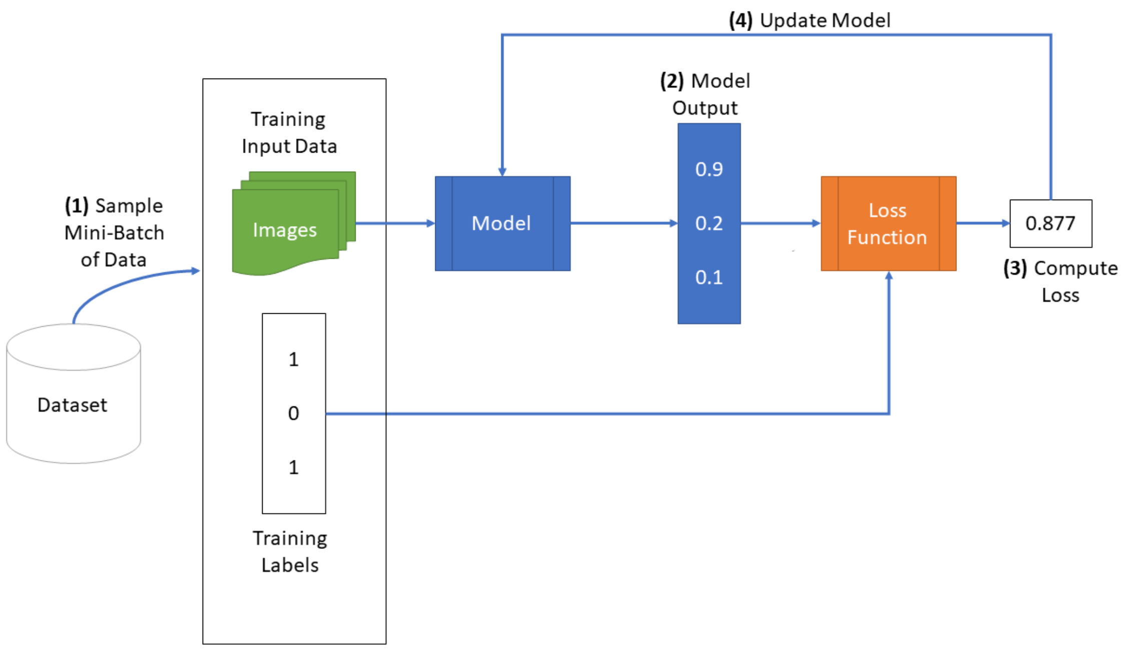
What is the total number of parameters in `small_net` and in `large_net`? (Hint: how many numbers are in each tensor?)

In [17]:
```
# for param in small_net.parameters():
#     print(param.shape)
# for param in large_net.parameters():
#     print(param.shape)

def count_param(model):
    return sum(param.numel() for param in model.parameters() if param.requires_

print(f"Total parameters in SmallNet: {count_param(small_net)}")
print(f"Total parameters in SmallNet: {count_param(large_net)}")

#Number of parameters = sum of layers
#Each layer = input x output + biases
```

```
Total parameters in SmallNet: 386
Total parameters in SmallNet: 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:



In [5]:
```
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
            # labels = labels.unsqueeze(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#.squeeze().long(
            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)
    return net, train_err, train_loss, val_err, 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`?

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.

When we call train_net with small_net, and train for 5 epochs, we got the following files are written to disk:

model_small_bs64_lr0.01_epoch0 model_small_bs64_lr0.01_epoch1 model_small_bs64_lr0.01_epoch2 model_small_bs64_lr0.01_epoch3 model_small_bs64_lr0.01_epoch4 These files are the model checkpoints where each contains the state dictionary of the neural network at the end of the corresponding epoch.

At the en of the training process, the last epoch is used constuct the filenames in CSV: model_small_bs64_lr0.01_epoch4_train_err.csv model_small_bs64_lr0.01_epoch4_train_loss.csv model_small_bs64_lr0.01_epoch4_val_err.csv model_small_bs64_lr0.01_epoch4_val_loss.csv These files contain the training and validation error and loss values recorded.

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 [22]:
```python
# 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')

start_time_small = time.time()
train_net(small_net)
end_time_small = time.time()
elapsed_time_small = end_time_small - start_time_small
print("Total time elapsed for small_net:{:.2f} seconds".format(elapsed_time_sma

start_time_small = time.time()
train_net(large_net)
end_time_small = time.time()
elapsed_time_small = end_time_small - start_time_small
print("Total time elapsed for small_net:{:.2f} seconds".format(elapsed_time_sma

'''large_net requires more time to process than small_net.
The reason lies in the number of parameters in each network.
large_net has more layers and a large number of neurons per
```

```
layer compared to small_net, leading to increased computational
coplexity and thus longer training times.'''
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.416875, Train loss: 0.6750125570297241 |Validation err:
0.3665, Validation loss: 0.651271503418684
Epoch 2: Train err: 0.365375, Train loss: 0.645360360622406 |Validation err:
0.3845, Validation loss: 0.6601899191737175
Epoch 3: Train err: 0.35025, Train loss: 0.6313966364860535 |Validation err:
0.345, Validation loss: 0.6224232353270054
Epoch 4: Train err: 0.33725, Train loss: 0.615510021686554 |Validation err: 0.
355, Validation loss: 0.6221916098147631
Epoch 5: Train err: 0.325, Train loss: 0.6042631051540375 |Validation err: 0.3
235, Validation loss: 0.6165914889425039
Epoch 6: Train err: 0.312875, Train loss: 0.5926906671524048 |Validation err:
0.335, Validation loss: 0.616919482126832
Epoch 7: Train err: 0.309625, Train loss: 0.5865938510894776 |Validation err:
0.333, Validation loss: 0.60477314889431
Epoch 8: Train err: 0.30275, Train loss: 0.5790050482749939 |Validation err:
0.33, Validation loss: 0.6020251903682947
Epoch 9: Train err: 0.300375, Train loss: 0.5771953854560852 |Validation err:
0.326, Validation loss: 0.6010829322040081
Epoch 10: Train err: 0.298625, Train loss: 0.5693206179141999 |Validation err:
0.3175, Validation loss: 0.5904763760045171
Epoch 11: Train err: 0.29225, Train loss: 0.5660302231311798 |Validation err:
0.325, Validation loss: 0.598873233422637
Epoch 12: Train err: 0.288, Train loss: 0.5601779036521911 |Validation err: 0.
332, Validation loss: 0.6010615658015013
Epoch 13: Train err: 0.28275, Train loss: 0.5616342921257019 |Validation err:
0.314, Validation loss: 0.597919387742877
Epoch 14: Train err: 0.286125, Train loss: 0.5547096033096314 |Validation err:
0.33, Validation loss: 0.6100959070026875
Epoch 15: Train err: 0.286125, Train loss: 0.5528013541698455 |Validation err:
0.3135, Validation loss: 0.5972558706998825
Epoch 16: Train err: 0.28775, Train loss: 0.558177636384964 |Validation err:
0.3125, Validation loss: 0.6017906814813614
Epoch 17: Train err: 0.286125, Train loss: 0.5528635742664337 |Validation err:
0.315, Validation loss: 0.5913266986608505
Epoch 18: Train err: 0.282625, Train loss: 0.5498520925045013 |Validation err:
0.314, Validation loss: 0.5923054600134492
Epoch 19: Train err: 0.280625, Train loss: 0.5468230969905853 |Validation err:
0.3125, Validation loss: 0.5979579910635948
Epoch 20: Train err: 0.275125, Train loss: 0.5450280342102051 |Validation err:
0.3115, Validation loss: 0.5971324592828751
Epoch 21: Train err: 0.282375, Train loss: 0.5475271475315094 |Validation err:
0.306, Validation loss: 0.5867987843230367
Epoch 22: Train err: 0.276375, Train loss: 0.5455792791843415 |Validation err:
0.314, Validation loss: 0.5975023871287704
Epoch 23: Train err: 0.2795, Train loss: 0.5461577324867248 |Validation err:
0.322, Validation loss: 0.594504171051085
Epoch 24: Train err: 0.27875, Train loss: 0.5423737134933472 |Validation err:
0.32, Validation loss: 0.5950107229873538
Epoch 25: Train err: 0.270875, Train loss: 0.539586375951767 |Validation err:
0.317, Validation loss: 0.5968796731904149
Epoch 26: Train err: 0.27525, Train loss: 0.5415864021778106 |Validation err:
0.3115, Validation loss: 0.5876989085227251
Epoch 27: Train err: 0.276875, Train loss: 0.5401414029598236 |Validation err:
0.308, Validation loss: 0.60043905954808
Epoch 28: Train err: 0.276875, Train loss: 0.5403922572135925 |Validation err:
0.3035, Validation loss: 0.590910043567419
Epoch 29: Train err: 0.275, Train loss: 0.5406191668510437 |Validation err: 0.
3105, Validation loss: 0.602784238755703
```

```
Epoch 30: Train err: 0.27275, Train loss: 0.5399593875408173 |Validation err:
0.3095, Validation loss: 0.5954833133146167
Finished Training
Total time elapsed: 776.01 seconds
Total time elapsed for small_net:796.33 seconds
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.45825, Train loss: 0.6907159585952759 |Validation err:
0.4285, Validation loss: 0.6825322918593884
Epoch 2: Train err: 0.42025, Train loss: 0.6778951029777527 |Validation err:
0.4135, Validation loss: 0.6724218428134918
Epoch 3: Train err: 0.403375, Train loss: 0.6652929220199585 |Validation err:
0.388, Validation loss: 0.6509127989411354
Epoch 4: Train err: 0.386875, Train loss: 0.6564120931625366 |Validation err:
0.3895, Validation loss: 0.6492501199245453
Epoch 5: Train err: 0.379125, Train loss: 0.6488895163536071 |Validation err:
0.3735, Validation loss: 0.6411709655076265
Epoch 6: Train err: 0.358625, Train loss: 0.6351959795951844 |Validation err:
0.353, Validation loss: 0.6288387458771467
Epoch 7: Train err: 0.34975, Train loss: 0.6244515461921691 |Validation err:
0.347, Validation loss: 0.621803779155016
Epoch 8: Train err: 0.333, Train loss: 0.60806897854805 |Validation err: 0.34
6, Validation loss: 0.6098825614899397
Epoch 9: Train err: 0.3255, Train loss: 0.6013412532806397 |Validation err: 0.
353, Validation loss: 0.6113541182130575
Epoch 10: Train err: 0.316125, Train loss: 0.589971958398819 |Validation err:
0.3275, Validation loss: 0.5958525044843554
Epoch 11: Train err: 0.304625, Train loss: 0.5755284597873688 |Validation err:
0.3225, Validation loss: 0.5974238198250532
Epoch 12: Train err: 0.29225, Train loss: 0.5633340358734131 |Validation err:
0.335, Validation loss: 0.6160336602479219
Epoch 13: Train err: 0.2945, Train loss: 0.5602497692108155 |Validation err:
0.3055, Validation loss: 0.5852993428707123
Epoch 14: Train err: 0.276, Train loss: 0.5410372107028961 |Validation err: 0.
3115, Validation loss: 0.5981923518702388
Epoch 15: Train err: 0.274375, Train loss: 0.5361248102188111 |Validation err:
0.322, Validation loss: 0.589577816426754
Epoch 16: Train err: 0.271875, Train loss: 0.5321849179267883 |Validation err:
0.33, Validation loss: 0.5949276192113757
Epoch 17: Train err: 0.258375, Train loss: 0.5216944117546082 |Validation err:
0.321, Validation loss: 0.6040281560271978
Epoch 18: Train err: 0.24625, Train loss: 0.5060607051849365 |Validation err:
0.2975, Validation loss: 0.576768814586103
Epoch 19: Train err: 0.242125, Train loss: 0.49587542009353636 |Validation er
r: 0.319, Validation loss: 0.5961064686998725
Epoch 20: Train err: 0.24025, Train loss: 0.4912865343093872 |Validation err:
0.3165, Validation loss: 0.6071177646517754
Epoch 21: Train err: 0.23825, Train loss: 0.48105130696296694 |Validation err:
0.3045, Validation loss: 0.5866228230297565
Epoch 22: Train err: 0.22875, Train loss: 0.4668804063796997 |Validation err:
0.3125, Validation loss: 0.6127813709899783
Epoch 23: Train err: 0.221375, Train loss: 0.4632520172595978 |Validation err:
0.319, Validation loss: 0.6030319491401315
Epoch 24: Train err: 0.2145, Train loss: 0.4519543213844299 |Validation err:
0.3095, Validation loss: 0.6148185972124338
Epoch 25: Train err: 0.208125, Train loss: 0.4380663480758667 |Validation err:
0.315, Validation loss: 0.6092131873592734
Epoch 26: Train err: 0.203875, Train loss: 0.431913818359375 |Validation err:
0.3015, Validation loss: 0.6327717043459415
Epoch 27: Train err: 0.191125, Train loss: 0.4144328112602234 |Validation err:
```

```
0.3035, Validation loss: 0.6441452819854021
Epoch 28: Train err: 0.187375, Train loss: 0.40448825764656066 |Validation er
r: 0.3165, Validation loss: 0.6951028285548091
Epoch 29: Train err: 0.178, Train loss: 0.3864849299192429 |Validation err: 0.
3135, Validation loss: 0.7589928675442934
Epoch 30: Train err: 0.170375, Train loss: 0.37377100086212156 |Validation er
r: 0.3105, Validation loss: 0.6959757674485445
Finished Training
Total time elapsed: 889.36 seconds
Total time elapsed for small_net:909.40 seconds
```

