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://drive.google.com/file/d/1Jh3ccWjwV25auWF2VMnLuLJ3mLMytfNc/view?usp=sharing

In [1]:
```
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 [2]:
```
##############################################################################
# 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 [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
```

```
100%|██████████| 170M/170M [00:02<00:00, 61.6MB/s]
```

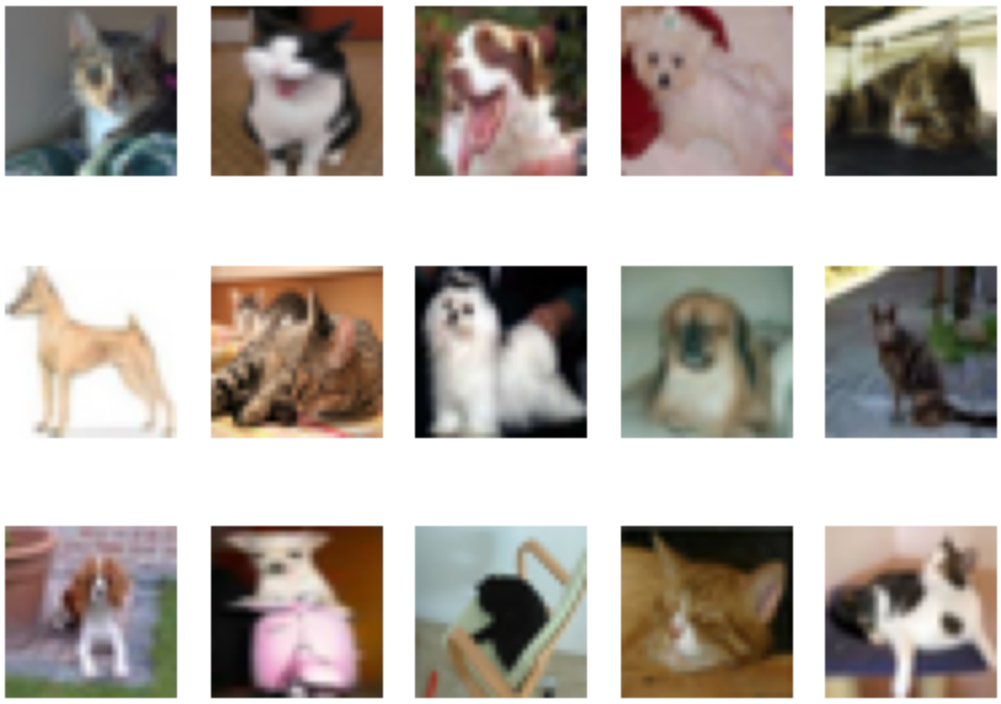
Part (a) -- 1 pt

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 [5]:
```
print("Number of training examples:", len(train_loader))
print("Number of validation examples:", len(val_loader))
print("Number of test examples:", len(test_loader))
```

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

Part (c) -- 3pt

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

A validation set is important because it helps with a model's generalization when training, helps tune hyperparameters, and detect overfitting. If you only use training loss/error for the results, you risk overfitting, since the model starts to memorize the training data but fail on data they haven't seen before. The validation set includes different data from the one we use to train so it makes sure that the model sees new data.

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 [6]:
```
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 [7]:
```
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 [8]:
```
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 [9]:
```
for param in small_net.parameters():
    print(param.shape)
print("====================")
for param in large_net.parameters():
    print(param.shape)
```

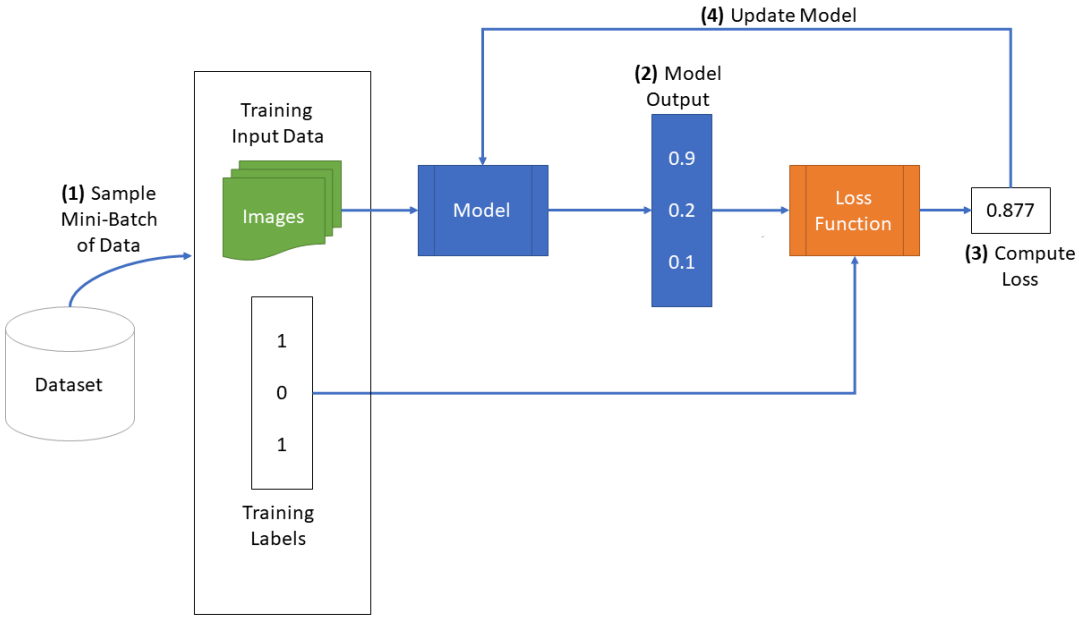
```
torch.Size([5, 3, 3, 3])
torch.Size([5])
torch.Size([1, 245])
torch.Size([1])
====================
torch.Size([5, 3, 5, 5])
torch.Size([5])
torch.Size([10, 5, 5, 5])
torch.Size([10])
torch.Size([32, 250])
torch.Size([32])
torch.Size([1, 32])
torch.Size([1])
```

In [10]:
```
SmallNet
'''
5x3x3x3 = 135
      5 = 5
  1x245 = 245
      1 = 1

Total parameters for SmallNet = 135+5+245+1 = 386
'''

LargeNet
'''
 5x3x5x5 = 375
       5 = 5
10x5x5x5 = 1250
      10 = 10
  32x250 = 8000
      32 = 32
    1x32 = 32
       1 = 1

Total parameters for LargeNet = 375+5+1250+10+8000+32+32+1 = 9705
'''
```

Out[10]:
```
'\n 5x3x5x5 = 375\n       5 = 5\n10x5x5x5 = 1250\n      10 = 10\n  32x250 = 8000\n      32 = 32\n    1x32 = 32\n       1 = 1\n\nTotal parameters for LargeNet = 375+5+1250+10+8000+32+32+1 = 9705\n'
```

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 [11]:
```
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`?

In [12]:
```
#The default values of the parameters batch_size=64, learning_rate=0.01, and
```

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 [13]:
```
train_net(small_net, num_epochs=5)
```

```
Epoch 1: Train err: 0.430625, Train loss: 0.6790979986190796 |Validation er
r: 0.3755, Validation loss: 0.6561036501079798
Epoch 2: Train err: 0.363, Train loss: 0.6424457354545593 |Validation err:
0.3705, Validation loss: 0.6553929131478071
Epoch 3: Train err: 0.342625, Train loss: 0.622077455997467 |Validation err:
0.3365, Validation loss: 0.617309944704175
Epoch 4: Train err: 0.33025, Train loss: 0.6028232989311219 |Validation err:
0.3435, Validation loss: 0.6167509537190199
Epoch 5: Train err: 0.31275, Train loss: 0.5910052700042725 |Validation err:
0.3135, Validation loss: 0.6073005832731724
Finished Training
Total time elapsed: 26.92 seconds
```

There was a total of 4 files written on the disk during each epoch

1. A file with the Training error
2. A file with the Training loss
3. A file with the Validation error
4. A file with the Validation loss

There are also 5 model checkpoints saved after every epoch

Part (d) -- 2pt

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

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

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

In [14]:
```
# 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')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, ca
ll drive.mount("/content/gdrive", force_remount=True).
```

In [15]:
```
train_net(small_net)
train_net(large_net)
```

```
Epoch 1: Train err: 0.30275, Train loss: 0.5800099325180054 |Validation err:
0.3125, Validation loss: 0.6017912533134222
Epoch 2: Train err: 0.294875, Train loss: 0.5722596468925476 |Validation er
r: 0.321, Validation loss: 0.6035115197300911
Epoch 3: Train err: 0.2905, Train loss: 0.5670961375236512 |Validation err:
0.309, Validation loss: 0.5905096214264631
Epoch 4: Train err: 0.29175, Train loss: 0.562696278333664 |Validation err:
0.311, Validation loss: 0.5952892387285829
Epoch 5: Train err: 0.287, Train loss: 0.5594099678993225 |Validation err:
0.308, Validation loss: 0.5886095650494099
Epoch 6: Train err: 0.28675, Train loss: 0.5538390901088714 |Validation err:
0.3085, Validation loss: 0.5968935443088412
Epoch 7: Train err: 0.28475, Train loss: 0.5551524124145508 |Validation err:
0.3075, Validation loss: 0.5884340517222881
Epoch 8: Train err: 0.283875, Train loss: 0.551520301103592 |Validation err:
0.3015, Validation loss: 0.5844288142398
Epoch 9: Train err: 0.280625, Train loss: 0.5523941254615784 |Validation er
r: 0.3125, Validation loss: 0.5903530102223158
Epoch 10: Train err: 0.28225, Train loss: 0.5468608279228211 |Validation er
r: 0.301, Validation loss: 0.5810361979529262
Epoch 11: Train err: 0.27975, Train loss: 0.5456312789916992 |Validation er
r: 0.293, Validation loss: 0.580539702437818
Epoch 12: Train err: 0.273, Train loss: 0.5424110586643219 |Validation err:
0.2995, Validation loss: 0.5909540029242635
Epoch 13: Train err: 0.27575, Train loss: 0.5448757596015931 |Validation er
r: 0.2985, Validation loss: 0.5810548672452569
Epoch 14: Train err: 0.2745, Train loss: 0.540036814212799 |Validation err:
0.2965, Validation loss: 0.596900125965476
Epoch 15: Train err: 0.26925, Train loss: 0.5382293922901153 |Validation er
r: 0.308, Validation loss: 0.5854612495750189
Epoch 16: Train err: 0.282, Train loss: 0.5445845577716827 |Validation err:
0.2995, Validation loss: 0.5862892773002386
Epoch 17: Train err: 0.27, Train loss: 0.5417836573123932 |Validation err:
0.2975, Validation loss: 0.5757443737238646
Epoch 18: Train err: 0.271375, Train loss: 0.5358350365161896 |Validation er
r: 0.3005, Validation loss: 0.5787800960242748
Epoch 19: Train err: 0.26975, Train loss: 0.5349725389480591 |Validation er
r: 0.2945, Validation loss: 0.5846672020852566
Epoch 20: Train err: 0.270625, Train loss: 0.5350171585083008 |Validation er
r: 0.2925, Validation loss: 0.5856741303578019
Epoch 21: Train err: 0.27225, Train loss: 0.5366043448448181 |Validation er
r: 0.291, Validation loss: 0.5776203498244286
Epoch 22: Train err: 0.27025, Train loss: 0.5353422133922577 |Validation er
r: 0.2975, Validation loss: 0.5953578688204288
Epoch 23: Train err: 0.274625, Train loss: 0.5367188413143158 |Validation er
r: 0.2985, Validation loss: 0.5913556227460504
Epoch 24: Train err: 0.264375, Train loss: 0.5319479484558105 |Validation er
r: 0.292, Validation loss: 0.5866347486153245
Epoch 25: Train err: 0.2655, Train loss: 0.5309536666870117 |Validation err:
0.289, Validation loss: 0.5799390431493521
Epoch 26: Train err: 0.267, Train loss: 0.5303733365535737 |Validation err:
0.298, Validation loss: 0.5792311681434512
Epoch 27: Train err: 0.270375, Train loss: 0.5312690494060517 |Validation er
r: 0.29, Validation loss: 0.5887622069567442
Epoch 28: Train err: 0.268375, Train loss: 0.5342589600086212 |Validation er
r: 0.2915, Validation loss: 0.5803237482905388
```