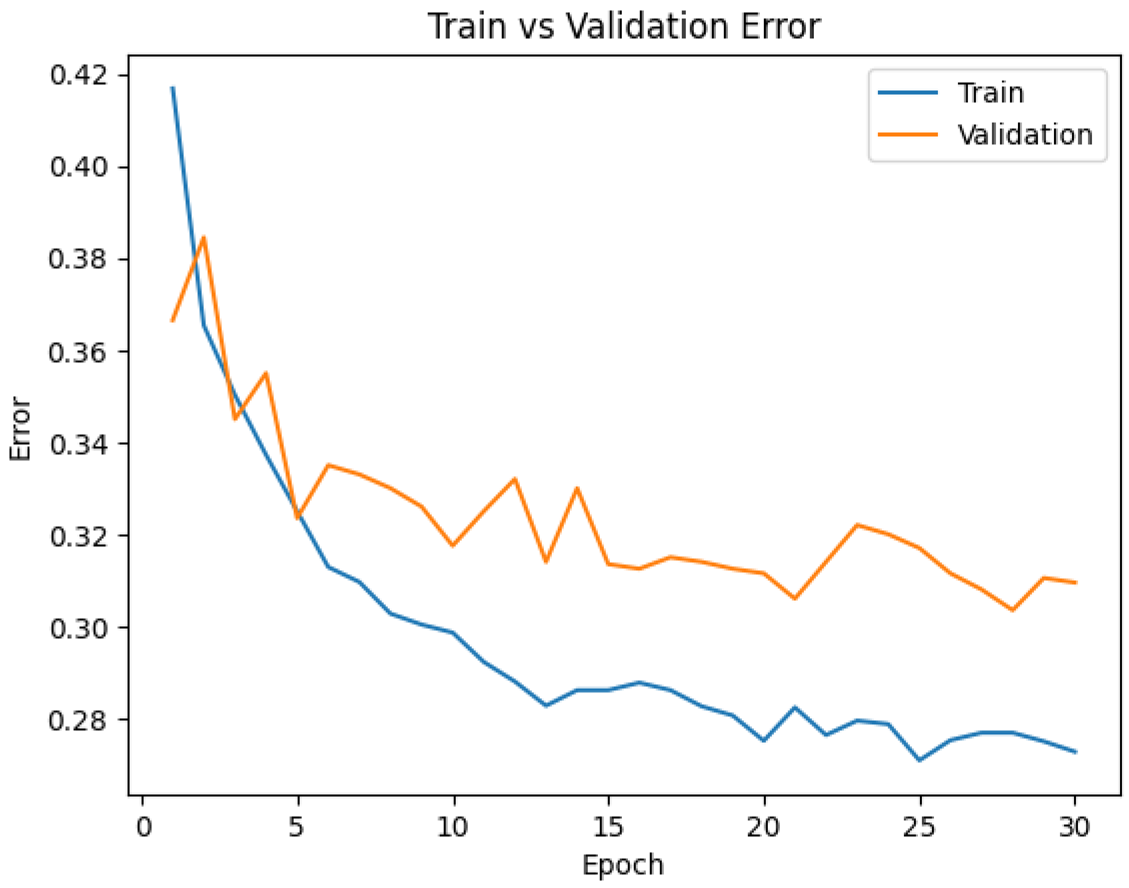
Out[22]:
```
'large_net requires more time to process than small_net. \nThe reason lies in
the number of parameters in each network.\nlarge_net has more layers and a lar
ge number of neurons per\nlayer compared to small_net, leading to increased co
mputational\ncoplexity and thus longer training times.'
```

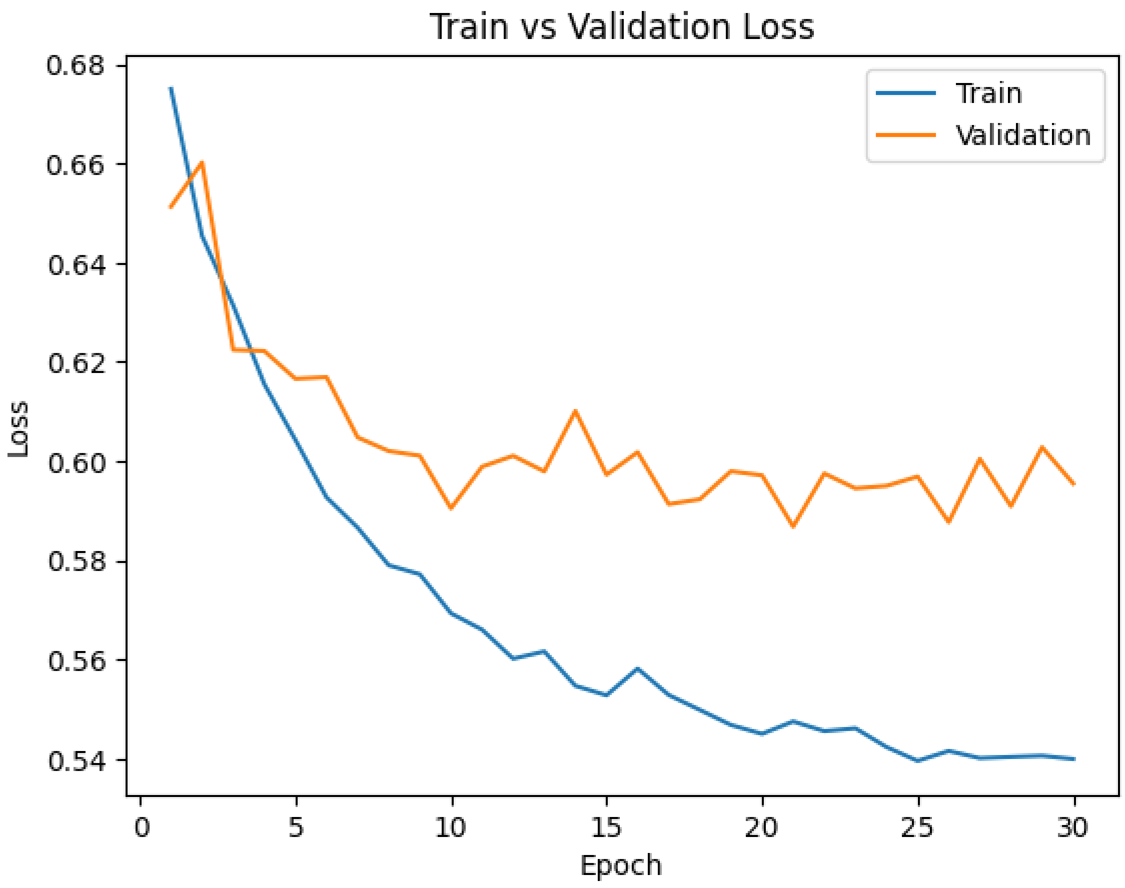
Part (e) - 2pt

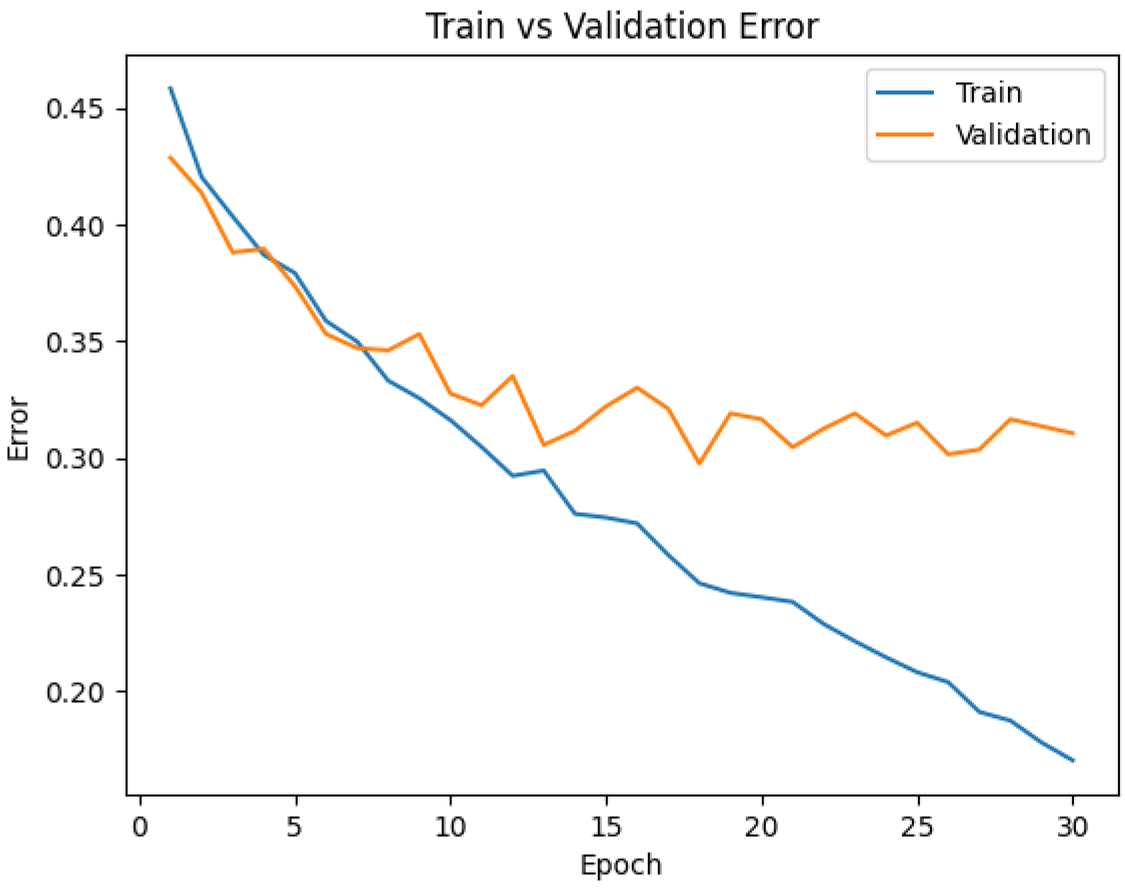
Use the function `plot_training_curve` to display the trajectory of the training/validation error and the training/validation loss. You will need to use the function `get_model_name` to generate the argument to the `plot_training_curve` function.

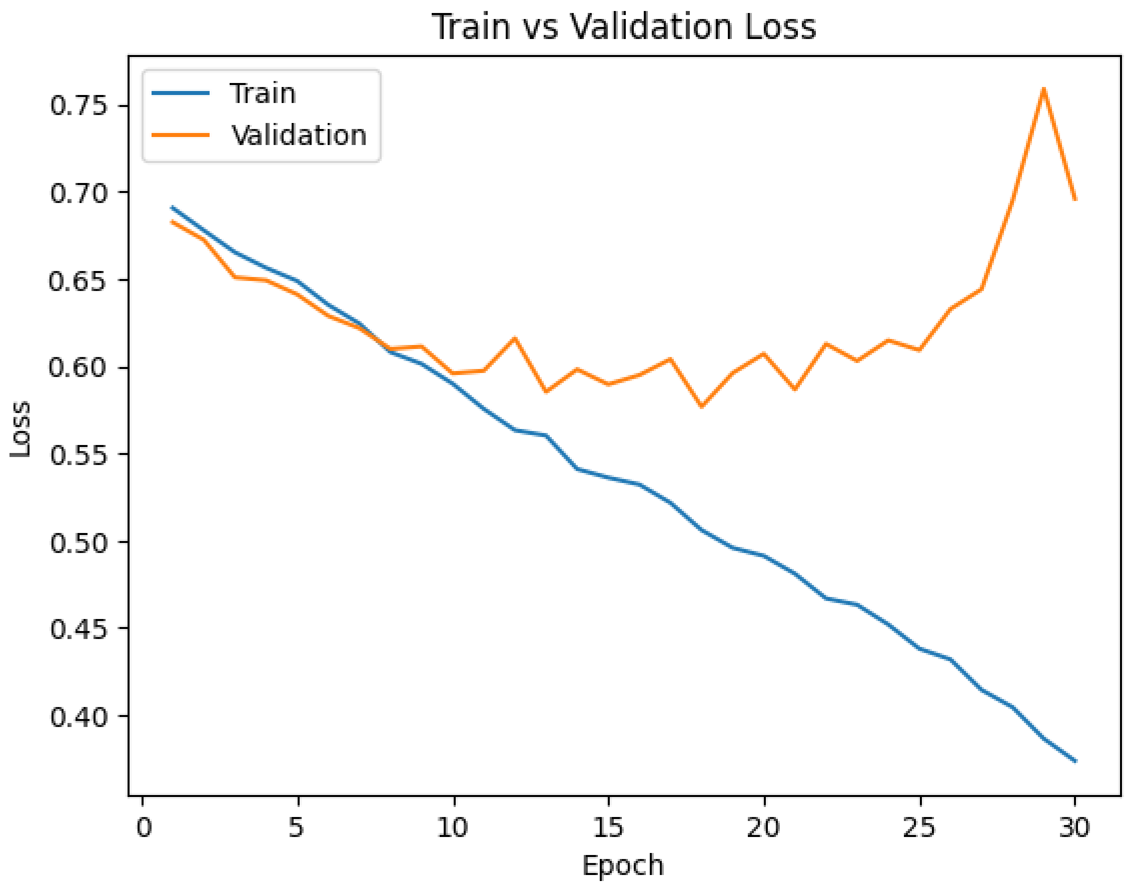
Do this for both the small network and the large network. Include both plots in your writeup.

In [24]:
```
small_net_model_name = get_model_name(small_net.name, batch_size=64, learning_
large_net_model_name = get_model_name(large_net.name, batch_size=64, learning_

plot_training_curve(small_net_model_name)
plot_training_curve(large_net_model_name)
```




Train vs Validation Loss
Train
Validation
Loss
Epoch
0.68
0.66
0.64
0.62
0.60
0.58
0.56
0.54
0
5
10
15
20
25
30


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



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.

Training curves for small_net:

- Training E/L: decreases over time but not reaches low values
- Validation E/L: slightly decreases over time but remains relatively high

Training curves for large_net:

- Training E/L: Decreases more linearly and rapidly to low values
- Validation E/L: initially decreases then either start increasing or plateauing.

--> small_net may be underfitting since both training and validation e/l are high and do not decrease significantly, suggesting the capacity is insufficient for the complexity of the data.

--> large_net may be overfitting since the training e/l is low but the validation e/l is high or increasing after an initial decrease, due to the fact that the model is memorizing the training data rather than generalizing.

Part 3. Optimization Parameters [12 pt]

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

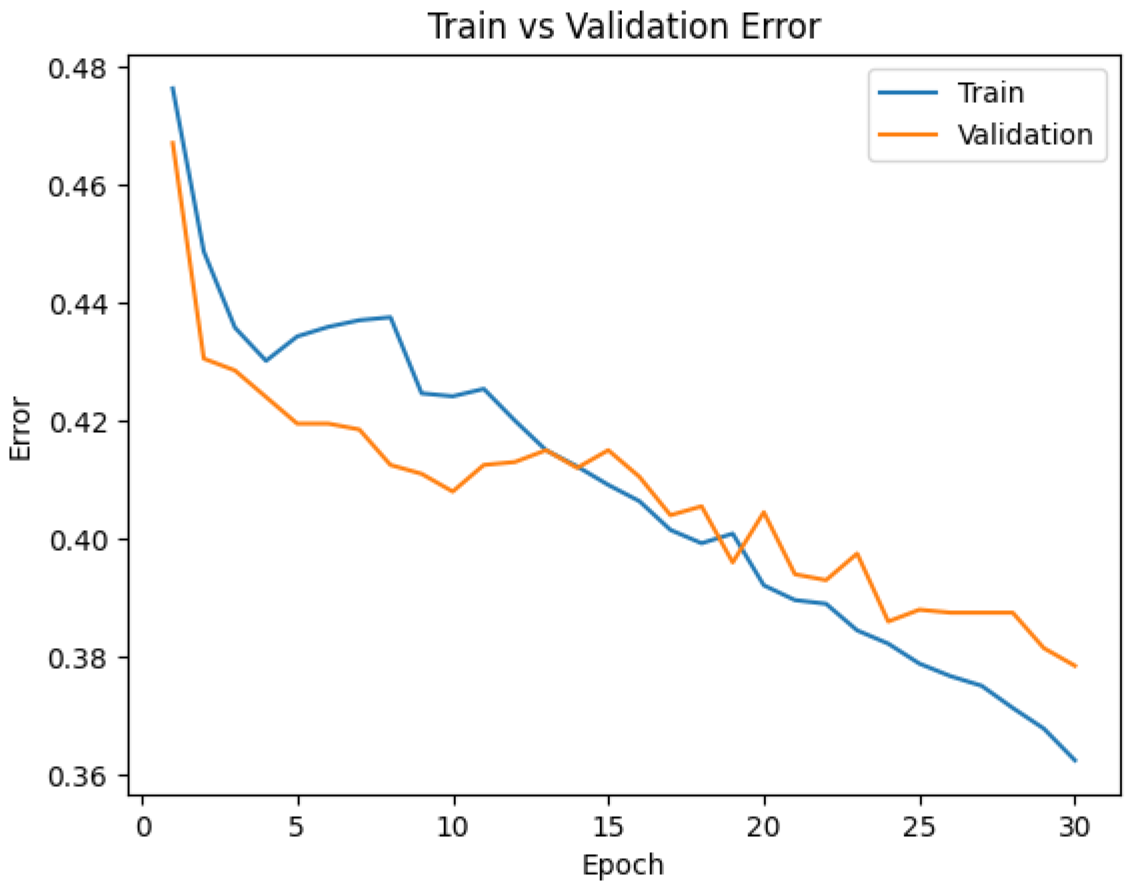
Part (a) - 3pt

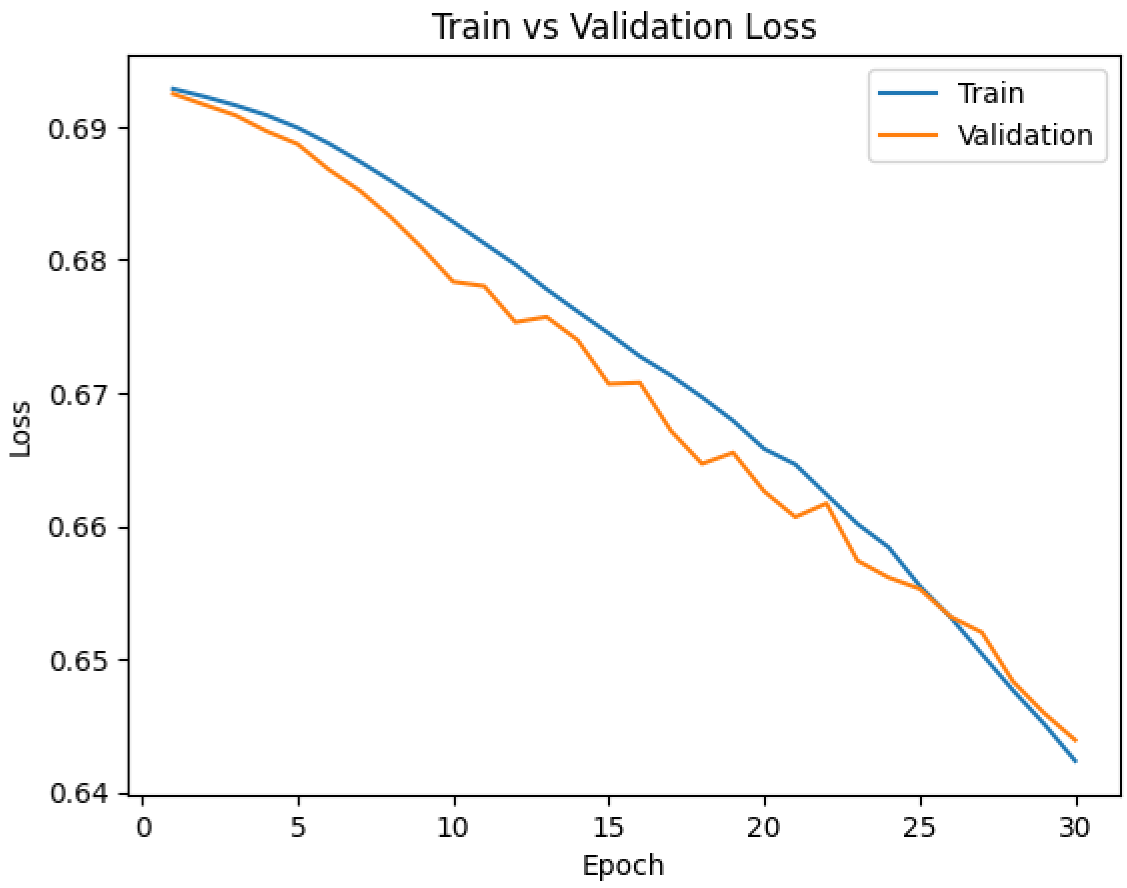
Train `large_net` with all default parameters, except set `learning_rate=0.001`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *lowering* the learning rate.

In [25]:
```
# 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_path = get_model_name(large_net.name, 64, 0.001, 29) #assuming 30 epochs
plot_training_curve(model_path)

'''Lowering the learning rate generally results in a longer training
porcess with a more gradual and stable convergence. This can help
achieving better generalization by avoiding overshooting and ensuring
the model does not skip over minima in the loss function. However,
an excessively low learning rate might lead to excessively slow training
and the potential for the model to get stuck in suboptimal solutions.

Time elasped: 1011.79s'''
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.47625, Train loss: 0.6928360052108765 |Validation err:
0.467, Validation loss: 0.6924686636775732
Epoch 2: Train err: 0.448625, Train loss: 0.6922589735984802 |Validation err:
0.4305, Validation loss: 0.6916493605822325
Epoch 3: Train err: 0.43575, Train loss: 0.6916067447662354 |Validation err:
0.4285, Validation loss: 0.6908544562757015
Epoch 4: Train err: 0.430125, Train loss: 0.6908613076210022 |Validation err:
0.424, Validation loss: 0.6896594148129225
Epoch 5: Train err: 0.43425, Train loss: 0.6899195213317871 |Validation err:
0.4195, Validation loss: 0.6886937506496906
Epoch 6: Train err: 0.435875, Train loss: 0.6887414779663086 |Validation err:
0.4195, Validation loss: 0.6867830120027065
Epoch 7: Train err: 0.437, Train loss: 0.6873778896331787 |Validation err: 0.4
185, Validation loss: 0.685198362916708
Epoch 8: Train err: 0.4375, Train loss: 0.6859283361434937 |Validation err: 0.
4125, Validation loss: 0.6831994950771332
Epoch 9: Train err: 0.424625, Train loss: 0.6844063110351563 |Validation err:
0.411, Validation loss: 0.6808883715420961
Epoch 10: Train err: 0.424125, Train loss: 0.6828512725830078 |Validation err:
0.408, Validation loss: 0.6783524416387081
Epoch 11: Train err: 0.425375, Train loss: 0.6812374067306518 |Validation err:
0.4125, Validation loss: 0.6780228316783905
Epoch 12: Train err: 0.42, Train loss: 0.6796347332000733 |Validation err: 0.4
13, Validation loss: 0.6753195822238922
Epoch 13: Train err: 0.415, Train loss: 0.6777958979606629 |Validation err: 0.
415, Validation loss: 0.6757139153778553
Epoch 14: Train err: 0.41225, Train loss: 0.676115725517273 |Validation err:
0.412, Validation loss: 0.6739730350673199
Epoch 15: Train err: 0.409125, Train loss: 0.6744775424003601 |Validation err:
0.415, Validation loss: 0.6706844922155142
Epoch 16: Train err: 0.406375, Train loss: 0.6727494630813599 |Validation err:
0.4105, Validation loss: 0.6707756388932467
Epoch 17: Train err: 0.4015, Train loss: 0.6713142442703247 |Validation err:
0.404, Validation loss: 0.6671578288078308
Epoch 18: Train err: 0.39925, Train loss: 0.6696815996170044 |Validation err:
0.4055, Validation loss: 0.6646826025098562
Epoch 19: Train err: 0.400875, Train loss: 0.6679153003692627 |Validation err:
0.396, Validation loss: 0.6655164361000061
Epoch 20: Train err: 0.392125, Train loss: 0.6657992796897888 |Validation err:
0.4045, Validation loss: 0.6626073978841305
Epoch 21: Train err: 0.389625, Train loss: 0.6646366119384766 |Validation err:
0.394, Validation loss: 0.6606824025511742
Epoch 22: Train err: 0.389, Train loss: 0.6623814749717712 |Validation err: 0.
393, Validation loss: 0.6617059614509344
Epoch 23: Train err: 0.3845, Train loss: 0.6601637983322144 |Validation err:
0.3975, Validation loss: 0.6574058458209038
Epoch 24: Train err: 0.38225, Train loss: 0.6584126424789428 |Validation err:
0.386, Validation loss: 0.6561386473476887
Epoch 25: Train err: 0.378875, Train loss: 0.6555113334655762 |Validation err:
0.388, Validation loss: 0.6552941966801882
Epoch 26: Train err: 0.37675, Train loss: 0.6531408720016479 |Validation err:
0.3875, Validation loss: 0.65318663418293
Epoch 27: Train err: 0.375125, Train loss: 0.6503939228057861 |Validation err:
0.3875, Validation loss: 0.6520215608179569
Epoch 28: Train err: 0.371375, Train loss: 0.6476678924560547 |Validation err:
0.3875, Validation loss: 0.6483367551118135
Epoch 29: Train err: 0.367875, Train loss: 0.6451551876068116 |Validation err:
0.3815, Validation loss: 0.6459614392369986
```

```
Epoch 30: Train err: 0.3625, Train loss: 0.6423756089210511 |Validation err:
0.3785, Validation loss: 0.6439380161464214
Finished Training
Total time elapsed: 1011.79 seconds
```





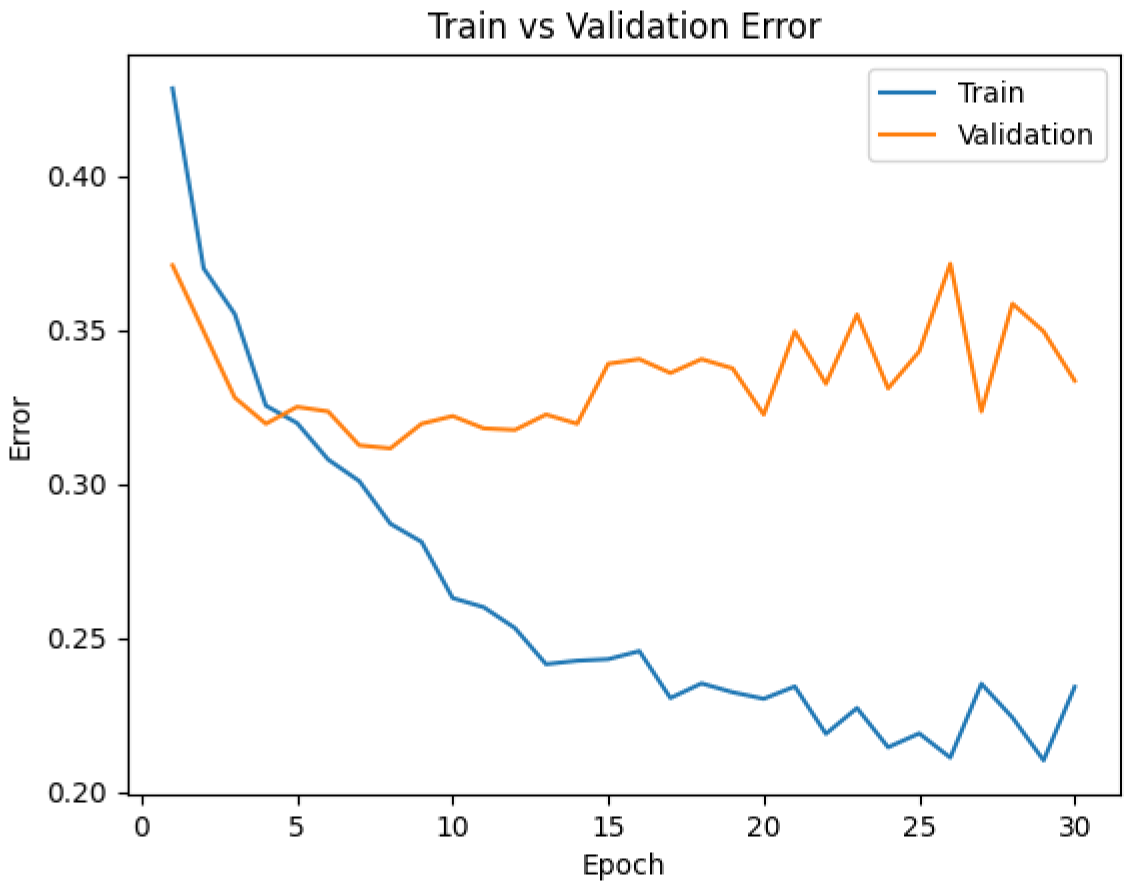
Part (b) - 3pt

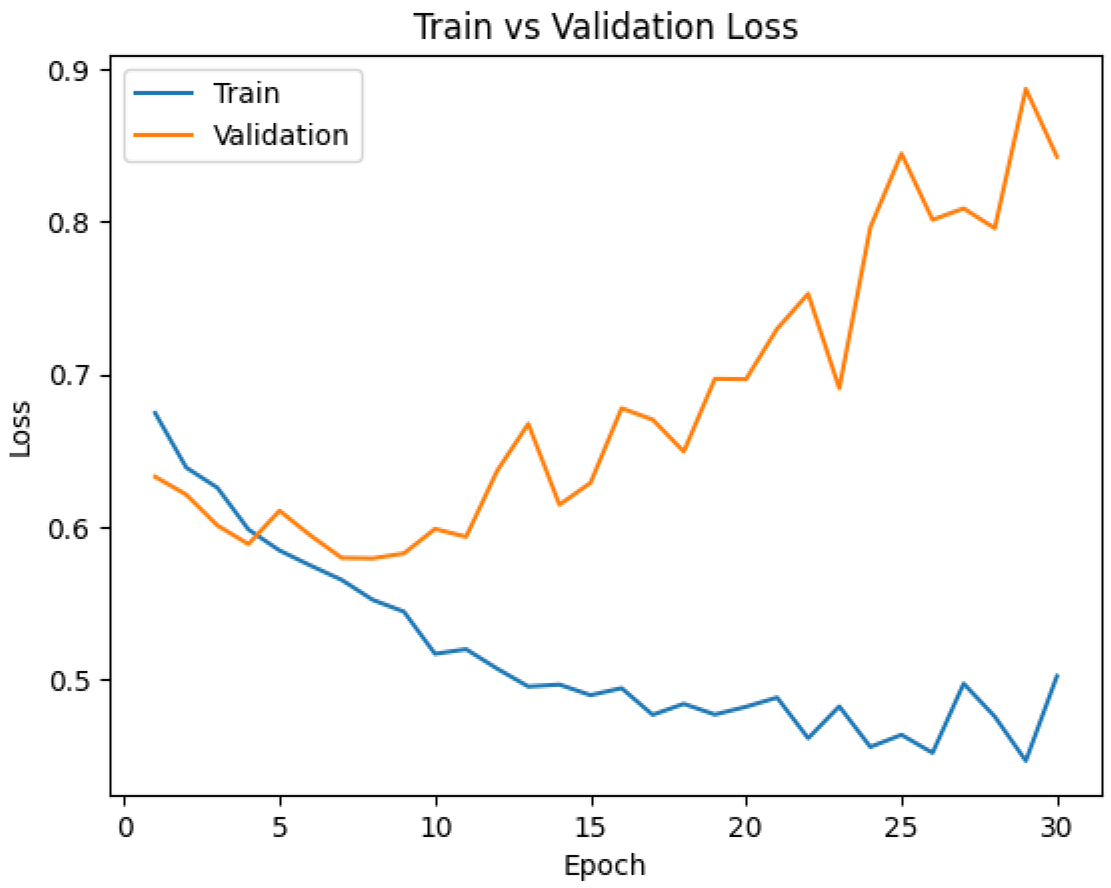
Train `large_net` with all default parameters, except set `learning_rate=0.1`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *increasing* the learning rate.