```
Epoch 29: Train err: 0.26725, Train loss: 0.5317573506832123 |Validation er
r: 0.292, Validation loss: 0.5876768659800291
Epoch 30: Train err: 0.2665, Train loss: 0.534029417514801 |Validation err:
0.2885, Validation loss: 0.573572694323957
Finished Training
Total time elapsed: 148.76 seconds
Epoch 1: Train err: 0.464625, Train loss: 0.6922771043777466 |Validation er
r: 0.438, Validation loss: 0.6854696627706289
Epoch 2: Train err: 0.430875, Train loss: 0.6825794444084168 |Validation er
r: 0.417, Validation loss: 0.6757657453417778
Epoch 3: Train err: 0.407375, Train loss: 0.6706802005767822 |Validation er
r: 0.4, Validation loss: 0.6565671898424625
Epoch 4: Train err: 0.3775, Train loss: 0.6528316922187806 |Validation err:
0.3555, Validation loss: 0.6377943493425846
Epoch 5: Train err: 0.351125, Train loss: 0.6315271453857422 |Validation er
r: 0.3375, Validation loss: 0.6186982095241547
Epoch 6: Train err: 0.331, Train loss: 0.6115642511844634 |Validation err:
0.318, Validation loss: 0.6024353094398975
Epoch 7: Train err: 0.322, Train loss: 0.5974183177947998 |Validation err:
0.3115, Validation loss: 0.5896804165095091
Epoch 8: Train err: 0.3075, Train loss: 0.5813269729614258 |Validation err:
0.306, Validation loss: 0.5881929295137525
Epoch 9: Train err: 0.297625, Train loss: 0.5710512759685517 |Validation er
r: 0.3205, Validation loss: 0.5905983317643404
Epoch 10: Train err: 0.289375, Train loss: 0.5579595160484314 |Validation er
r: 0.2995, Validation loss: 0.5798544567078352
Epoch 11: Train err: 0.278125, Train loss: 0.5465031101703643 |Validation er
r: 0.3095, Validation loss: 0.5983587801456451
Epoch 12: Train err: 0.270625, Train loss: 0.5340203671455384 |Validation er
r: 0.31, Validation loss: 0.5983828343451023
Epoch 13: Train err: 0.263, Train loss: 0.5274744827747345 |Validation err:
0.307, Validation loss: 0.5847445726394653
Epoch 14: Train err: 0.26125, Train loss: 0.5151024074554443 |Validation er
r: 0.3125, Validation loss: 0.592924851924181
Epoch 15: Train err: 0.249, Train loss: 0.5033743464946747 |Validation err:
0.291, Validation loss: 0.5780315808951855
Epoch 16: Train err: 0.24825, Train loss: 0.5028959078788757 |Validation er
r: 0.3115, Validation loss: 0.5901016071438789
Epoch 17: Train err: 0.247125, Train loss: 0.4959986355304718 |Validation er
r: 0.3045, Validation loss: 0.5802422277629375
Epoch 18: Train err: 0.23225, Train loss: 0.477262264251709 |Validation err:
0.305, Validation loss: 0.5946605931967497
Epoch 19: Train err: 0.22975, Train loss: 0.471501832485199 |Validation err:
0.3045, Validation loss: 0.5921727316454053
Epoch 20: Train err: 0.22425, Train loss: 0.46158327889442446 |Validation er
r: 0.3105, Validation loss: 0.6510420683771372
Epoch 21: Train err: 0.219, Train loss: 0.45723691368103025 |Validation err:
0.297, Validation loss: 0.5931584350764751
Epoch 22: Train err: 0.212875, Train loss: 0.4476243801116943 |Validation er
r: 0.292, Validation loss: 0.5842673517763615
Epoch 23: Train err: 0.211875, Train loss: 0.43993136978149416 |Validation e
rr: 0.3015, Validation loss: 0.5934903426095843
Epoch 24: Train err: 0.19725, Train loss: 0.42940050101280214 |Validation er
r: 0.301, Validation loss: 0.5997317293658853
Epoch 25: Train err: 0.191625, Train loss: 0.41229853296279906 |Validation e
rr: 0.2825, Validation loss: 0.6056713238358498
```

```
Epoch 26: Train err: 0.1815, Train loss: 0.40324258267879487 |Validation er
r: 0.311, Validation loss: 0.6400953726843
Epoch 27: Train err: 0.176875, Train loss: 0.38879777693748474 |Validation e
rr: 0.2975, Validation loss: 0.6234819376841187
Epoch 28: Train err: 0.178, Train loss: 0.38754239642620086 |Validation err:
0.2935, Validation loss: 0.6200641160830855
Epoch 29: Train err: 0.166625, Train loss: 0.3684359495639801 |Validation er
r: 0.307, Validation loss: 0.7366922665387392
Epoch 30: Train err: 0.167125, Train loss: 0.36464789855480195 |Validation e
rr: 0.298, Validation loss: 0.6570104565471411
Finished Training
Total time elapsed: 169.88 seconds
```

In [16]:
```
'''
*Just an FYI- the values I wrote as the answers and the ones I've written fo
Since then, I ran them multiple times, so the values might be a little diffe

Total elapsed time for SmallNet: 142.53 seconds
Total elapsed time for LargeNet: 162.18 seconds

Training LargeNet took longer because it has more parameters. It has 9705 wh
Since it has more parameters, it needs to do more work to train it, which wi
'''
```

Out[16]:
```
'\nTotal elapsed time for SmallNet: 142.53 seconds\nTotal elapsed time for
LargeNet: 162.18 seconds\n\nTraining LargeNet took longer because it has mo
re parameters. It has 9705 while SmallNet only has 386 (from part A).\nSinc
e it has more parameters, it needs to do more work to train it, which will
take longer.\n'
```

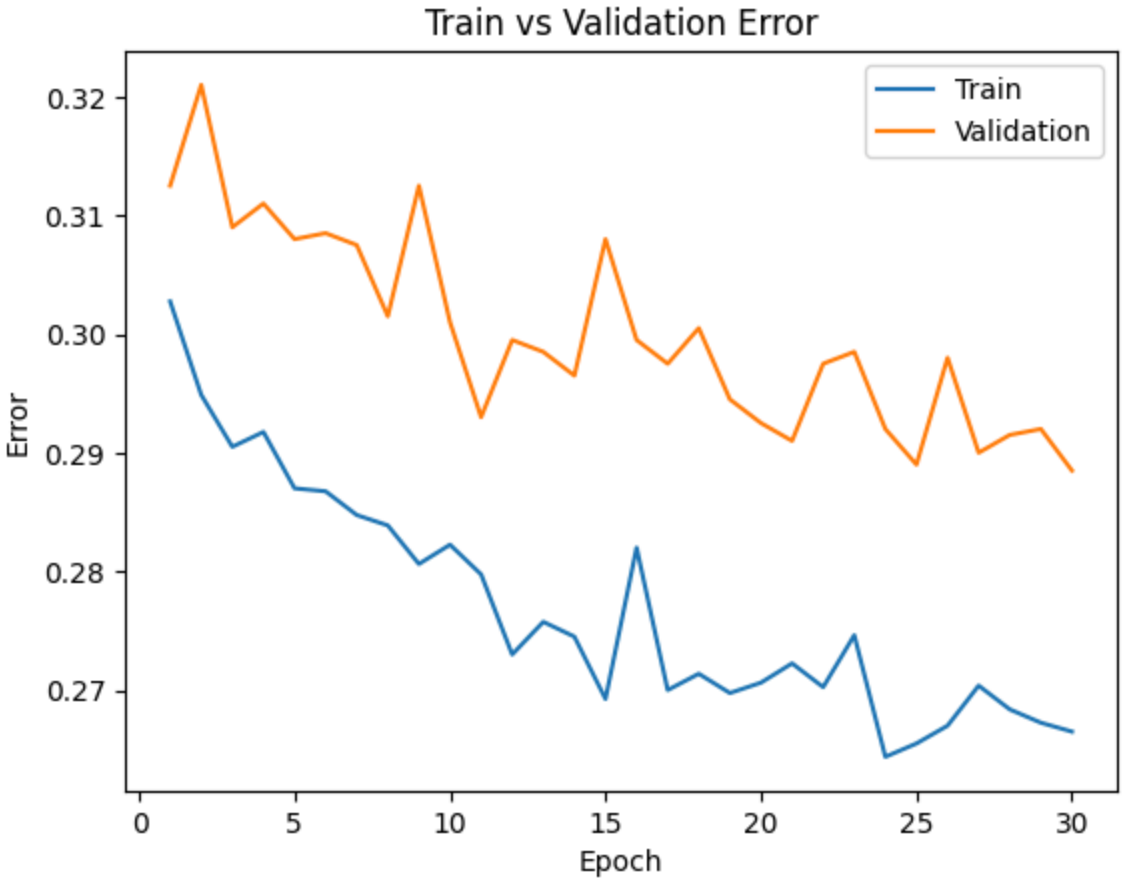
Part (e) - 2pt

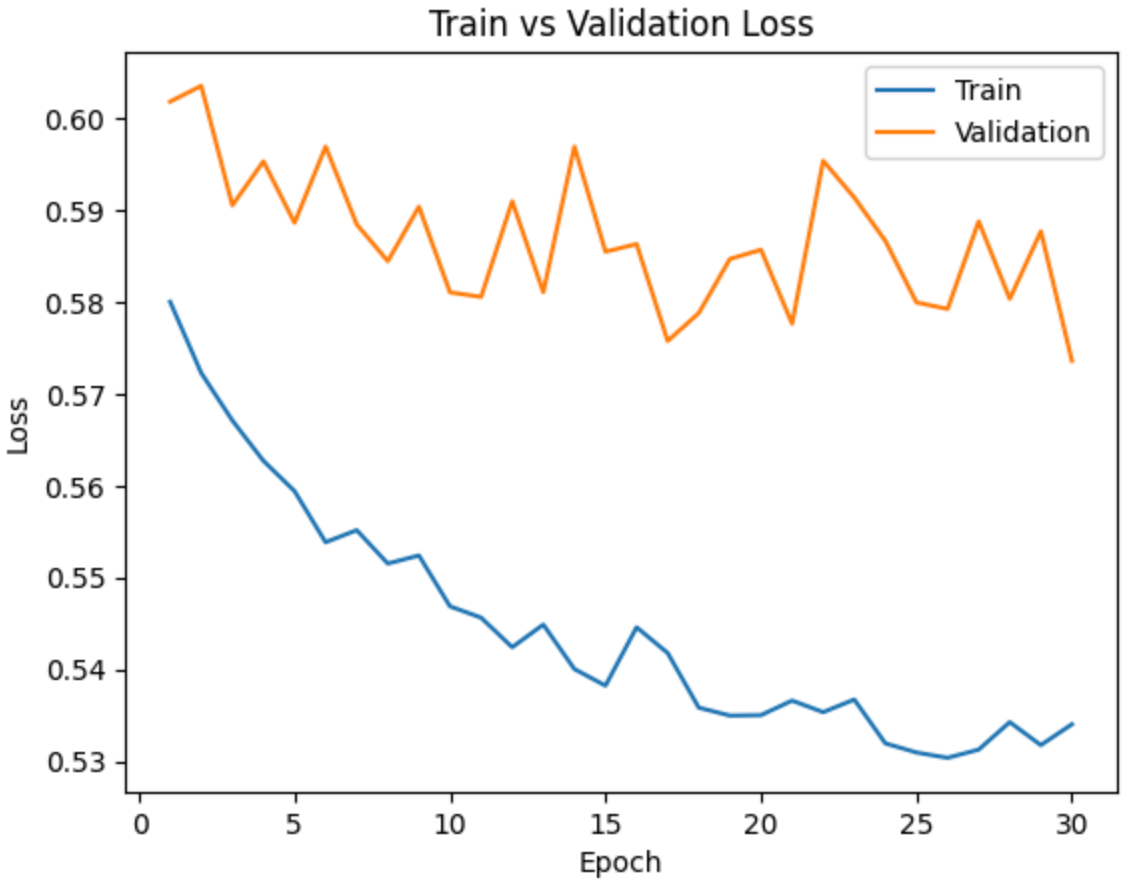
Use the function `plot_training_curve` to display the trajectory of the training/validation error and the training/validation loss. You will need to use the function `get_model_name` to generate the argument to the `plot_training_curve` function.

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

In [17]:
```
#model_path = get_model_name("small", batch_size=??, learning_rate=??, epoch