In [26]:
```
large_net = LargeNet()
train_net(large_net, learning_rate=0.1)

model_path = get_model_name(large_net.name, 64, 0.1, 29) #assuming 30 epochs
plot_training_curve(model_path)

'''Increasing the learning rate to 0.1 generally results in a shorter
training process due to faster intial convergence. However, this comes
wth the potential downside of instability in the training process. The
training and validation errors and losses may not decrease smoothly and
might oscillate or increase if the learning rate is excessively high,
preventing the model from converging to an optimal solution.

Time elapsed: 859.60s'''
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.4285, Train loss: 0.6748176345825195 |Validation err: 0.
371, Validation loss: 0.6331098210066557
Epoch 2: Train err: 0.37, Train loss: 0.6390678839683532 |Validation err: 0.34
95, Validation loss: 0.6213860679417849
Epoch 3: Train err: 0.355125, Train loss: 0.6258888664245605 |Validation err:
0.328, Validation loss: 0.6011855136603117
Epoch 4: Train err: 0.325375, Train loss: 0.5985033819675446 |Validation err:
0.3195, Validation loss: 0.5889167711138725
Epoch 5: Train err: 0.31975, Train loss: 0.5847481439113617 |Validation err:
0.325, Validation loss: 0.6108177285641432
Epoch 6: Train err: 0.307875, Train loss: 0.5748495907783508 |Validation err:
0.3235, Validation loss: 0.594539794139564
Epoch 7: Train err: 0.300875, Train loss: 0.5654441576004028 |Validation err:
0.3125, Validation loss: 0.5799257354810834
Epoch 8: Train err: 0.287, Train loss: 0.5523363053798676 |Validation err: 0.3
115, Validation loss: 0.5795844318345189
Epoch 9: Train err: 0.281125, Train loss: 0.5446515836715698 |Validation err:
0.3195, Validation loss: 0.5828199805691838
Epoch 10: Train err: 0.262875, Train loss: 0.5169935195446015 |Validation err:
0.322, Validation loss: 0.5988915394991636
Epoch 11: Train err: 0.259875, Train loss: 0.5200313606262207 |Validation err:
0.318, Validation loss: 0.5937206912785769
Epoch 12: Train err: 0.253125, Train loss: 0.5072792971134186 |Validation err:
0.3175, Validation loss: 0.6370595134794712
Epoch 13: Train err: 0.241375, Train loss: 0.4955726802349091 |Validation err:
0.3225, Validation loss: 0.66776735894382
Epoch 14: Train err: 0.2425, Train loss: 0.4968647246360779 |Validation err:
0.3195, Validation loss: 0.6147276423871517
Epoch 15: Train err: 0.243, Train loss: 0.4899309017658234 |Validation err: 0.
339, Validation loss: 0.6290505044162273
Epoch 16: Train err: 0.245625, Train loss: 0.49452349185943606 |Validation er
r: 0.3405, Validation loss: 0.6779557932168245
Epoch 17: Train err: 0.230375, Train loss: 0.4770533633232117 |Validation err:
0.336, Validation loss: 0.6705080633983016
Epoch 18: Train err: 0.235125, Train loss: 0.48424429941177366 |Validation er
r: 0.3405, Validation loss: 0.6495688920840621
Epoch 19: Train err: 0.23225, Train loss: 0.477221688747406 |Validation err:
0.3375, Validation loss: 0.6972468625754118
Epoch 20: Train err: 0.230125, Train loss: 0.48232338643074035 |Validation er
r: 0.3225, Validation loss: 0.6969501907005906
Epoch 21: Train err: 0.234125, Train loss: 0.4883787717819214 |Validation err:
0.3495, Validation loss: 0.729984562844038
Epoch 22: Train err: 0.21875, Train loss: 0.4616849253177643 |Validation err:
0.3325, Validation loss: 0.7529665129259229
Epoch 23: Train err: 0.227125, Train loss: 0.482577397108078 |Validation err:
0.355, Validation loss: 0.690998699516058
Epoch 24: Train err: 0.214375, Train loss: 0.4560214042663574 |Validation err:
0.331, Validation loss: 0.7966298069804907
Epoch 25: Train err: 0.218875, Train loss: 0.46395377862453463 |Validation er
r: 0.343, Validation loss: 0.8448417577892542
Epoch 26: Train err: 0.211, Train loss: 0.45213429880142214 |Validation err:
0.3715, Validation loss: 0.8014498688280582
Epoch 27: Train err: 0.235, Train loss: 0.4975194091796875 |Validation err: 0.
3235, Validation loss: 0.8089121002703905
Epoch 28: Train err: 0.224, Train loss: 0.47590521931648255 |Validation err:
0.3585, Validation loss: 0.7959228344261646
Epoch 29: Train err: 0.210125, Train loss: 0.44688198709487914 |Validation er
r: 0.3495, Validation loss: 0.8872387297451496
```

```
Epoch 30: Train err: 0.234, Train loss: 0.5024558358192444 |Validation err: 0.
3335, Validation loss: 0.842799480073154
Finished Training
Total time elapsed: 859.60 seconds
```





Out[26]:
```
'Increasing the learning rate to 0.1 generally results in a shorter\ntraining
process due to faster intial convergence. However, this comes\nwth the potenti
al downside of instability in the training process. The \ntraining and validat
ion errors and losses may not decrease smoothly and \nmight oscillate or incre
ase if the learning rate is excessively high, \npreventing the model from conv
erging to an optimal solution.'
```

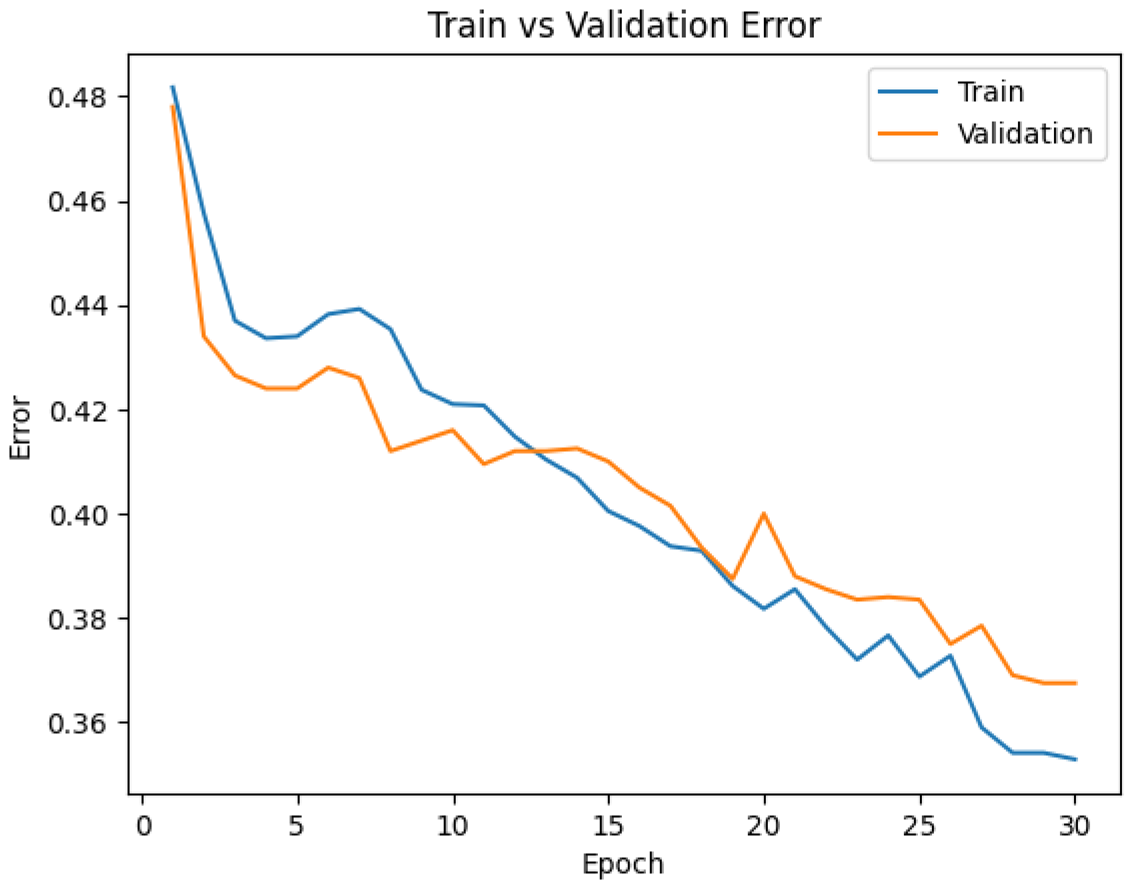
Part (c) - 3pt

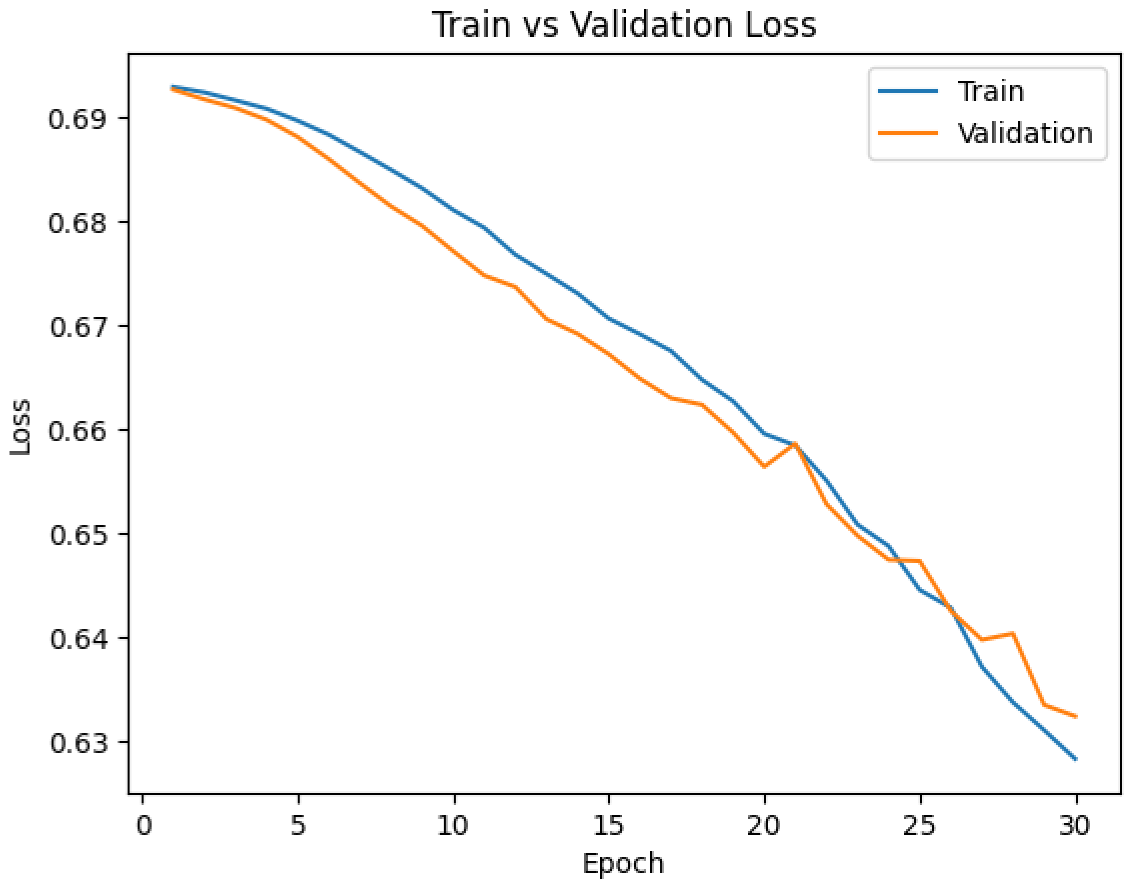
Train `large_net` with all default parameters, including with `learning_rate=0.01`. Now, set `batch_size=512`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *increasing* the batch size.

In [27]:
```
large_net = LargeNet()
train_net(large_net, batch_size=512)

model_path = get_model_name(large_net.name, 512, 0.01, 29) #assuming 30 epochs
plot_training_curve(model_path)

'''Increasing the batch size to 512 generally results in fewer updates
per epoch, which can lead to faster processing per epoch. The training
and validation curves are likely to show smoother and more stable
error and loss reductions due to more accurate estimates. However, this
might slow down the covergence rate per epoch, as larger batches provide
less frequent updates.

Time elapsed: 731.50s'''
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.48175, Train loss: 0.6929379478096962 |Validation err:
0.478, Validation loss: 0.6926824003458023
Epoch 2: Train err: 0.457625, Train loss: 0.6924103945493698 |Validation err:
0.434, Validation loss: 0.6917425245046616
Epoch 3: Train err: 0.437, Train loss: 0.6916500553488731 |Validation err: 0.4
265, Validation loss: 0.6909130066633224
Epoch 4: Train err: 0.433625, Train loss: 0.6908450201153755 |Validation err:
0.424, Validation loss: 0.6897871494293213
Epoch 5: Train err: 0.434, Train loss: 0.6896935999393463 |Validation err: 0.4
24, Validation loss: 0.6881357729434967
Epoch 6: Train err: 0.43825, Train loss: 0.6883535273373127 |Validation err:
0.428, Validation loss: 0.6860138028860092
Epoch 7: Train err: 0.43925, Train loss: 0.6866881288588047 |Validation err:
0.426, Validation loss: 0.6836980283260345
Epoch 8: Train err: 0.435375, Train loss: 0.6849783509969711 |Validation err:
0.412, Validation loss: 0.6814675629138947
Epoch 9: Train err: 0.42375, Train loss: 0.6832022815942764 |Validation err:
0.414, Validation loss: 0.6795944273471832
Epoch 10: Train err: 0.421, Train loss: 0.6811105087399483 |Validation err: 0.
416, Validation loss: 0.6771572679281235
Epoch 11: Train err: 0.42075, Train loss: 0.679404579102993 |Validation err:
0.4095, Validation loss: 0.6748130768537521
Epoch 12: Train err: 0.41475, Train loss: 0.676807101815939 |Validation err:
0.412, Validation loss: 0.673710897564888
Epoch 13: Train err: 0.410375, Train loss: 0.6749714314937592 |Validation err:
0.412, Validation loss: 0.6706132143735886
Epoch 14: Train err: 0.406875, Train loss: 0.6730894558131695 |Validation err:
0.4125, Validation loss: 0.6692064553499222
Epoch 15: Train err: 0.4005, Train loss: 0.6706800721585751 |Validation err:
0.41, Validation loss: 0.6672562062740326
Epoch 16: Train err: 0.397625, Train loss: 0.6691757440567017 |Validation err:
0.405, Validation loss: 0.6649072021245956
Epoch 17: Train err: 0.39375, Train loss: 0.6675703041255474 |Validation err:
0.4015, Validation loss: 0.6630297154188156
Epoch 18: Train err: 0.392875, Train loss: 0.6647942252457142 |Validation err:
0.3935, Validation loss: 0.6623944640159607
Epoch 19: Train err: 0.386125, Train loss: 0.662734717130661 |Validation err:
0.3875, Validation loss: 0.6597277820110321
Epoch 20: Train err: 0.38175, Train loss: 0.6596063487231731 |Validation err:
0.4, Validation loss: 0.656437024474144
Epoch 21: Train err: 0.3855, Train loss: 0.6584837883710861 |Validation err:
0.388, Validation loss: 0.6586578786373138
Epoch 22: Train err: 0.37825, Train loss: 0.6551220901310444 |Validation err:
0.3855, Validation loss: 0.6528529226779938
Epoch 23: Train err: 0.372, Train loss: 0.6508823968470097 |Validation err: 0.
3835, Validation loss: 0.6498084962368011
Epoch 24: Train err: 0.376625, Train loss: 0.6488143876194954 |Validation err:
0.384, Validation loss: 0.6474964022636414
Epoch 25: Train err: 0.36875, Train loss: 0.644597377628088 |Validation err:
0.3835, Validation loss: 0.6473769396543503
Epoch 26: Train err: 0.37275, Train loss: 0.6428937427699566 |Validation err:
0.375, Validation loss: 0.6425864994525909
Epoch 27: Train err: 0.359, Train loss: 0.637214906513691 |Validation err: 0.3
785, Validation loss: 0.6398078054189682
Epoch 28: Train err: 0.354125, Train loss: 0.633796326816082 |Validation err:
0.369, Validation loss: 0.6404024064540863
Epoch 29: Train err: 0.354125, Train loss: 0.6311298832297325 |Validation err:
0.3675, Validation loss: 0.6335538029670715
```

```
Epoch 30: Train err: 0.352875, Train loss: 0.6283673532307148 |Validation err:
0.3675, Validation loss: 0.6324604004621506
Finished Training
Total time elapsed: 731.50 seconds
```





```
Out[27]: 'Increasing the batch size to 512 generally results in fewer updates\nper epoc
         h, which can lead to faster processing per epoch. The training \nand validatio
         n curves are likely to show smoother and more stable\nerror and loss reduction
         s due to more accurate estimates. However, this \nmight slow down the covergen
         ce rate per epoch, as larger batches provide\nless frequent updates.'
```

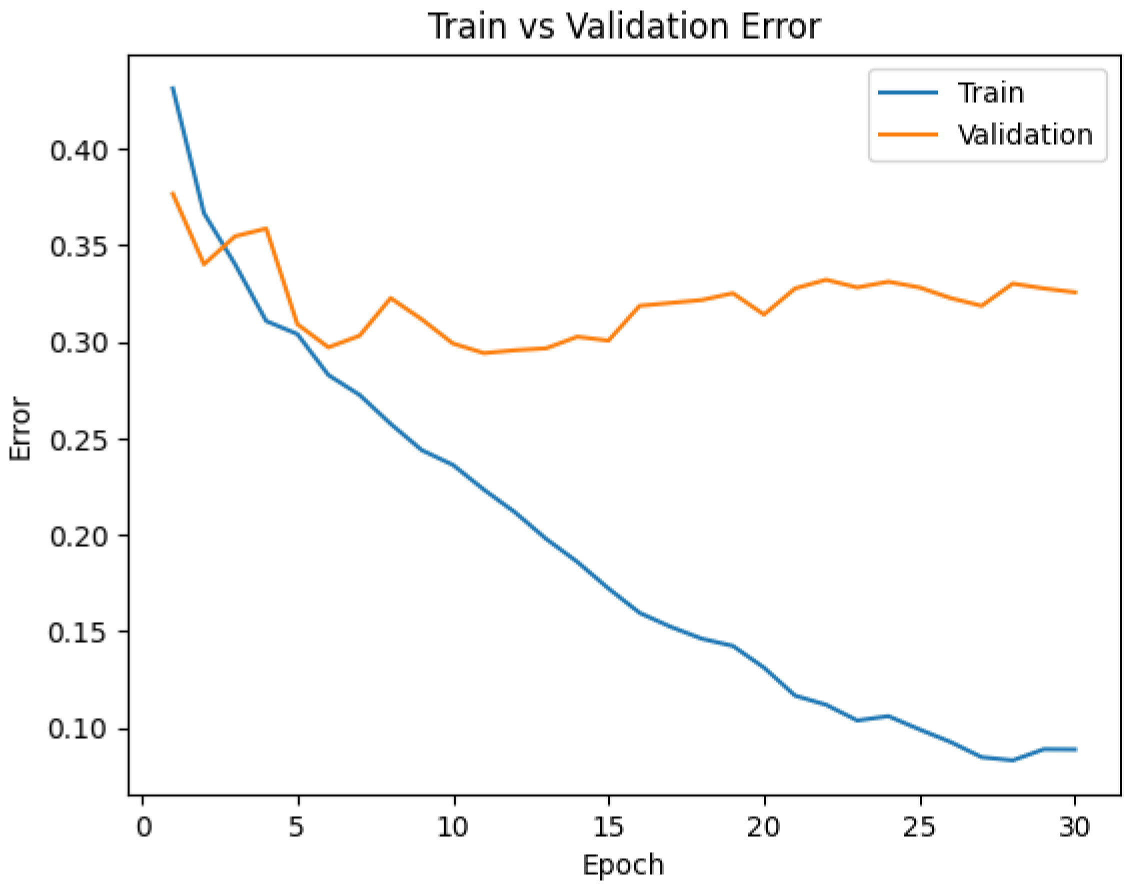
Part (d) - 3pt

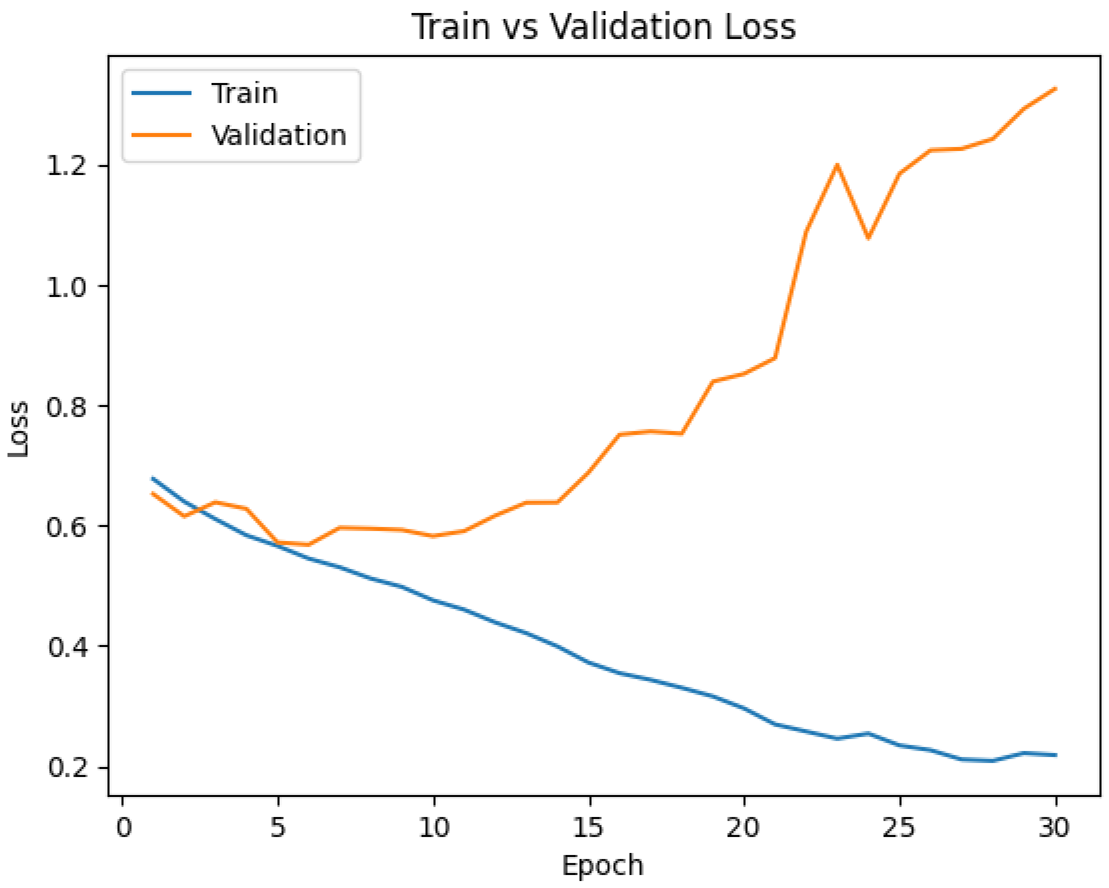
Train `large_net` with all default parameters, including with `learning_rate=0.01`. Now, set `batch_size=16`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *decreasing* the batch size.

```
In [28]: large_net = LargeNet()
         train_net(large_net, batch_size=16)

         model_path = get_model_name(large_net.name, 16, 0.01, 29) #assuming 30 epochs
         plot_training_curve(model_path)

         '''Decreasiing the batch size to 16 generally results in more
         updates per epoch, which can lead to longer training processing per
         epoch. The training and validation curves are likely to show more
         fluctuations and variability due to noisier gradient estimates,
         causing a less stable and more erratic descent of the e/l values.

         Time elapsed: 786.35s'''
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.431125, Train loss: 0.6775951156616211 |Validation err:
0.3765, Validation loss: 0.6524865489006042
Epoch 2: Train err: 0.3665, Train loss: 0.6397962672710419 |Validation err: 0.
34, Validation loss: 0.6153583776950836
Epoch 3: Train err: 0.34, Train loss: 0.6108648151755333 |Validation err: 0.35
45, Validation loss: 0.6386994345188141
Epoch 4: Train err: 0.310625, Train loss: 0.5838440904021263 |Validation err:
0.3585, Validation loss: 0.627867395401001
Epoch 5: Train err: 0.303875, Train loss: 0.5661497976779938 |Validation err:
0.309, Validation loss: 0.5720634469985962
Epoch 6: Train err: 0.282625, Train loss: 0.5452762733697891 |Validation err:
0.297, Validation loss: 0.5681465764045716
Epoch 7: Train err: 0.272375, Train loss: 0.5307247740030289 |Validation err:
0.303, Validation loss: 0.5963114368915557
Epoch 8: Train err: 0.257375, Train loss: 0.5120504722297191 |Validation err:
0.3225, Validation loss: 0.594950798034668
Epoch 9: Train err: 0.24375, Train loss: 0.49815547588467596 |Validation err:
0.3115, Validation loss: 0.5927945764064789
Epoch 10: Train err: 0.236125, Train loss: 0.4758587064445019 |Validation err:
0.299, Validation loss: 0.5824692375659942
Epoch 11: Train err: 0.22325, Train loss: 0.46039793533086776 |Validation err:
0.294, Validation loss: 0.5907773015499115
Epoch 12: Train err: 0.2115, Train loss: 0.4392077845335007 |Validation err:
0.2955, Validation loss: 0.6165005106925965
Epoch 13: Train err: 0.19775, Train loss: 0.4213145221620798 |Validation err:
0.2965, Validation loss: 0.6381007109880448
Epoch 14: Train err: 0.185875, Train loss: 0.39938959342241287 |Validation er
r: 0.3025, Validation loss: 0.6384129821062088
Epoch 15: Train err: 0.172125, Train loss: 0.37252350616455077 |Validation er
r: 0.3005, Validation loss: 0.6880015993118286
Epoch 16: Train err: 0.1595, Train loss: 0.3548425424098969 |Validation err:
0.3185, Validation loss: 0.7510374298095703
Epoch 17: Train err: 0.15225, Train loss: 0.34372536893188954 |Validation err:
0.32, Validation loss: 0.756384923696518
Epoch 18: Train err: 0.146125, Train loss: 0.33057632213830945 |Validation er
r: 0.3215, Validation loss: 0.7525357532501221
Epoch 19: Train err: 0.142375, Train loss: 0.3163160899281502 |Validation err:
0.325, Validation loss: 0.839108394742012
Epoch 20: Train err: 0.131125, Train loss: 0.2964155449643731 |Validation err:
0.314, Validation loss: 0.8519868032932282
Epoch 21: Train err: 0.116625, Train loss: 0.2698152696490288 |Validation err:
0.3275, Validation loss: 0.8781985543966293
Epoch 22: Train err: 0.111875, Train loss: 0.25801056348532436 |Validation er
r: 0.332, Validation loss: 1.0879512681961059
Epoch 23: Train err: 0.10375, Train loss: 0.24617139928415419 |Validation err:
0.328, Validation loss: 1.199315859735012
Epoch 24: Train err: 0.106, Train loss: 0.2545570976734161 |Validation err: 0.
331, Validation loss: 1.0771006989479064
Epoch 25: Train err: 0.099125, Train loss: 0.23489023365452885 |Validation er
r: 0.328, Validation loss: 1.1844299555420876
Epoch 26: Train err: 0.092625, Train loss: 0.2269826663825661 |Validation err:
0.3225, Validation loss: 1.2233365597724914
Epoch 27: Train err: 0.08475, Train loss: 0.21149816323816775 |Validation err:
0.3185, Validation loss: 1.2257391151189805
Epoch 28: Train err: 0.083125, Train loss: 0.20922722659260035 |Validation er
r: 0.33, Validation loss: 1.2421558672189712
Epoch 29: Train err: 0.088875, Train loss: 0.22206926218047737 |Validation er
r: 0.3275, Validation loss: 1.2922988674640656
```

```
Epoch 30: Train err: 0.08875, Train loss: 0.21878464705869555 |Validation err: 
0.3255, Validation loss: 1.3253305872678758
Finished Training
Total time elapsed: 786.35 seconds
```