print("Small Net")
small_net_path = get_model_name("small", batch_size=64, learning_rate=0.01,
plot_training_curve(small_net_path)

print("Large Net")
large_net_path = get_model_name("large", batch_size=64, learning_rate=0.01,
plot_training_curve(large_net_path)
```

```
Small Net
```


Train vs Validation Error
Train
Validation
Error
Epoch
0.32
0.31
0.30
0.29
0.28
0.27
0
5
10
15
20
25
30


Train vs Validation Loss
Train
Validation
Loss
Epoch
0.60
0.59
0.58
0.57
0.56
0.55
0.54
0.53
0
5
10
15
20
25
30

Large Net


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


Train vs Validation Loss
Train
Validation
Loss
Epoch
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
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.

Answer

SmallNet: Looking at these graphs I can see that the validation curves for both have a constant gap between them and the training curve, which means that the model continues to do better on the training data than the unseen one. After around epoch 15, the validation curves start to flatten out while the training curve still continues to decrease, which means it is overfitting with poor generalization. These validation curves also fluctuate a lot, which could mean that the model is very sensitive to small changes in data it hasn't seen yet.

LargeNet: Looking at these graphs I can see that the training data continues to decrease continuously after every epoch. The validation graphs however, both initially decrease with the training curve, which means it is learning, but then starts to increase a lot. The gap between the 2 curves starts to grow when a certain epoch is executed (around epoch 13). This tells us that when the curve starts to separate from one another, the model is greatly overfitting because validation loss starts to increase while the training curve continues to decrease. That is when the model starts to memorize rather than learn.

Part 3. Optimization Parameters [12 pt]

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

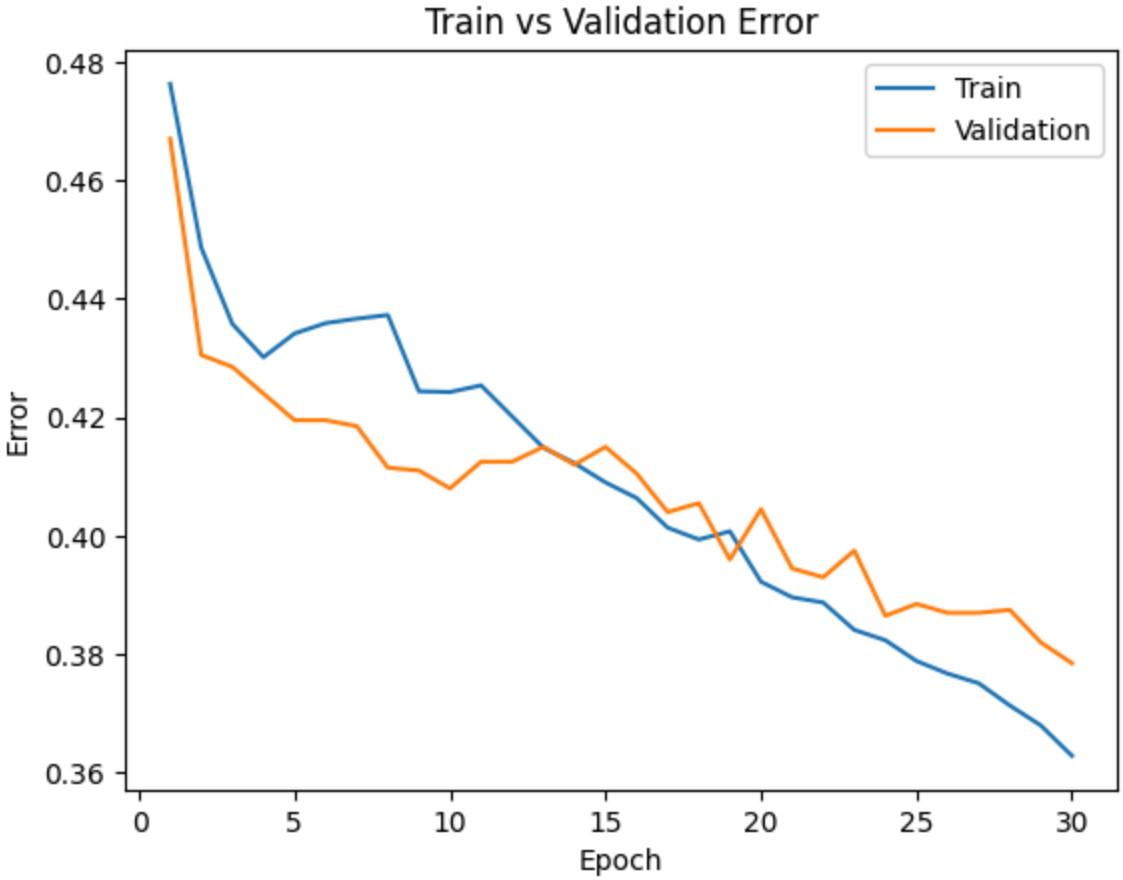
Part (a) - 3pt

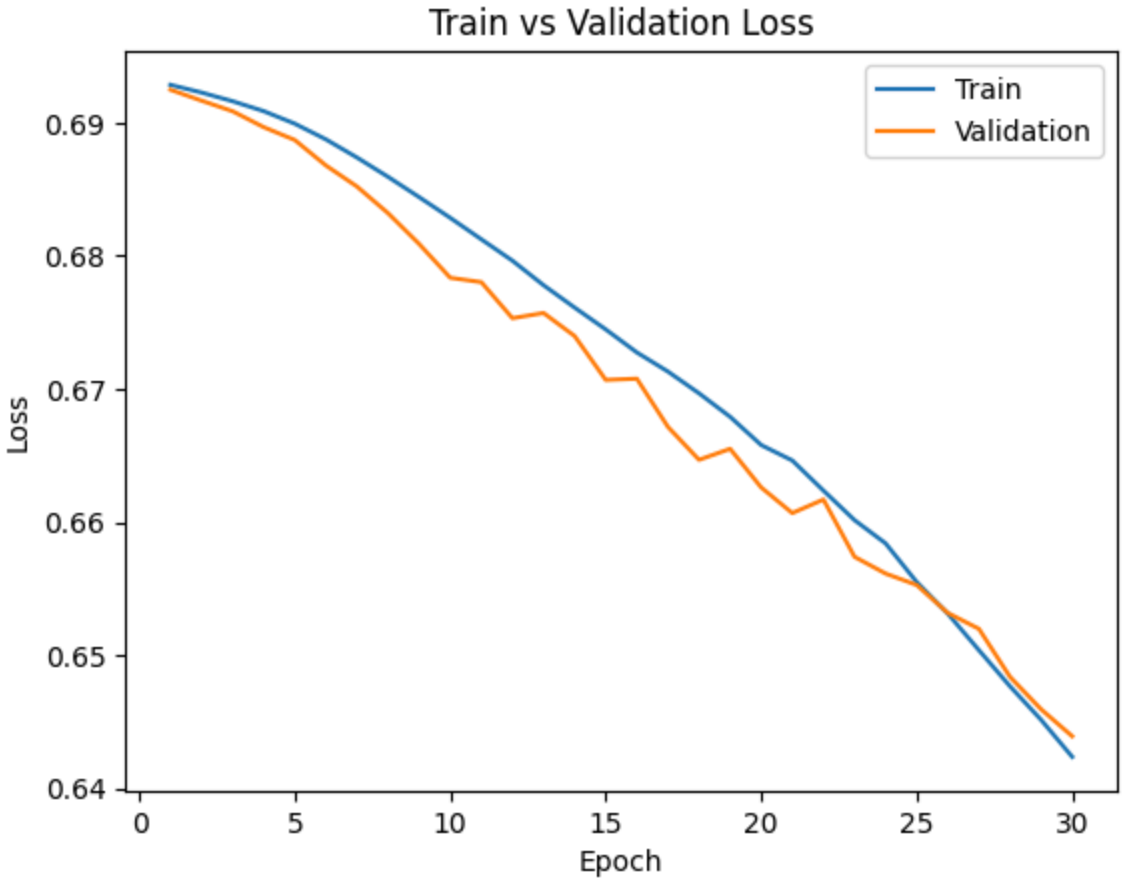
Train `large_net` with all default parameters, except set `learning_rate=0.001`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *lowering* the learning rate.

In [18]:
```
# 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)
plot_training_curve(get_model_name("large", batch_size=64, learning_rate=0.0
```