Out[28]:
```
'Decreasiing the batch size to 16 generally results in more \nupdates per epoc
h, which can lead to longer training processing per \nepoch. The training and
validation curves are likely to show more\nfluctuations and variability due to
noisier gradient estimates,\ncausing a less stable and more erratic descent of
the e/l values. '
```

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 [ ]:
```
'''
Network: 'LargeNet'
--> 'LargeNet' architecture shows better capcacity to model the complex
patterns compared to 'SmallNet', with its deeper layers and greater
number of parameters.

Batchsize: 128
--> 128 is large enough to benefit from the speedups of batch processing
while still providing a reasonably stable estimate of the gradients,
reducing noise compared to smallers batch sizes like 16 or 64, leading
to more stable training and better convergence.

Learning Rate: 0.005
--> This is a compromise choice betweeen the very small learning rates 0.001
```

```
and 0.01. A smaller learning rate can help the model converge more smoothly,
ensuring stady progress towards convergence without the risk of the training
process becoming too slow. '''
```

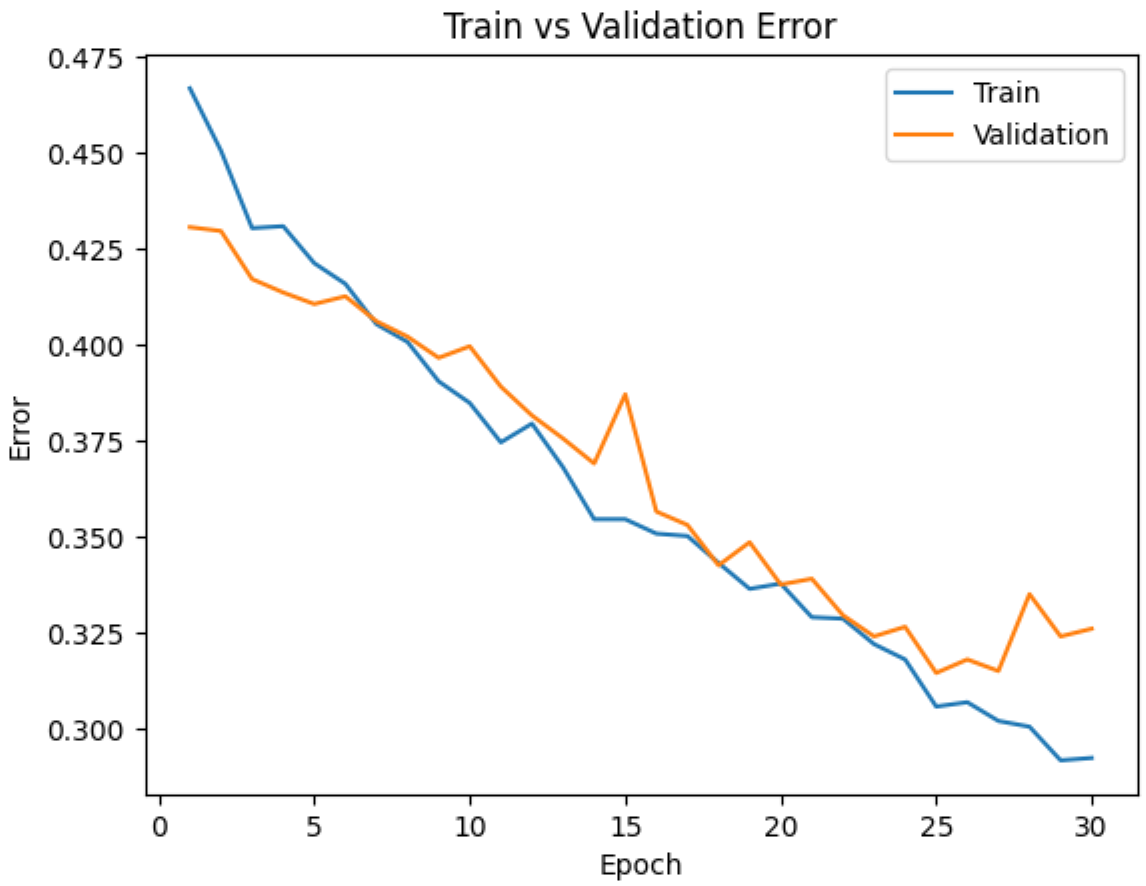
Part (b) - 1pt

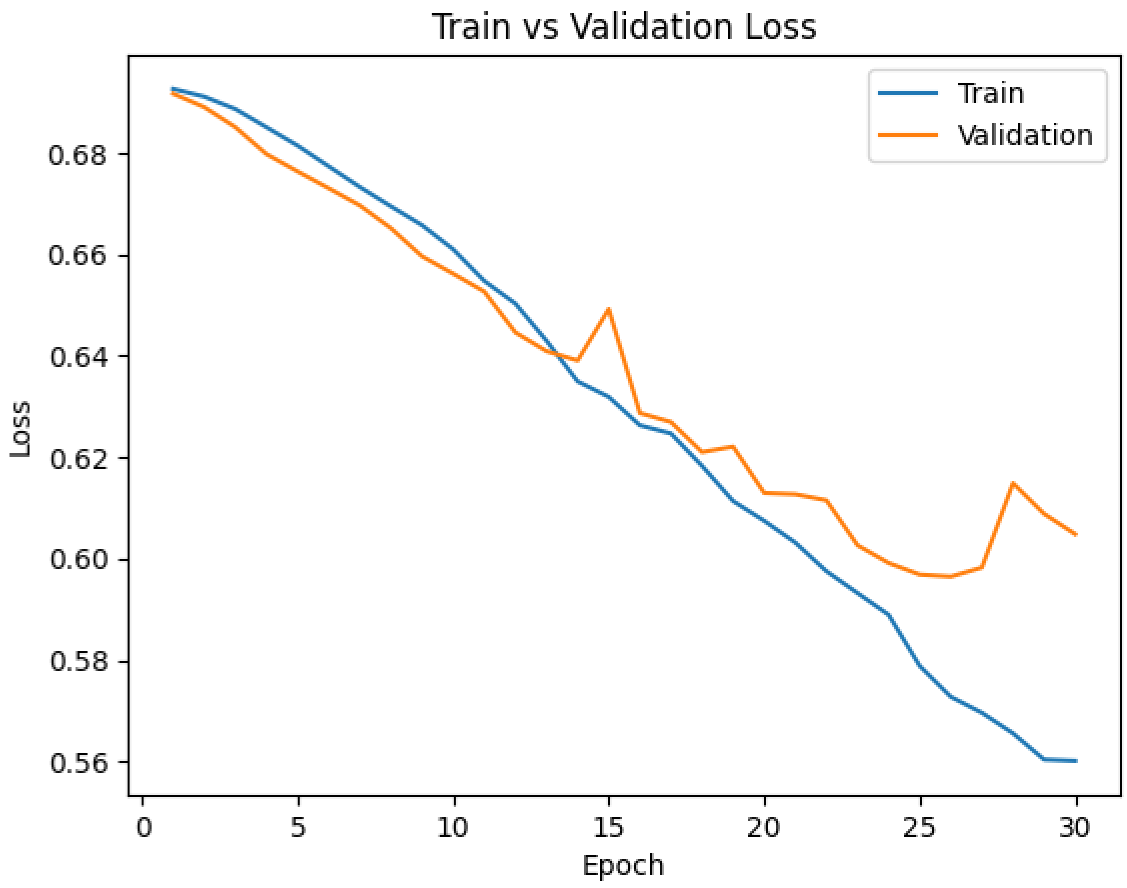
Train the model with the hyperparameters you chose in part(a), and include the training curve.

In [29]:
```
large_net = LargeNet()
train_net(large_net, learning_rate=0.005, batch_size=128)

model_path = get_model_name(large_net.name, 128, 0.005, 29) #assuming 30 epoch
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.466625, Train loss: 0.6925613274649968 |Validation err:
0.4305, Validation loss: 0.6916250251233578
Epoch 2: Train err: 0.450375, Train loss: 0.6910346604528881 |Validation err:
0.4295, Validation loss: 0.6889703720808029
Epoch 3: Train err: 0.43025, Train loss: 0.6885915559435648 |Validation err:
0.417, Validation loss: 0.6850200556218624
Epoch 4: Train err: 0.43075, Train loss: 0.6850066941881937 |Validation err:
0.4135, Validation loss: 0.6797109991312027
Epoch 5: Train err: 0.421125, Train loss: 0.6813858889398121 |Validation err:
0.4105, Validation loss: 0.6762858554720879
Epoch 6: Train err: 0.41575, Train loss: 0.6772955551980033 |Validation err:
0.4125, Validation loss: 0.6729710213840008
Epoch 7: Train err: 0.40525, Train loss: 0.6732198548695397 |Validation err:
0.406, Validation loss: 0.6695918701589108
Epoch 8: Train err: 0.400625, Train loss: 0.6694106298779684 |Validation err:
0.402, Validation loss: 0.6651057451963425
Epoch 9: Train err: 0.390375, Train loss: 0.6656965234922985 |Validation err:
0.3965, Validation loss: 0.6595603972673416
Epoch 10: Train err: 0.38475, Train loss: 0.6609482859808301 |Validation err:
0.3995, Validation loss: 0.6561327464878559
Epoch 11: Train err: 0.3745, Train loss: 0.6547089446158636 |Validation err:
0.389, Validation loss: 0.6526065990328789
Epoch 12: Train err: 0.379375, Train loss: 0.6502427931815858 |Validation err:
0.3815, Validation loss: 0.644555889070034
Epoch 13: Train err: 0.368, Train loss: 0.6429186813415043 |Validation err: 0.
3755, Validation loss: 0.6408654823899269
Epoch 14: Train err: 0.3545, Train loss: 0.6349006607418969 |Validation err:
0.369, Validation loss: 0.6390239223837852
Epoch 15: Train err: 0.3545, Train loss: 0.6318594690353151 |Validation err:
0.387, Validation loss: 0.6491694524884224
Epoch 16: Train err: 0.35075, Train loss: 0.6262631151411269 |Validation err:
0.3565, Validation loss: 0.628677923232317
Epoch 17: Train err: 0.350125, Train loss: 0.6246447080657596 |Validation err:
0.353, Validation loss: 0.6269082017242908
Epoch 18: Train err: 0.343125, Train loss: 0.6182646609487987 |Validation err:
0.3425, Validation loss: 0.6209848523139954
Epoch 19: Train err: 0.336375, Train loss: 0.6113258013649593 |Validation err:
0.3485, Validation loss: 0.6220515631139278
Epoch 20: Train err: 0.33775, Train loss: 0.6074292054252018 |Validation err:
0.3375, Validation loss: 0.6129200533032417
Epoch 21: Train err: 0.329, Train loss: 0.6031468643082513 |Validation err: 0.
339, Validation loss: 0.6126730777323246
Epoch 22: Train err: 0.328625, Train loss: 0.5975030670090328 |Validation err:
0.3295, Validation loss: 0.6114750765264034
Epoch 23: Train err: 0.322, Train loss: 0.5931989011310396 |Validation err: 0.
324, Validation loss: 0.6026201024651527
Epoch 24: Train err: 0.318, Train loss: 0.5889226595560709 |Validation err: 0.
3265, Validation loss: 0.5991240553557873
Epoch 25: Train err: 0.30575, Train loss: 0.5788383200055077 |Validation err:
0.3145, Validation loss: 0.5968427658081055
Epoch 26: Train err: 0.306875, Train loss: 0.5727524052536677 |Validation err:
0.318, Validation loss: 0.5964446403086185
Epoch 27: Train err: 0.302, Train loss: 0.569608471696339 |Validation err: 0.3
15, Validation loss: 0.5982305780053139
Epoch 28: Train err: 0.3005, Train loss: 0.5655598697208223 |Validation err:
0.335, Validation loss: 0.6148588359355927
Epoch 29: Train err: 0.29175, Train loss: 0.5603908352435581 |Validation err:
0.324, Validation loss: 0.6088103912770748
```

```
Epoch 30: Train err: 0.292375, Train loss: 0.5601361673029642 |Validation err:
0.326, Validation loss: 0.604809433221817
Finished Training
Total time elapsed: 791.68 seconds
```





Part (c) - 2pt

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

```
In [ ]: '''
Network: LargeNet
--> Continuing with LargeNet because it provides more capacity
for learning complex patterns compared to SmallNet, which is
beneficial for a dataset like CIFAR-10.

Learning Rate: 0.01
--> This can help speed up the convergence, allowing the model to
learn more quickly, especially when the trainng curves from the previous
hyperparameter showed that the model could benefit from a faster
learning rate without becoing unstable.

Batch size: 256
--> This can help in stablizing the training process by providing more
accurate gradient estimates, leading to a better generalization.'''
```

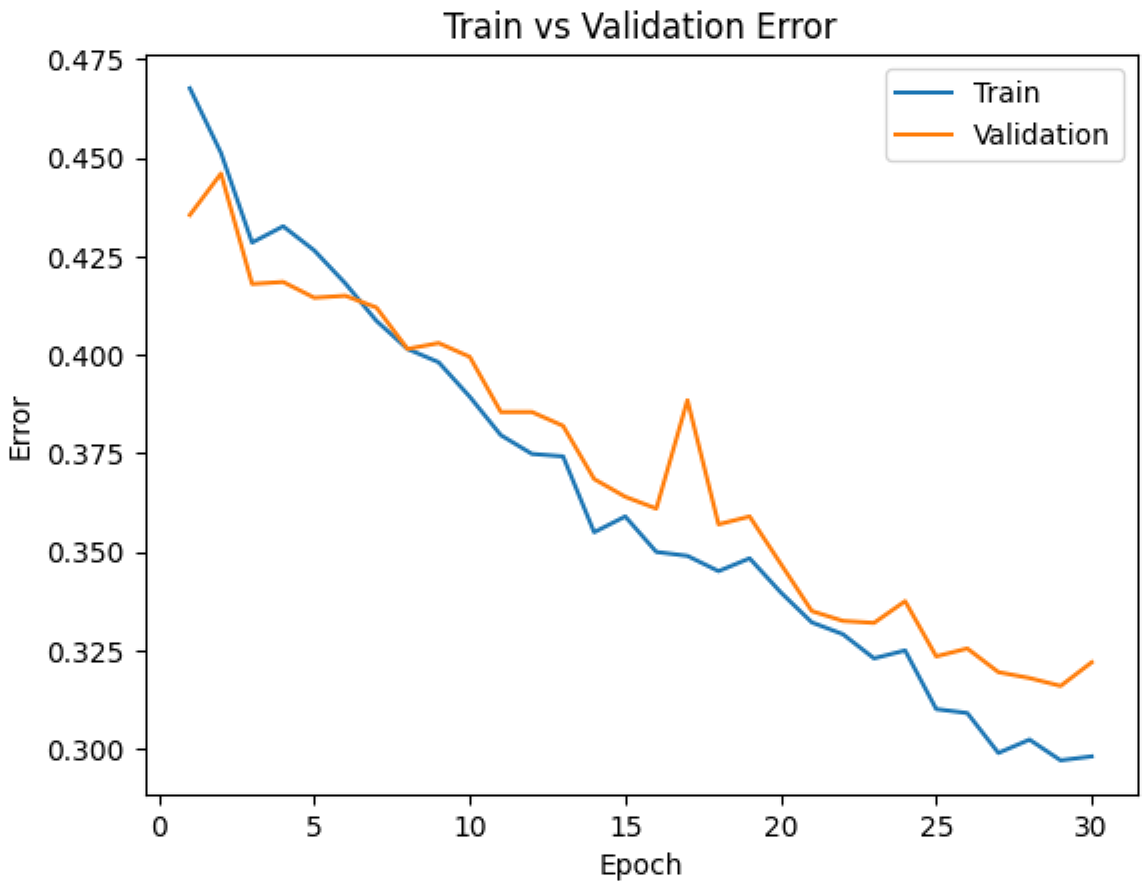
Part (d) - 1pt

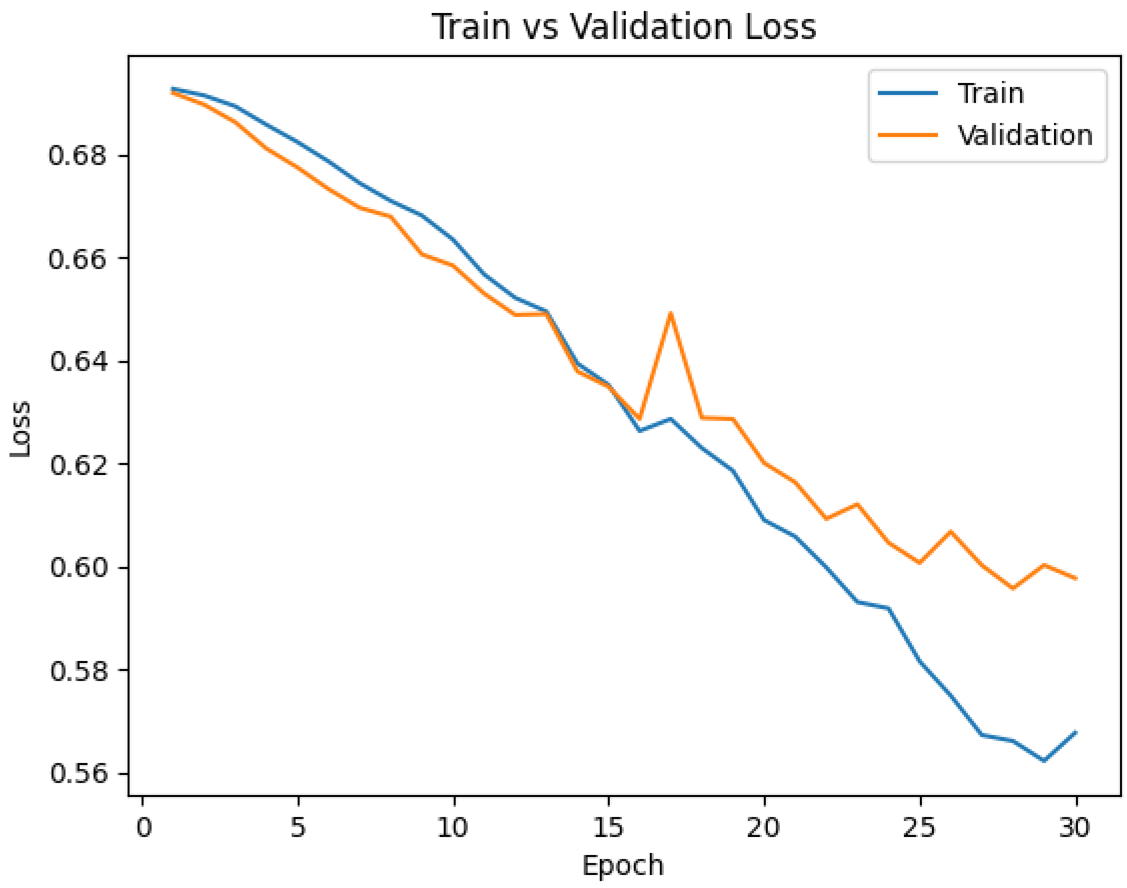
Train the model with the hyperparameters you chose in part(c), and include the training curve.

In [30]:
```
large_net = LargeNet()
train_net(large_net, learning_rate=0.01, batch_size=256)

model_path = get_model_name(large_net.name, 256, 0.01, 29) #assuming 30 epochs
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.467625, Train loss: 0.6926687750965357 |Validation err:
0.4355, Validation loss: 0.6918838322162628
Epoch 2: Train err: 0.45125, Train loss: 0.6913920938968658 |Validation err:
0.446, Validation loss: 0.6896145716309547
Epoch 3: Train err: 0.4285, Train loss: 0.6892960034310818 |Validation err: 0.
418, Validation loss: 0.6862078309059143
Epoch 4: Train err: 0.432625, Train loss: 0.6857281904667616 |Validation err:
0.4185, Validation loss: 0.6810974553227425
Epoch 5: Train err: 0.4265, Train loss: 0.68235875479877 |Validation err: 0.41
45, Validation loss: 0.6774234771728516
Epoch 6: Train err: 0.418125, Train loss: 0.6785927079617977 |Validation err:
0.415, Validation loss: 0.6732236221432686
Epoch 7: Train err: 0.408625, Train loss: 0.6743733454495668 |Validation err:
0.412, Validation loss: 0.6696045845746994
Epoch 8: Train err: 0.4015, Train loss: 0.6709288191050291 |Validation err: 0.
4015, Validation loss: 0.6678666546940804
Epoch 9: Train err: 0.398125, Train loss: 0.6680959537625313 |Validation err:
0.403, Validation loss: 0.660556748509407
Epoch 10: Train err: 0.389375, Train loss: 0.6634314749389887 |Validation err:
0.3995, Validation loss: 0.658379316329956
Epoch 11: Train err: 0.379625, Train loss: 0.6566815190017223 |Validation err:
0.3855, Validation loss: 0.6529561430215836
Epoch 12: Train err: 0.374875, Train loss: 0.6521265283226967 |Validation err:
0.3855, Validation loss: 0.6488026455044746
Epoch 13: Train err: 0.37425, Train loss: 0.6494845673441887 |Validation err:
0.382, Validation loss: 0.6489764079451561
Epoch 14: Train err: 0.355, Train loss: 0.6393899209797382 |Validation err: 0.
3685, Validation loss: 0.6378336399793625
Epoch 15: Train err: 0.359, Train loss: 0.6352421008050442 |Validation err: 0.
364, Validation loss: 0.6348829343914986
Epoch 16: Train err: 0.35, Train loss: 0.626285532489419 |Validation err: 0.36
1, Validation loss: 0.6286248341202736
Epoch 17: Train err: 0.349, Train loss: 0.6286160126328468 |Validation err: 0.
3885, Validation loss: 0.6491348221898079
Epoch 18: Train err: 0.345125, Train loss: 0.6229680832475424 |Validation err:
0.357, Validation loss: 0.6288279891014099
Epoch 19: Train err: 0.348375, Train loss: 0.6185612175613642 |Validation err:
0.359, Validation loss: 0.628604456782341
Epoch 20: Train err: 0.33975, Train loss: 0.6089958921074867 |Validation err:
0.347, Validation loss: 0.620079830288887
Epoch 21: Train err: 0.332125, Train loss: 0.6057972330600023 |Validation err:
0.335, Validation loss: 0.616322323679924
Epoch 22: Train err: 0.329125, Train loss: 0.5998155809938908 |Validation err:
0.3325, Validation loss: 0.6092384159564972
Epoch 23: Train err: 0.323, Train loss: 0.5930873621255159 |Validation err: 0.
332, Validation loss: 0.6120522990822792
Epoch 24: Train err: 0.325, Train loss: 0.5918655060231686 |Validation err: 0.
3375, Validation loss: 0.6046039462089539
Epoch 25: Train err: 0.310125, Train loss: 0.5815890226513147 |Validation err:
0.3235, Validation loss: 0.6006881669163704
Epoch 26: Train err: 0.309125, Train loss: 0.574936468154192 |Validation err:
0.3255, Validation loss: 0.606749102473259
Epoch 27: Train err: 0.299, Train loss: 0.5672708619385958 |Validation err: 0.
3195, Validation loss: 0.6002172753214836
Epoch 28: Train err: 0.302375, Train loss: 0.5661260196939111 |Validation err:
0.318, Validation loss: 0.5957315787672997
Epoch 29: Train err: 0.297125, Train loss: 0.5622713100165129 |Validation err:
0.316, Validation loss: 0.6002558171749115
```

```
Epoch 30: Train err: 0.298125, Train loss: 0.5677183549851179 |Validation err:
0.322, Validation loss: 0.5977195203304291
Finished Training
Total time elapsed: 768.61 seconds
```





Part 4. Evaluating the Best Model [15 pt]

Part (a) - 1pt

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

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

In [32]:
```
best_model = LargeNet()
model_path = get_model_name(best_model.name, batch_size=256, learning_rate=0.0
best_state = torch.load(model_path)
best_model.load_state_dict(best_state)
```

Out[32]:
```
<All keys matched successfully>
```

Part (b) - 2pt

Justify your choice of model from part (a).

In [ ]:
```
'''
I chose the best model based on:

- Time elapsed: the model should take a balance amount of time to
```

```
converge. Somewhere between 760 to 800 seconds.

- Training vs Validation Error Graph: the graph shows a relatively
minimal difference between the two errors, and a peak of 0.446
(relatively low) indicating good genralization; and the errors
generally decrease over epochs.

- Training vs Validation Loss Graph: Similarly, the training and validation
loss curves are close and decreasing steadily (despite a spike at
17th epoch), suggesting the model is improving its performance on
both the training and validation sets.
'''
```

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 [33]:
```
# If you use the `evaluate` function provided in part 0, you will need to
# set batch_size > 1
train_loader, val_loader, test_loader, classes = get_data_loader(
    target_classes=["cat", "dog"],
    batch_size=64)

net = LargeNet()
best_model_path = "model_large_bs256_lr0.01_epoch28"
net.load_state_dict(torch.load(best_model_path))
net.eval()

criterion = nn.BCEWithLogitsLoss()
test_err, test_loss = evaluate(net, test_loader, criterion)

print("Test Classification Error: {:.4f}".format(test_err))
```

```
Files already downloaded and verified
Files already downloaded and verified
Test Classification Error: 0.3180
```

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.

In [ ]:
```
'''
The test classification error is 0.318 and the validation errror is
0.316. This small diffeerne indicates that the model generalizes well
to unseen data. The results are close, suggesting that the model's
performance on the test set is consistent with its performance during
validation.

The test error is typically expected to be higher than the validation
error because the model is often fine-tuned and optimzed based on the
validation set performance, giving a slight bias towards this data.
Additionally, the test set represents completely unseen data, so the
model might perform slightly worse on it. If the model had overfitted
the training data, we would expecct a significant increase in error
```

```
on the test set, but the small discrepancy here indicates good
generalization.
'''
```

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?

In [ ]:
```
'''
We only used the test dataset ata the very end to ensure that our
model evaluation is unbiased and accurately reflects its performance
on truly unseen data. This practice helps to assess the model's
generalization ability, ensuring that it performs well on new and
unseen data.

It is important to use the test data as little as possible to prevent
any indirect influence on the model during training and validation.
Frequent use of the test data coudl lead to overfitting on the test
set. We may obtain a more accurate measure of the model's true
performance and generalizability by keeping the test data separate
and only using it for final evaluation.
'''
```

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.