```
Epoch 1: Train err: 0.47625, Train loss: 0.6928360004425049 |Validation err:
0.467, Validation loss: 0.6924686580896378
Epoch 2: Train err: 0.448625, Train loss: 0.6922589716911316 |Validation er
r: 0.4305, Validation loss: 0.6916493363678455
Epoch 3: Train err: 0.43575, Train loss: 0.6916067404747009 |Validation err:
0.4285, Validation loss: 0.6908544525504112
Epoch 4: Train err: 0.430125, Train loss: 0.6908613877296448 |Validation er
r: 0.424, Validation loss: 0.68965969607234
Epoch 5: Train err: 0.434125, Train loss: 0.6899198365211486 |Validation er
r: 0.4195, Validation loss: 0.688694279640913
Epoch 6: Train err: 0.435875, Train loss: 0.6887419748306275 |Validation er
r: 0.4195, Validation loss: 0.6867837496101856
Epoch 7: Train err: 0.436625, Train loss: 0.6873781814575195 |Validation er
r: 0.4185, Validation loss: 0.6851996649056673
Epoch 8: Train err: 0.43725, Train loss: 0.6859267811775207 |Validation err:
0.4115, Validation loss: 0.6831991244107485
Epoch 9: Train err: 0.424375, Train loss: 0.6844044809341431 |Validation er
r: 0.411, Validation loss: 0.6808866038918495
Epoch 10: Train err: 0.42425, Train loss: 0.6828490204811096 |Validation er
r: 0.408, Validation loss: 0.678350143134594
Epoch 11: Train err: 0.425375, Train loss: 0.6812362918853759 |Validation er
r: 0.4125, Validation loss: 0.6780207101255655
Epoch 12: Train err: 0.420125, Train loss: 0.6796324462890625 |Validation er
r: 0.4125, Validation loss: 0.6753147002309561
Epoch 13: Train err: 0.414875, Train loss: 0.6777921686172486 |Validation er
r: 0.415, Validation loss: 0.6757054049521685
Epoch 14: Train err: 0.41225, Train loss: 0.6761114087104797 |Validation er
r: 0.412, Validation loss: 0.67397416010499
Epoch 15: Train err: 0.409, Train loss: 0.6744724254608154 |Validation err:
0.415, Validation loss: 0.6706814523786306
Epoch 16: Train err: 0.406375, Train loss: 0.6727445869445801 |Validation er
r: 0.4105, Validation loss: 0.6707715895026922
Epoch 17: Train err: 0.401375, Train loss: 0.67130890417099 |Validation err:
0.404, Validation loss: 0.6671548075973988
Epoch 18: Train err: 0.399375, Train loss: 0.6696756863594056 |Validation er
r: 0.4055, Validation loss: 0.6646822784096003
Epoch 19: Train err: 0.40075, Train loss: 0.6679100017547608 |Validation er
r: 0.396, Validation loss: 0.6655073687434196
Epoch 20: Train err: 0.39225, Train loss: 0.665790048122406 |Validation err:
0.4045, Validation loss: 0.6626051049679518
Epoch 21: Train err: 0.389625, Train loss: 0.6646309685707092 |Validation er
r: 0.3945, Validation loss: 0.6606833804398775
Epoch 22: Train err: 0.38875, Train loss: 0.6623744034767151 |Validation er
r: 0.393, Validation loss: 0.6617004871368408
Epoch 23: Train err: 0.384125, Train loss: 0.6601551241874695 |Validation er
r: 0.3975, Validation loss: 0.6574001368135214
Epoch 24: Train err: 0.382375, Train loss: 0.6584072341918945 |Validation er
r: 0.3865, Validation loss: 0.6561403777450323
Epoch 25: Train err: 0.378875, Train loss: 0.6555052490234375 |Validation er
r: 0.3885, Validation loss: 0.6552855931222439
Epoch 26: Train err: 0.37675, Train loss: 0.6531328277587891 |Validation er
r: 0.387, Validation loss: 0.6531829237937927
Epoch 27: Train err: 0.375125, Train loss: 0.6503917617797852 |Validation er
r: 0.387, Validation loss: 0.6519917994737625
Epoch 28: Train err: 0.371375, Train loss: 0.647677414894104 |Validation er
r: 0.3875, Validation loss: 0.6483855955302715
```

```
Epoch 29: Train err: 0.368, Train loss: 0.6451585369110108 |Validation err:
0.382, Validation loss: 0.6459537353366613
Epoch 30: Train err: 0.362875, Train loss: 0.6423822708129883 |Validation er
r: 0.3785, Validation loss: 0.6439289432018995
Finished Training
Total time elapsed: 168.82 seconds
```





Answer

Total elapsed time was 162.41 seconds. Compared to 2d, this took only slightly longer to train, a few milliseconds longer. The output validation curves closely follow the training curves, and there is a steady and constant decrease for both training and validation curves as each epoch increases. This shows that there is better generalization and is neither overfitting nor underfitting.

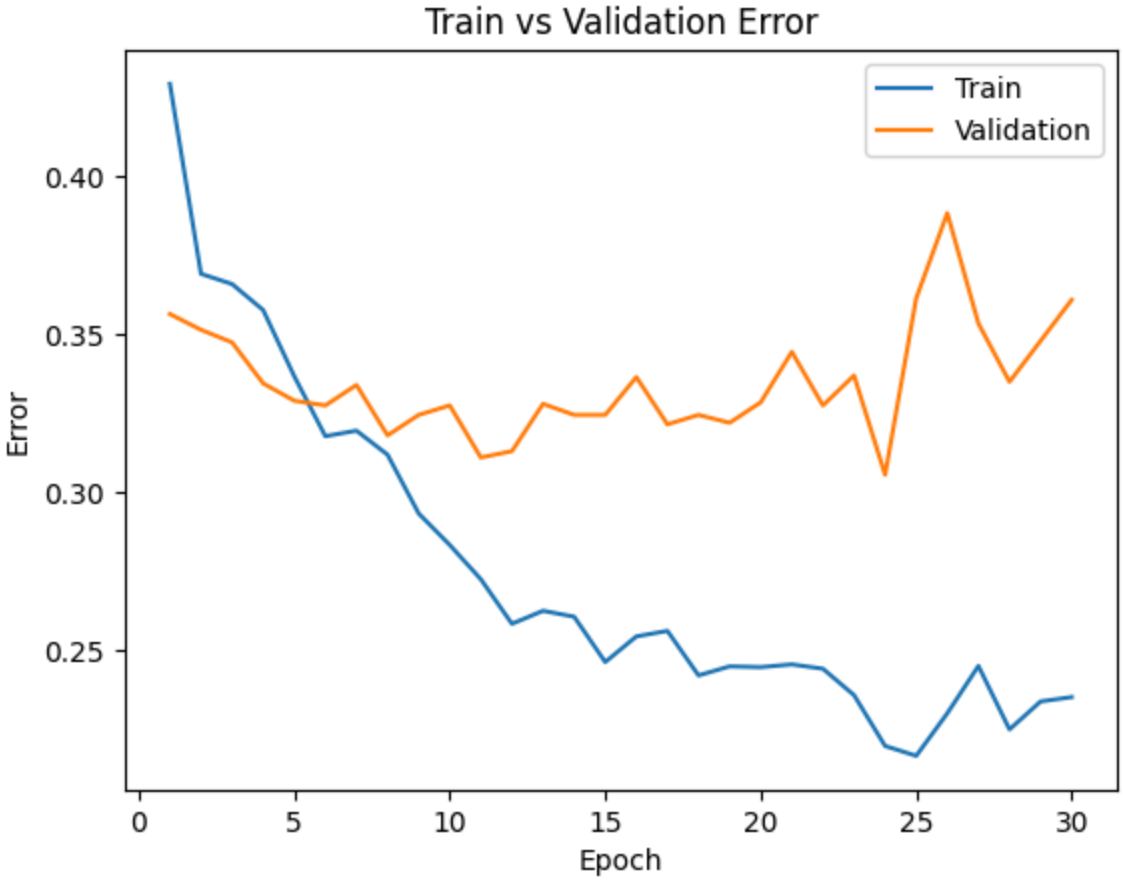
Part (b) - 3pt

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 [19]:
```
large_net = LargeNet()
train_net(large_net, learning_rate=0.1)
plot_training_curve(get_model_name("large", batch_size=64, learning_rate=0.1
```

```
Epoch 1: Train err: 0.4295, Train loss: 0.6743808445930481 |Validation err:
0.3565, Validation loss: 0.6350402887910604
Epoch 2: Train err: 0.36925, Train loss: 0.6415545463562011 |Validation err:
0.3515, Validation loss: 0.6243696268647909
Epoch 3: Train err: 0.366, Train loss: 0.6304733338356018 |Validation err:
0.3475, Validation loss: 0.6173483617603779
Epoch 4: Train err: 0.35775, Train loss: 0.6251179370880127 |Validation err:
0.3345, Validation loss: 0.620793430134654
Epoch 5: Train err: 0.33675, Train loss: 0.6045795600414277 |Validation err:
0.329, Validation loss: 0.5937281344085932
Epoch 6: Train err: 0.31775, Train loss: 0.5828781847953797 |Validation err:
0.3275, Validation loss: 0.6041664052754641
Epoch 7: Train err: 0.3195, Train loss: 0.5794640259742737 |Validation err:
0.334, Validation loss: 0.6052154619246721
Epoch 8: Train err: 0.311875, Train loss: 0.5757701637744903 |Validation er
r: 0.318, Validation loss: 0.5993684772402048
Epoch 9: Train err: 0.29325, Train loss: 0.5609976608753204 |Validation err:
0.3245, Validation loss: 0.5896207233890891
Epoch 10: Train err: 0.28325, Train loss: 0.5477550032138825 |Validation er
r: 0.3275, Validation loss: 0.5987504478543997
Epoch 11: Train err: 0.272375, Train loss: 0.5326508040428162 |Validation er
r: 0.311, Validation loss: 0.6202037688344717
Epoch 12: Train err: 0.25825, Train loss: 0.5120045669078827 |Validation er
r: 0.313, Validation loss: 0.6050943713635206
Epoch 13: Train err: 0.262375, Train loss: 0.5099549624919891 |Validation er
r: 0.328, Validation loss: 0.6595966126769781
Epoch 14: Train err: 0.2605, Train loss: 0.508032098531723 |Validation err:
0.3245, Validation loss: 0.7022156557068229
Epoch 15: Train err: 0.246125, Train loss: 0.4948580477237701 |Validation er
r: 0.3245, Validation loss: 0.6181976906955242
Epoch 16: Train err: 0.25425, Train loss: 0.5112269511222839 |Validation er
r: 0.3365, Validation loss: 0.6773550705984235
Epoch 17: Train err: 0.256, Train loss: 0.5096174550056457 |Validation err:
0.3215, Validation loss: 0.6484736818820238
Epoch 18: Train err: 0.241875, Train loss: 0.48050818586349486 |Validation e
rr: 0.3245, Validation loss: 0.644468174315989
Epoch 19: Train err: 0.24475, Train loss: 0.497201379776001 |Validation err:
0.322, Validation loss: 0.6747357528656721
Epoch 20: Train err: 0.2445, Train loss: 0.4948779764175415 |Validation err:
0.3285, Validation loss: 0.7009244337677956
Epoch 21: Train err: 0.245375, Train loss: 0.49160154938697814 |Validation e
rr: 0.3445, Validation loss: 0.6986064556986094
Epoch 22: Train err: 0.244, Train loss: 0.48209749364852905 |Validation err:
0.3275, Validation loss: 0.7424729950726032
Epoch 23: Train err: 0.235625, Train loss: 0.47574091553688047 |Validation e
rr: 0.337, Validation loss: 0.7379261311143637
Epoch 24: Train err: 0.2195, Train loss: 0.45225173568725585 |Validation er
r: 0.3055, Validation loss: 0.8094820464029908
Epoch 25: Train err: 0.216375, Train loss: 0.44918966364860535 |Validation e
rr: 0.3615, Validation loss: 0.7755749644711614
Epoch 26: Train err: 0.229875, Train loss: 0.46733192253112793 |Validation e
rr: 0.3885, Validation loss: 0.7462155539542437
Epoch 27: Train err: 0.244875, Train loss: 0.4980352530479431 |Validation er
r: 0.3535, Validation loss: 0.7864172533154488
Epoch 28: Train err: 0.22475, Train loss: 0.4615201361179352 |Validation er
r: 0.335, Validation loss: 0.8736669421195984
```

```
Epoch 29: Train err: 0.233625, Train loss: 0.4676371040344238 |Validation er
r: 0.348, Validation loss: 0.7977995797991753
Epoch 30: Train err: 0.235, Train loss: 0.4715475182533264 |Validation err:
0.361, Validation loss: 0.8199478834867477
Finished Training
Total time elapsed: 168.26 seconds
```



Answer

The total elapsed time it took was 165.67 seconds. This took a few seconds longer to train compared to the other values. But you can see the model start to overfit a lot faster. The validation curves start to diverge earlier on.

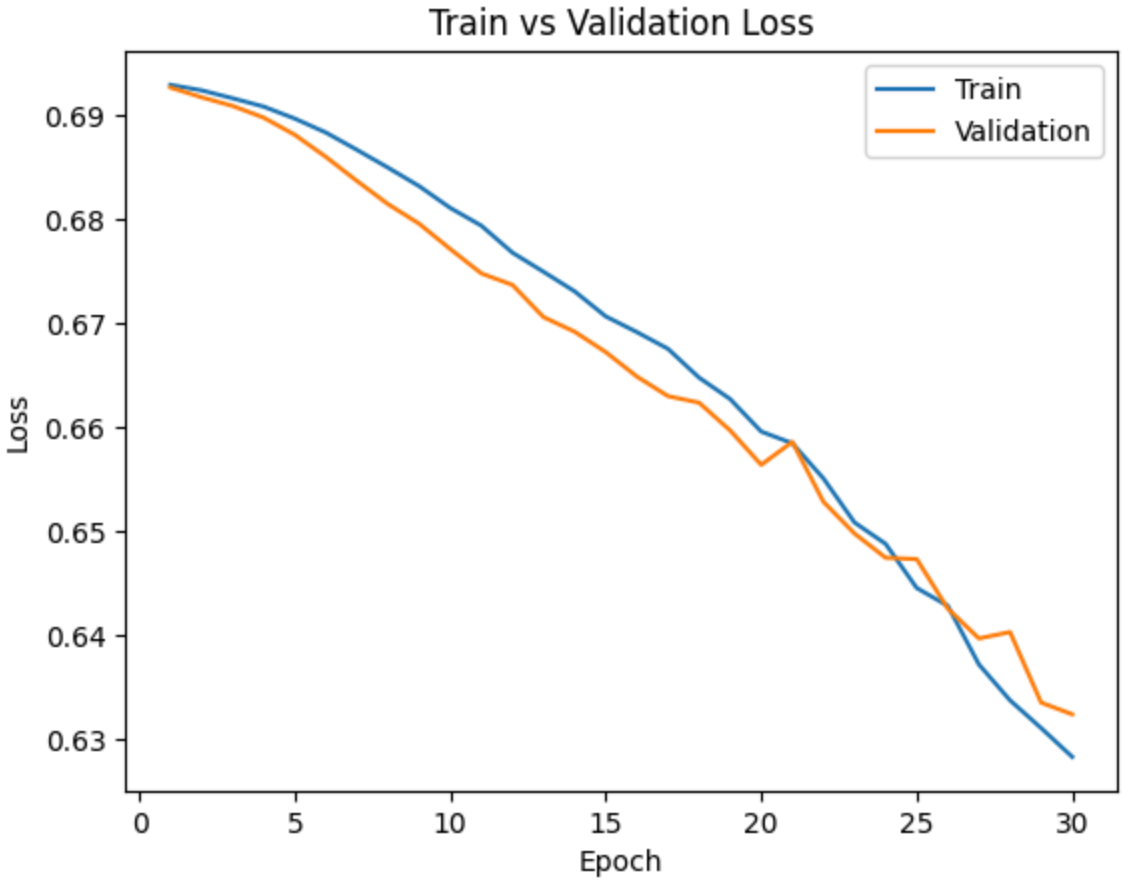
Part (c) - 3pt

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 [20]:
```
large_net = LargeNet()
train_net(large_net, batch_size=512)
plot_training_curve(get_model_name("large", batch_size=512, learning_rate=0.
```

```
Epoch 1: Train err: 0.48175, Train loss: 0.6929379552602768 |Validation err:
0.478, Validation loss: 0.6926824003458023
Epoch 2: Train err: 0.457625, Train loss: 0.692410409450531 |Validation err:
0.434, Validation loss: 0.6917425245046616
Epoch 3: Train err: 0.437, Train loss: 0.6916500627994537 |Validation err:
0.4265, Validation loss: 0.6909130215644836
Epoch 4: Train err: 0.433625, Train loss: 0.6908449903130531 |Validation er
r: 0.424, Validation loss: 0.6897870302200317
Epoch 5: Train err: 0.434, Train loss: 0.6896935515105724 |Validation err:
0.424, Validation loss: 0.6881355047225952
Epoch 6: Train err: 0.438, Train loss: 0.6883532106876373 |Validation err:
0.4285, Validation loss: 0.686011865735054
Epoch 7: Train err: 0.439375, Train loss: 0.6866871826350689 |Validation er
r: 0.426, Validation loss: 0.6836968660354614
Epoch 8: Train err: 0.43525, Train loss: 0.6849770620465279 |Validation err:
0.411, Validation loss: 0.6814672648906708
Epoch 9: Train err: 0.42375, Train loss: 0.6832008883357048 |Validation err:
0.414, Validation loss: 0.6795914173126221
Epoch 10: Train err: 0.421, Train loss: 0.6811088025569916 |Validation err:
0.416, Validation loss: 0.6771541684865952
Epoch 11: Train err: 0.42075, Train loss: 0.6794030703604221 |Validation er
r: 0.4095, Validation loss: 0.6748128235340118
Epoch 12: Train err: 0.41475, Train loss: 0.6768062897026539 |Validation er
r: 0.412, Validation loss: 0.6737067103385925
Epoch 13: Train err: 0.41025, Train loss: 0.6749698966741562 |Validation er
r: 0.412, Validation loss: 0.6706121861934662
Epoch 14: Train err: 0.407125, Train loss: 0.6730909235775471 |Validation er
r: 0.4125, Validation loss: 0.6692100465297699
Epoch 15: Train err: 0.4005, Train loss: 0.6706824786961079 |Validation err:
0.41, Validation loss: 0.6672518998384476
Epoch 16: Train err: 0.397625, Train loss: 0.669177521020174 |Validation er
r: 0.405, Validation loss: 0.6649047136306763
Epoch 17: Train err: 0.393875, Train loss: 0.6675704158842564 |Validation er
r: 0.401, Validation loss: 0.6630221307277679
Epoch 18: Train err: 0.393, Train loss: 0.6648007929325104 |Validation err:
0.3945, Validation loss: 0.6623915135860443
Epoch 19: Train err: 0.386125, Train loss: 0.6627416461706161 |Validation er
r: 0.3875, Validation loss: 0.6597241759300232
Epoch 20: Train err: 0.38175, Train loss: 0.659615233540535 |Validation err:
0.401, Validation loss: 0.6564163863658905
Epoch 21: Train err: 0.385875, Train loss: 0.6584861390292645 |Validation er
r: 0.388, Validation loss: 0.6586255580186844
Epoch 22: Train err: 0.378625, Train loss: 0.6551117822527885 |Validation er
r: 0.3855, Validation loss: 0.6528623402118683
Epoch 23: Train err: 0.371125, Train loss: 0.6508784741163254 |Validation er
r: 0.3835, Validation loss: 0.6498105823993683
Epoch 24: Train err: 0.37675, Train loss: 0.6488191038370132 |Validation er
r: 0.384, Validation loss: 0.647487610578537
Epoch 25: Train err: 0.369, Train loss: 0.6445828042924404 |Validation err:
0.3825, Validation loss: 0.6473540216684341
Epoch 26: Train err: 0.373, Train loss: 0.6428647711873055 |Validation err:
0.375, Validation loss: 0.6425962299108505
Epoch 27: Train err: 0.35975, Train loss: 0.6372342072427273 |Validation er
r: 0.378, Validation loss: 0.6397327333688736
Epoch 28: Train err: 0.354125, Train loss: 0.6337889917194843 |Validation er
r: 0.37, Validation loss: 0.6403442025184631
```

```
Epoch 29: Train err: 0.353875, Train loss: 0.6311212778091431 |Validation er
r: 0.3665, Validation loss: 0.6335662603378296
Epoch 30: Train err: 0.35275, Train loss: 0.6283531747758389 |Validation er
r: 0.367, Validation loss: 0.6324412226676941
Finished Training
Total time elapsed: 148.78 seconds
```



Answer

The total elapsed time is 139.18, which was a lot faster than the other tests - less training time. When you increase the batch size, the validation curves closly follow the training curves indicating the model is generalizing better and not overfitting nor underfitting as much.

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 [21]:
```
large_net = LargeNet()
train_net(large_net, batch_size=16)
plot_training_curve(get_model_name("large", batch_size=16, learning_rate=0.0
```

```
Epoch 1: Train err: 0.432625, Train loss: 0.6775506126880646 |Validation er
r: 0.378, Validation loss: 0.6512571973800659
Epoch 2: Train err: 0.366375, Train loss: 0.6387728816270828 |Validation er
r: 0.337, Validation loss: 0.612742235660553
Epoch 3: Train err: 0.339375, Train loss: 0.6119522891640663 |Validation er
r: 0.345, Validation loss: 0.6396356213092804
Epoch 4: Train err: 0.312125, Train loss: 0.5861616842746734 |Validation er
r: 0.3375, Validation loss: 0.6223122742176056
Epoch 5: Train err: 0.309125, Train loss: 0.5655454085469246 |Validation er
r: 0.303, Validation loss: 0.5682719912528992
Epoch 6: Train err: 0.283875, Train loss: 0.546434996843338 |Validation err:
0.3125, Validation loss: 0.581913836479187
Epoch 7: Train err: 0.275625, Train loss: 0.5315411986708641 |Validation er
r: 0.3185, Validation loss: 0.5755203785896301
Epoch 8: Train err: 0.253625, Train loss: 0.5110043309926987 |Validation er
r: 0.326, Validation loss: 0.6016001827716827
Epoch 9: Train err: 0.249375, Train loss: 0.4988122125864029 |Validation er
r: 0.301, Validation loss: 0.5766945521831512
Epoch 10: Train err: 0.234375, Train loss: 0.47769227081537247 |Validation e
rr: 0.2935, Validation loss: 0.5778298568725586
Epoch 11: Train err: 0.2245, Train loss: 0.4637914804518223 |Validation err:
0.295, Validation loss: 0.6023752751350403
Epoch 12: Train err: 0.208625, Train loss: 0.4436814887523651 |Validation er
r: 0.3015, Validation loss: 0.5988463146686553
Epoch 13: Train err: 0.1995, Train loss: 0.42190410897135733 |Validation er
r: 0.2965, Validation loss: 0.6023776261806488
Epoch 14: Train err: 0.18225, Train loss: 0.3964607211649418 |Validation er
r: 0.285, Validation loss: 0.6563077869415284
Epoch 15: Train err: 0.176625, Train loss: 0.38504845155775547 |Validation e
rr: 0.3015, Validation loss: 0.684573037981987
Epoch 16: Train err: 0.160125, Train loss: 0.35515162971615793 |Validation e
rr: 0.313, Validation loss: 0.7153961782455445
Epoch 17: Train err: 0.146875, Train loss: 0.33127270932495595 |Validation e
rr: 0.3095, Validation loss: 0.8292664550542831
Epoch 18: Train err: 0.1435, Train loss: 0.3216609486192465 |Validation err:
0.3265, Validation loss: 0.8005202133655548
Epoch 19: Train err: 0.130375, Train loss: 0.30764051334559916 |Validation e
rr: 0.3255, Validation loss: 0.8987283756732941
Epoch 20: Train err: 0.11675, Train loss: 0.28437252435833216 |Validation er
r: 0.309, Validation loss: 0.8813227322101593
Epoch 21: Train err: 0.121625, Train loss: 0.27901479678601027 |Validation e
rr: 0.3175, Validation loss: 0.8441186798810959
Epoch 22: Train err: 0.112875, Train loss: 0.2728129132091999 |Validation er
r: 0.3125, Validation loss: 0.8382637614011764
Epoch 23: Train err: 0.1, Train loss: 0.24255631332099437 |Validation err:
0.3265, Validation loss: 1.0690370055437088
Epoch 24: Train err: 0.095375, Train loss: 0.2318725396282971 |Validation er
r: 0.322, Validation loss: 1.0889509381055833
Epoch 25: Train err: 0.095875, Train loss: 0.23257503168098628 |Validation e
rr: 0.329, Validation loss: 1.0898759965896607
Epoch 26: Train err: 0.08275, Train loss: 0.20350570978596808 |Validation er
r: 0.321, Validation loss: 1.127944498538971
Epoch 27: Train err: 0.09075, Train loss: 0.2147584371343255 |Validation er
r: 0.3205, Validation loss: 1.1985667142868042
Epoch 28: Train err: 0.076125, Train loss: 0.18171320636058227 |Validation e
rr: 0.322, Validation loss: 1.3782274899482727
```