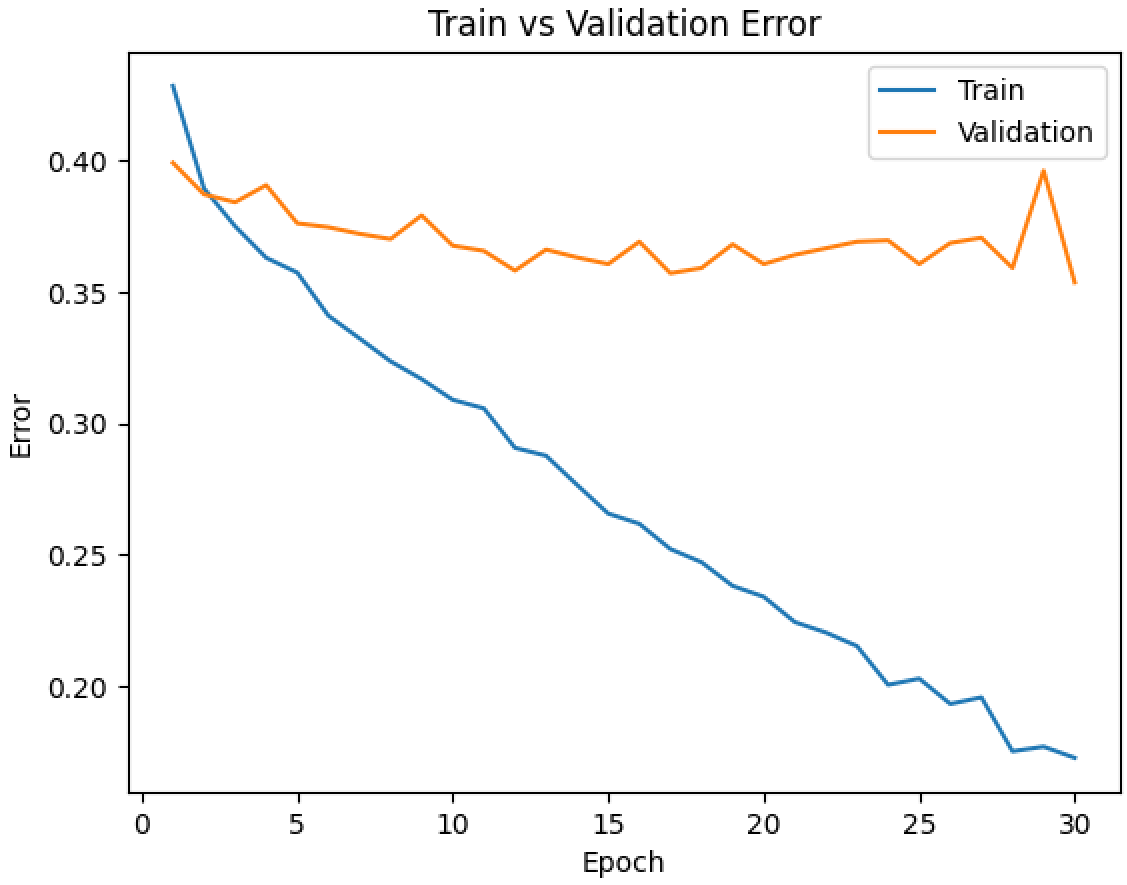
Hint: The ANN in lab 1 was applied on greyscale images. The cat and dog images are colour (RGB) and so you will need to flatted and concatinate all three colour layers before feeding them into an ANN.

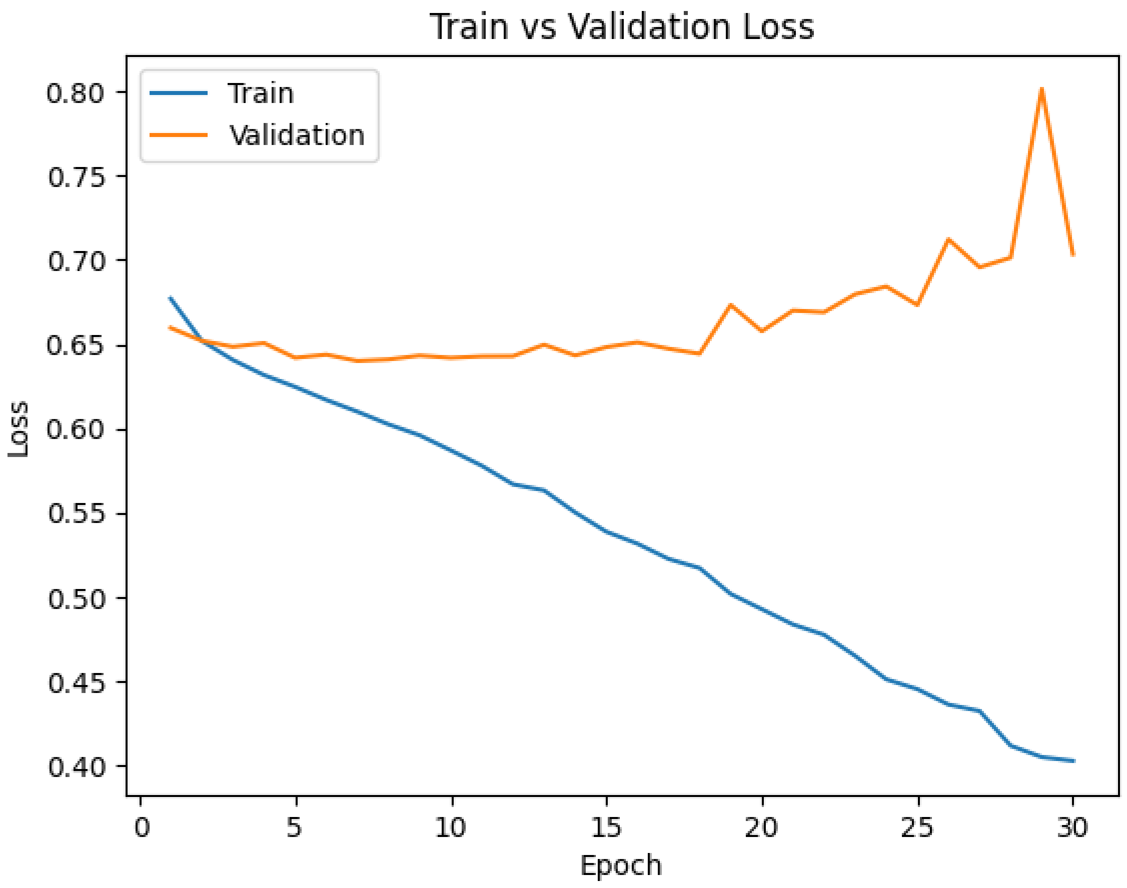
In [6]:
```
torch.manual_seed(1)
#Adapt the ANN for RGB colours
class RGBPigeon(nn.Module):
    def __init__(self, hidden_units):
        super(RGBPigeon, self).__init__()
        self.layer1 = nn.Linear(32*32*3, hidden_units)
        self.layer2 = nn.Linear(hidden_units, 1)
        self.name = "RGBPigeon"

    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
```

```
train_loader, val_loader, test_loader, classes = get_data_loader(
    target_classes=["cat", "dog"],
    batch_size=64
)

hidden_units = 50
learning_rate = 0.01
batch_size = 256
num_epochs = 30

'''Note: the best hyperparameter I found was the third tuning (see below) but
for the comparison's sake I sticked to the same tuning values as CNN's.'''

ann_net = RGBPigeon(hidden_units)
train_net(ann_net, batch_size, learning_rate, num_epochs)

model_name = get_model_name(ann_net.name, batch_size, learning_rate, num_epochs

best_state = torch.load(model_name)
ann_net.load_state_dict(best_state)


plot_training_curve(model_name)


'''
First tuning:
- hidden units: 50
- learning rate: 0.01
- batch_size: 128
_ epochs: 30
--> validation error (lowest): 0.375
--> time elapsed: 767.37s
--> huge gap between train and validation e/l graphs --> lower learning rate
--> validation loss graph does not decrease

Second tuning:
- hidden units: 50
- learning rate: 0.001
- batch_size: 128
_ epochs: 30
--> validation error: 0.387
--> time elapsed: 734.88s
--> smaller gap
--> validation loss graph decreases before plateauing in the end
--> increase batch size

Third tuning:
- hidden units: 50
- learning rate: 0.001
- batch_size: 256
_ epochs: 30
--> validation error: 0.3745
--> time elapsed: 769.85
--> smallest gap
--> validation loss graph mostly decreases
--> choose this hyperparameter

However to compare with CNN's hyperparameter we'll stick to:
- hidden units: 50
- learning rate: 0.01
- batch_size: 256
_ epochs: 30
'''
```

```
Files already downloaded and verified
Files already downloaded and verified
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.42825, Train loss: 0.6769170872867107 |Validation err:
0.399, Validation loss: 0.6595642492175102
Epoch 2: Train err: 0.38925, Train loss: 0.6519035808742046 |Validation err:
0.387, Validation loss: 0.6519387736916542
Epoch 3: Train err: 0.374875, Train loss: 0.640537403523922 |Validation err:
0.384, Validation loss: 0.6483716294169426
Epoch 4: Train err: 0.362875, Train loss: 0.6315739527344704 |Validation err:
0.3905, Validation loss: 0.6505655869841576
Epoch 5: Train err: 0.35725, Train loss: 0.6245692558586597 |Validation err:
0.376, Validation loss: 0.6418333128094673
Epoch 6: Train err: 0.340875, Train loss: 0.6168352533131838 |Validation err:
0.3745, Validation loss: 0.6437233984470367
Epoch 7: Train err: 0.33225, Train loss: 0.6098569743335247 |Validation err:
0.372, Validation loss: 0.6399524658918381
Epoch 8: Train err: 0.3235, Train loss: 0.6023368798196316 |Validation err: 0.
37, Validation loss: 0.6408602595329285
Epoch 9: Train err: 0.31675, Train loss: 0.5958488956093788 |Validation err:
0.379, Validation loss: 0.6431653872132301
Epoch 10: Train err: 0.308875, Train loss: 0.5869064759463072 |Validation err:
0.3675, Validation loss: 0.641803428530693
Epoch 11: Train err: 0.305625, Train loss: 0.5777441300451756 |Validation err:
0.3655, Validation loss: 0.6427131667733192
Epoch 12: Train err: 0.290625, Train loss: 0.5667552649974823 |Validation err:
0.358, Validation loss: 0.6428103223443031
Epoch 13: Train err: 0.287625, Train loss: 0.5631313007324934 |Validation err:
0.366, Validation loss: 0.6496094092726707
Epoch 14: Train err: 0.2765, Train loss: 0.5500430027022958 |Validation err:
0.363, Validation loss: 0.643232561647892
Epoch 15: Train err: 0.265625, Train loss: 0.5386046478524804 |Validation err:
0.3605, Validation loss: 0.6481608599424362
Epoch 16: Train err: 0.26175, Train loss: 0.531502615660429 |Validation err:
0.369, Validation loss: 0.6510078310966492
Epoch 17: Train err: 0.252125, Train loss: 0.5224597165361047 |Validation err:
0.357, Validation loss: 0.6472270339727402
Epoch 18: Train err: 0.247125, Train loss: 0.5170230884104967 |Validation err:
0.359, Validation loss: 0.6442254483699799
Epoch 19: Train err: 0.238125, Train loss: 0.5016351230442524 |Validation err:
0.368, Validation loss: 0.6731988787651062
Epoch 20: Train err: 0.234, Train loss: 0.49260974396020174 |Validation err:
0.3605, Validation loss: 0.6575946733355522
Epoch 21: Train err: 0.224375, Train loss: 0.4835393028333783 |Validation err:
0.364, Validation loss: 0.6698974892497063
Epoch 22: Train err: 0.220375, Train loss: 0.47741825971752405 |Validation er
r: 0.3665, Validation loss: 0.6688803806900978
Epoch 23: Train err: 0.21525, Train loss: 0.4649003678932786 |Validation err:
0.369, Validation loss: 0.6795682236552238
Epoch 24: Train err: 0.2005, Train loss: 0.4509896459057927 |Validation err:
0.3695, Validation loss: 0.6841867566108704
Epoch 25: Train err: 0.202875, Train loss: 0.44514959678053856 |Validation er
r: 0.3605, Validation loss: 0.673137292265892
Epoch 26: Train err: 0.19325, Train loss: 0.43599317874759436 |Validation err:
0.3685, Validation loss: 0.7121472507715225
Epoch 27: Train err: 0.19575, Train loss: 0.4321617428213358 |Validation err:
0.3705, Validation loss: 0.6955582797527313
Epoch 28: Train err: 0.17525, Train loss: 0.4115605400875211 |Validation err:
0.359, Validation loss: 0.7012656480073929
```

```
Epoch 29: Train err: 0.177, Train loss: 0.4048158936202526 |Validation err: 0.
396, Validation loss: 0.801465816795826
Epoch 30: Train err: 0.17275, Train loss: 0.40256923995912075 |Validation err:
0.3535, Validation loss: 0.703335627913475
Finished Training
Total time elapsed: 767.92 seconds
```





```
Out[6]: "\nFirst tuning:\n- hidden units: 50\n- learning rate: 0.01\n- batch_size: 128
\n_ epochs: 30\n--> validation error (lowest): 0.375\n--> time elapsed: 767.37
s\n--> huge gap between train and validation e/l graphs --> lower learning rat
e\n--> validation loss graph does not decrease\n\nSecond tuning:\n- hidden uni
ts: 50\n- learning rate: 0.001\n- batch_size: 128\n_ epochs: 30\n--> validatio
n error: 0.387\n--> time elapsed: 734.88s \n--> smaller gap\n--> validation lo
ss graph decreases before plateauing in the end\n--> increase batch size\n\nTh
ird tuning:\n- hidden units: 50\n- learning rate: 0.001\n- batch_size: 256\n_
epochs: 30\n--> validation error: 0.3745\n--> time elapsed: 769.85\n--> smalle
st gap\n--> validation loss graph mostly decreases\n--> choose this hyperparam
eter\n\nHowever to compare with CNN's hyperparameter we'll stick to:\n- hidden
units: 50\n- learning rate: 0.01\n- batch_size: 256\n_ epochs: 30\n"
```

```
In [7]: train_loader, val_loader, test_loader, classes = get_data_loader(
            target_classes=["cat", "dog"],
            batch_size=64)

        net = RGBPigeon(hidden_units)
        best_model_path = "model_RGBPigeon_bs256_lr0.01_epoch29" #0.3535
        net.load_state_dict(torch.load(best_model_path))
        net.eval()

        criterion = nn.BCEWithLogitsLoss()
        test_err, test_loss = evaluate(net, test_loader, criterion)

        print("Test Classification Error: {:.4f}".format(test_err))

        '''
        Test error for my best CNN model is lower than the test error for ANN in lab 1
        with 0.3180 compared to 0.3570, suggesting that the CNN model is better suited
        for the task classifying cat and dog images. This difference could be due to
        the CNN's ability to learn spatial hierarchies of features, compared to more
```

```
limited capability of the ANN to capture such complex patterns.
'''
```

```
Files already downloaded and verified
Files already downloaded and verified
Test Classification Error: 0.3570
```

Out[7]:

```
"\nTest error for my best CNN model is lower than the test error for ANN in la
b 1,\nwith 0.3180 compared to 0.3585, suggesting that the CNN model is better
suited\nfor the task classifying cat and dog images. This difference could be
due to \nthe CNN's ability to learn spatial hierarchies of features, compared
to more\nlimited capability of the ANN to capture such complex patterns. \n"
```