```
Epoch 29: Train err: 0.091875, Train loss: 0.21968170262500644 |Validation e
rr: 0.3245, Validation loss: 1.2601391798257828
Epoch 30: Train err: 0.07725, Train loss: 0.1865569370612502 |Validation er
r: 0.323, Validation loss: 1.3147952773571014
Finished Training
Total time elapsed: 250.09 seconds
```

Answer

Total elapsed time was 243.33 seconds, which was significantly slower than the other tests. In addition, the training data continues to decrease while validation error starts to even out and validation loss starts to increase dramatically. Both validation curves start to diverge from the training curve which means it is overfitting a lot faster (it seems - compared to the other trials, starts to overfit at around epoch 5). This means the model is memorizing and not generalizing.

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.

Answer

The hyperparameters values I would choose is (large_net, batch_size=512, learning_rate=0.001), these values, when I tested them above, were the ones where the

validation curves were the closest to the training curves. This way, the model will not drastically overfit/underfit and will be better at generalizing.

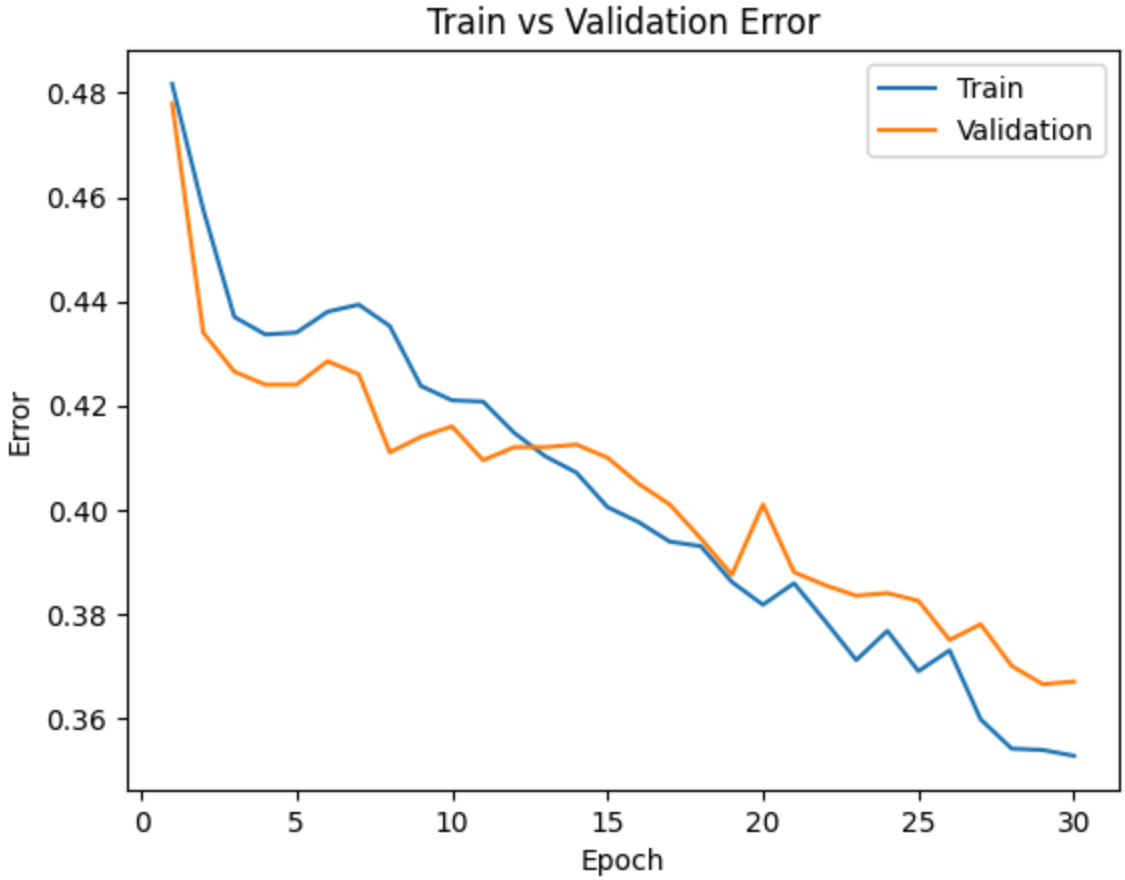
Part (b) - 1pt

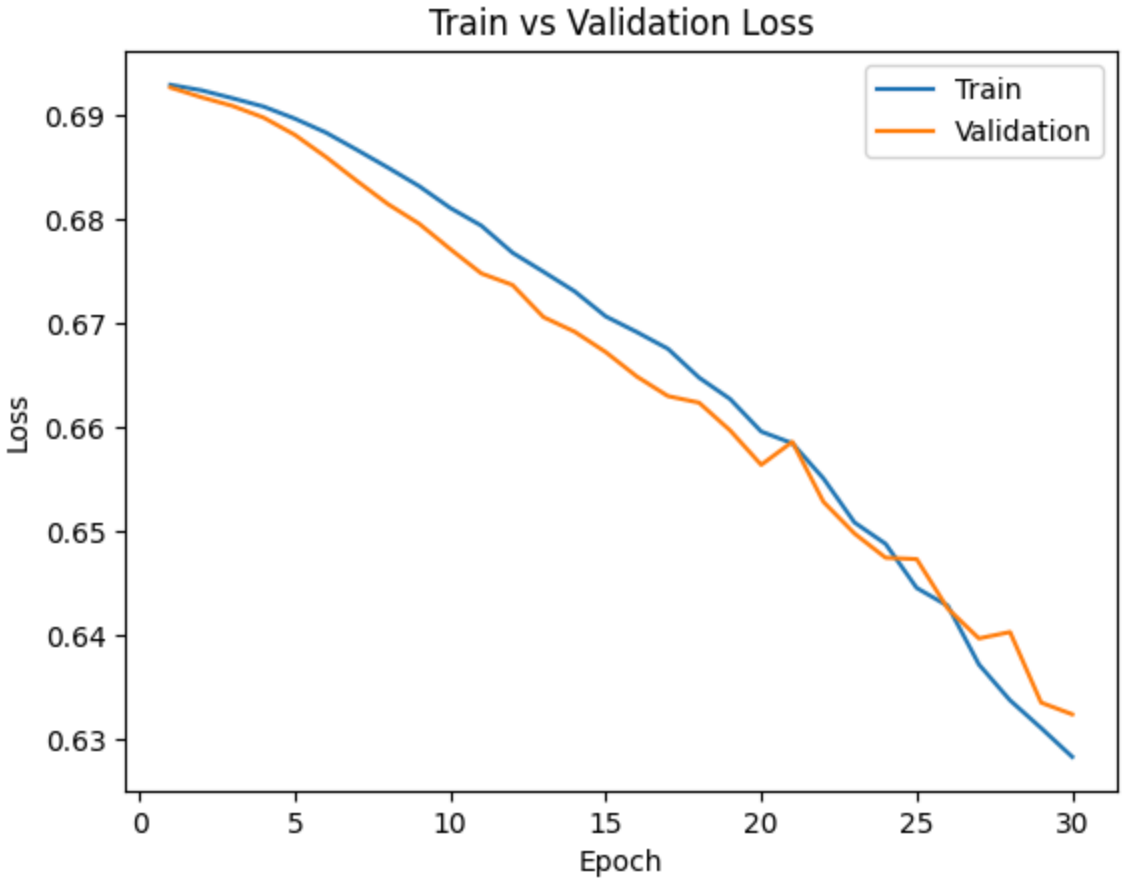
Train the model with the hyperparameters you chose in part(a), and include the training curve.

In [22]:
```
large_net = LargeNet()
train_net(large_net, batch_size=512, learning_rate=0.001)
plot_training_curve(get_model_name("large", batch_size=512, learning_rate=0.
```

```
Epoch 1: Train err: 0.48825, Train loss: 0.6930677518248558 |Validation err:
0.4955, Validation loss: 0.6931361854076385
Epoch 2: Train err: 0.483125, Train loss: 0.692995510995388 |Validation err:
0.4945, Validation loss: 0.6930360347032547
Epoch 3: Train err: 0.480375, Train loss: 0.6929280385375023 |Validation er
r: 0.493, Validation loss: 0.6929539740085602
Epoch 4: Train err: 0.477, Train loss: 0.6928808502852917 |Validation err:
0.4885, Validation loss: 0.692870706319809
Epoch 5: Train err: 0.473375, Train loss: 0.6927744261920452 |Validation er
r: 0.4835, Validation loss: 0.6927504986524582
Epoch 6: Train err: 0.469, Train loss: 0.6926896274089813 |Validation err:
0.472, Validation loss: 0.6926551759243011
Epoch 7: Train err: 0.46325, Train loss: 0.6926203593611717 |Validation err:
0.47, Validation loss: 0.692552462220192
Epoch 8: Train err: 0.46225, Train loss: 0.6925435587763786 |Validation err:
0.463, Validation loss: 0.6924485564231873
Epoch 9: Train err: 0.459625, Train loss: 0.6924680471420288 |Validation er
r: 0.457, Validation loss: 0.6923621594905853
Epoch 10: Train err: 0.458, Train loss: 0.6923965588212013 |Validation err:
0.4555, Validation loss: 0.6922826021909714
Epoch 11: Train err: 0.454875, Train loss: 0.6923230737447739 |Validation er
r: 0.4505, Validation loss: 0.6921818852424622
Epoch 12: Train err: 0.4535, Train loss: 0.6922412514686584 |Validation err:
0.441, Validation loss: 0.6920914500951767
Epoch 13: Train err: 0.450375, Train loss: 0.6921614445745945 |Validation er
r: 0.437, Validation loss: 0.691996842622757
Epoch 14: Train err: 0.44725, Train loss: 0.692103236913681 |Validation err:
0.433, Validation loss: 0.6918932348489761
Epoch 15: Train err: 0.449375, Train loss: 0.6920064575970173 |Validation er
r: 0.432, Validation loss: 0.6917892247438431
Epoch 16: Train err: 0.44425, Train loss: 0.6919283680617809 |Validation er
r: 0.432, Validation loss: 0.6916972249746323
Epoch 17: Train err: 0.441375, Train loss: 0.6918644681572914 |Validation er
r: 0.431, Validation loss: 0.6916135102510452
Epoch 18: Train err: 0.438125, Train loss: 0.6917712353169918 |Validation er
r: 0.4295, Validation loss: 0.6915201544761658
Epoch 19: Train err: 0.436375, Train loss: 0.6917018331587315 |Validation er
r: 0.428, Validation loss: 0.6914086639881134
Epoch 20: Train err: 0.436375, Train loss: 0.6915871091187 |Validation err:
0.4275, Validation loss: 0.6913044154644012
Epoch 21: Train err: 0.437, Train loss: 0.6915052309632301 |Validation err:
0.4285, Validation loss: 0.6911860555410385
Epoch 22: Train err: 0.438625, Train loss: 0.6914149597287178 |Validation er
r: 0.428, Validation loss: 0.6910803616046906
Epoch 23: Train err: 0.436875, Train loss: 0.6912974417209625 |Validation er
r: 0.428, Validation loss: 0.6909734308719635
Epoch 24: Train err: 0.436875, Train loss: 0.6912120580673218 |Validation er
r: 0.425, Validation loss: 0.6908644735813141
Epoch 25: Train err: 0.435125, Train loss: 0.6910865157842636 |Validation er
r: 0.4255, Validation loss: 0.6907256692647934
Epoch 26: Train err: 0.434625, Train loss: 0.6910119131207466 |Validation er
r: 0.4245, Validation loss: 0.6906051337718964
Epoch 27: Train err: 0.43675, Train loss: 0.6909283213317394 |Validation er
r: 0.4265, Validation loss: 0.6904648989439011
Epoch 28: Train err: 0.43575, Train loss: 0.690827514976263 |Validation err:
0.4265, Validation loss: 0.6903412938117981
```

```
Epoch 29: Train err: 0.436375, Train loss: 0.6906765215098858 |Validation er
r: 0.423, Validation loss: 0.6901802867650986
Epoch 30: Train err: 0.435625, Train loss: 0.6905755065381527 |Validation er
r: 0.4235, Validation loss: 0.6900565028190613
Finished Training
Total time elapsed: 148.85 seconds
```





Part (c) - 2pt

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

Answer

I think that decreasing the batch size to 256, while increasing the learning rate to 0.005 and increasing the number of epoch to 30 might result in better generilization. I am hoping that this way, the error/loss will decrease more rapidly to help with generalization.

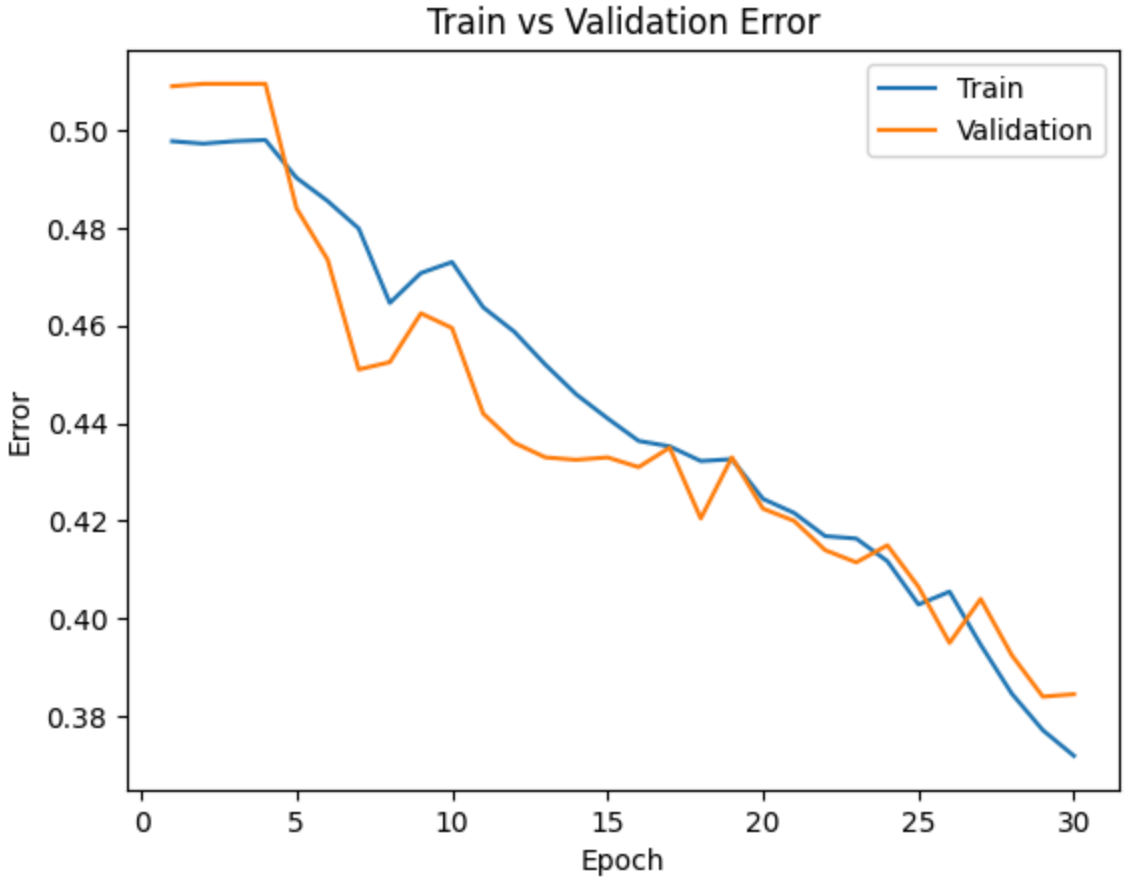
Part (d) - 1pt

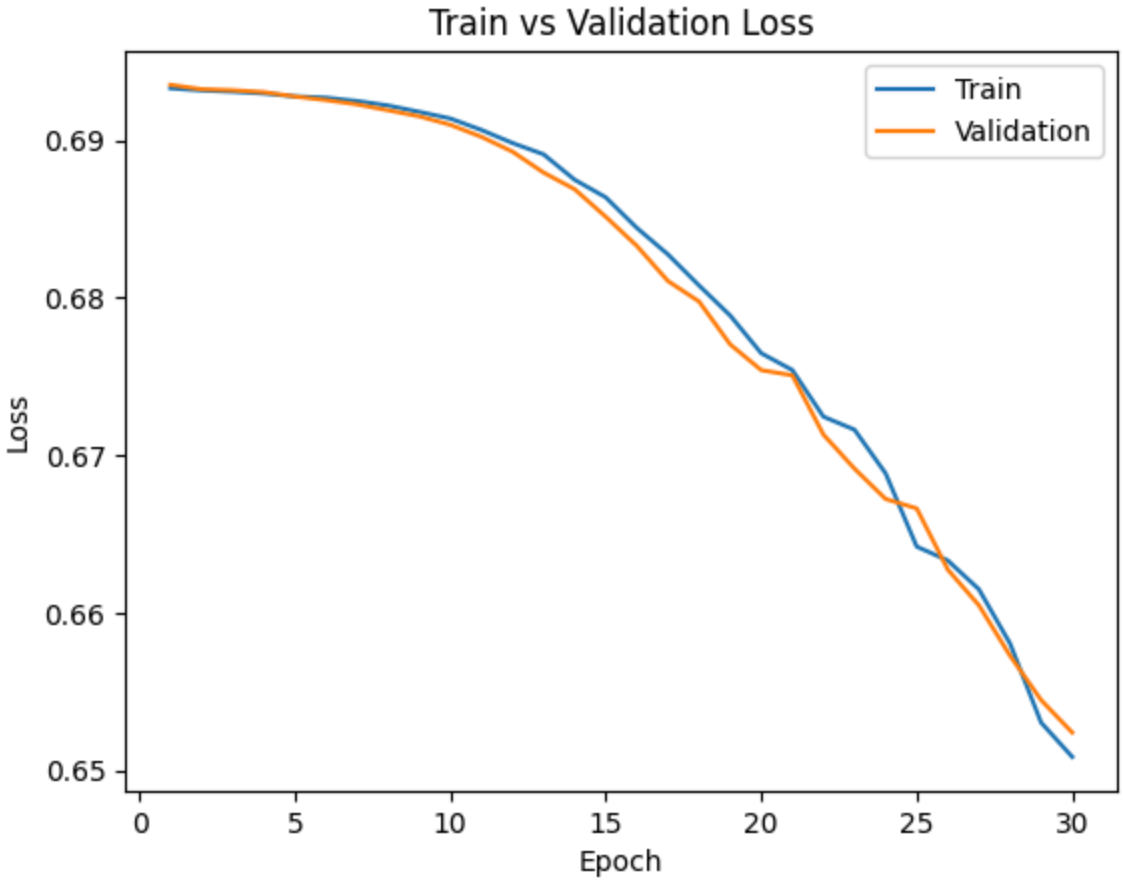
Train the model with the hyperparameters you chose in part(c), and include the training curve.

In [32]:
```
large_net = LargeNet()
train_net(large_net, batch_size=256, learning_rate=0.005)
plot_training_curve(get_model_name("large", batch_size=256, learning_rate=0.
```

```
Epoch 1: Train err: 0.49775, Train loss: 0.6932864934206009 |Validation err:
0.509, Validation loss: 0.6935058310627937
Epoch 2: Train err: 0.49725, Train loss: 0.6931500993669033 |Validation err:
0.5095, Validation loss: 0.6932283341884613
Epoch 3: Train err: 0.49775, Train loss: 0.6930701192468405 |Validation err:
0.5095, Validation loss: 0.6931635290384293
Epoch 4: Train err: 0.498, Train loss: 0.6929577831178904 |Validation err:
0.5095, Validation loss: 0.6930425390601158
Epoch 5: Train err: 0.49025, Train loss: 0.6927728783339262 |Validation err:
0.484, Validation loss: 0.692765511572361
Epoch 6: Train err: 0.4855, Train loss: 0.692691458389163 |Validation err:
0.4735, Validation loss: 0.6925371438264847
Epoch 7: Train err: 0.479875, Train loss: 0.6924769636243582 |Validation er
r: 0.451, Validation loss: 0.692267544567585
Epoch 8: Train err: 0.464625, Train loss: 0.6921934299170971 |Validation er
r: 0.4525, Validation loss: 0.6918911710381508
Epoch 9: Train err: 0.47075, Train loss: 0.6917915064841509 |Validation err:
0.4625, Validation loss: 0.6915095075964928
Epoch 10: Train err: 0.473, Train loss: 0.691368954256177 |Validation err:
0.4595, Validation loss: 0.6909696832299232
Epoch 11: Train err: 0.46375, Train loss: 0.6906428448855877 |Validation er
r: 0.442, Validation loss: 0.690207839012146
Epoch 12: Train err: 0.45875, Train loss: 0.6898134779185057 |Validation er
r: 0.436, Validation loss: 0.6892660111188889
Epoch 13: Train err: 0.452, Train loss: 0.6890941150486469 |Validation err:
0.433, Validation loss: 0.6879362463951111
Epoch 14: Train err: 0.445875, Train loss: 0.687480540946126 |Validation er
r: 0.4325, Validation loss: 0.6868794336915016
Epoch 15: Train err: 0.441, Train loss: 0.6863649897277355 |Validation err:
0.433, Validation loss: 0.6851314529776573
Epoch 16: Train err: 0.436375, Train loss: 0.6844243220984936 |Validation er
r: 0.431, Validation loss: 0.6832825914025307
Epoch 17: Train err: 0.43525, Train loss: 0.6827412955462933 |Validation er
r: 0.435, Validation loss: 0.6810513883829117
Epoch 18: Train err: 0.43225, Train loss: 0.6807704139500856 |Validation er
r: 0.4205, Validation loss: 0.6797515526413918
Epoch 19: Train err: 0.432625, Train loss: 0.6788658425211906 |Validation er
r: 0.433, Validation loss: 0.6770314201712608
Epoch 20: Train err: 0.4245, Train loss: 0.6764682028442621 |Validation err:
0.4225, Validation loss: 0.6753945350646973
Epoch 21: Train err: 0.421625, Train loss: 0.6754005029797554 |Validation er
r: 0.42, Validation loss: 0.675066277384758
Epoch 22: Train err: 0.416875, Train loss: 0.672440342605114 |Validation er
r: 0.414, Validation loss: 0.6713044494390488
Epoch 23: Train err: 0.416375, Train loss: 0.6716131493449211 |Validation er
r: 0.4115, Validation loss: 0.6691384613513947
Epoch 24: Train err: 0.41175, Train loss: 0.6688405908644199 |Validation er
r: 0.415, Validation loss: 0.667217306792736
Epoch 25: Train err: 0.402875, Train loss: 0.66419299505651 |Validation err:
0.4065, Validation loss: 0.66661436855793
Epoch 26: Train err: 0.4055, Train loss: 0.6633104812353849 |Validation err:
0.395, Validation loss: 0.6627303212881088
Epoch 27: Train err: 0.394625, Train loss: 0.6614811085164547 |Validation er
r: 0.404, Validation loss: 0.6604675725102425
Epoch 28: Train err: 0.384625, Train loss: 0.6580445114523172 |Validation er
r: 0.3925, Validation loss: 0.6572717651724815
```

```
Epoch 29: Train err: 0.377125, Train loss: 0.6530223544687033 |Validation er
r: 0.384, Validation loss: 0.6544522047042847
Epoch 30: Train err: 0.371875, Train loss: 0.6508463714271784 |Validation er
r: 0.3845, Validation loss: 0.65239118039608
Finished Training
Total time elapsed: 149.58 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 [26]:
```
net = LargeNet()
model_path = get_model_name(net.name, batch_size=256, learning_rate=0.005, e
state = torch.load(model_path)
net.load_state_dict(state)
```

Out[26]:
```
<All keys matched successfully>
```

Part (b) - 2pt

Justify your choice of model from part (a).

Answer

I decided to choose the parameters for the model I trained in part 3d. In my opinion, the validation curve closely followed the training curve, which means that it has good generalization throughout, didn't overfit since the 2 curves never diverged from one another.

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 [27]:
```
# 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)

test = nn.BCEWithLogitsLoss()
test_err, test_loss = evaluate(net, test_loader, test)
print("Test error: {}".format(test_err))
print("Test loss: {}".format(test_loss))

validation_err, validation_loss = evaluate(net, val_loader, test)
print("Validation error: {}".format(validation_err))
print("Validation loss: {}".format(validation_loss))
```

```
Test error: 0.343
Test loss: 0.621306125074625
Validation error: 0.356
Validation loss: 0.6200363542884588
```

Answer

Test classification error is 0.343

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

Usually, you would expect the test error to be higher than the validation error because the model was based on validation performance during training. I think this would make it slightly biased toward the validation set so the error would be lower, but my test error is actually lower. Maybe this was caused by random differences between the validation and test samples, where the test set happened to contain easier examples to classify?

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

We save the test dataset for the very end to check how well our model works on new, unseen data. If we use the test data too much at the start, it can "learn" from the test data, making the results seem better than they actually are, resulting in overfitting. By using the test data only once at the end, we get an unbiased result of the model's performance.

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.

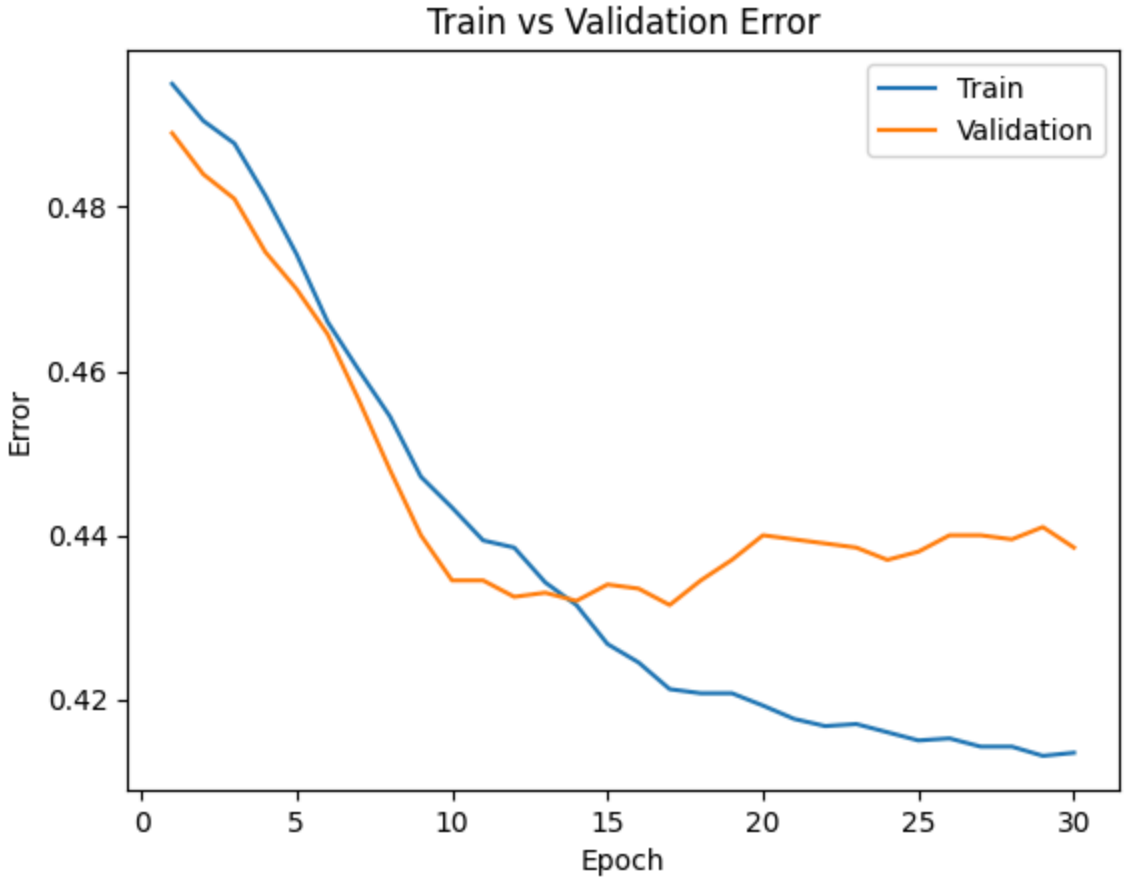
Hint: The ANN in lab 1 was applied on greyscale images. The cat and dog images are colour (RGB) and so you will need to flatted and concatinate all three colour layers before feeding them into an ANN.

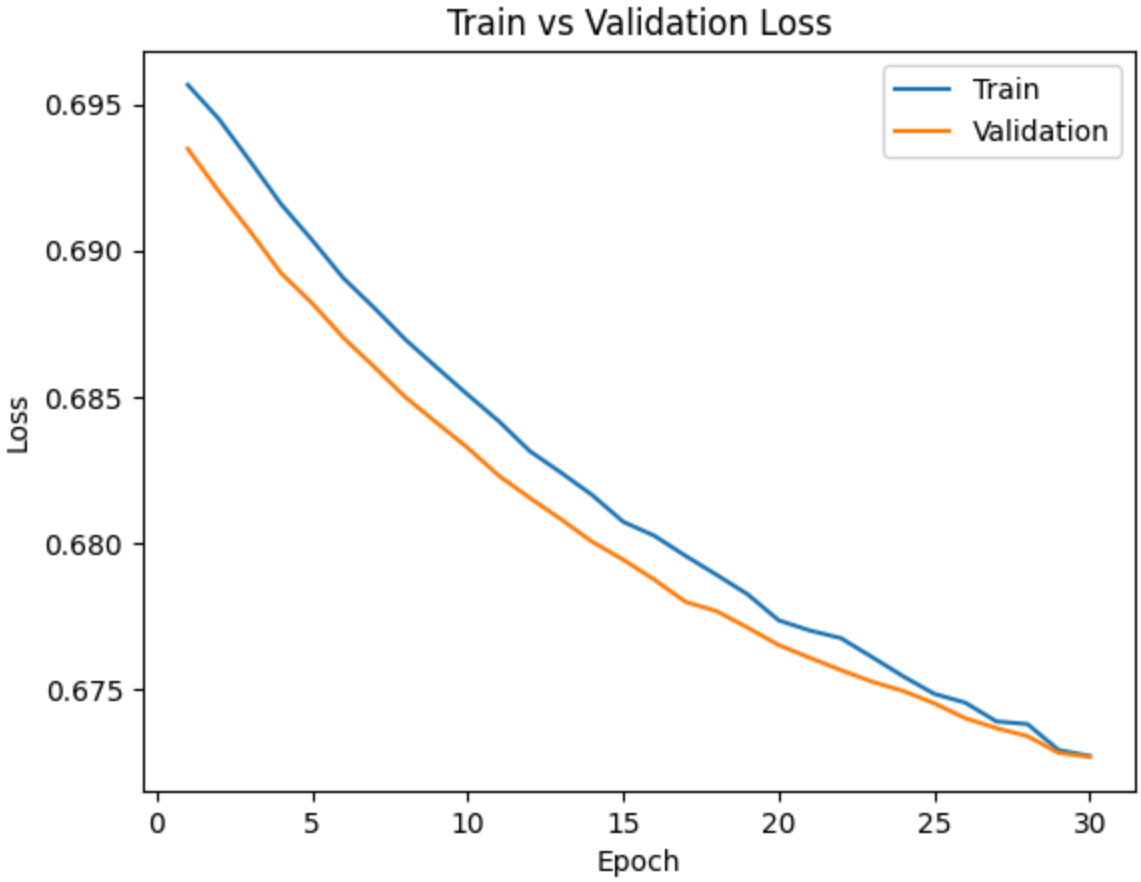
In [35]:
```
#from lab 1

import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, transforms
import matplotlib.pyplot as plt # for plotting
import torch.optim as optim

torch.manual_seed(1) # set the random seed

class Pigeon(nn.Module):
    def __init__(self):
        super(Pigeon, self).__init__()
        self.name = "pigeon"
        self.layer1 = nn.Linear(32 * 32 * 3, 30)
        self.layer2 = nn.Linear(30, 1)

    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()

train_loader, val_loader, test_loader, classes = get_data_loader(
target_classes=["cat", "dog"],
batch_size=512)

#training
train_net(pigeon, batch_size=512, learning_rate=0.0001)
model_path = get_model_name("pigeon", batch_size=512, learning_rate=0.0001,
plot_training_curve(model_path)

# Testing
criterion = nn.BCEWithLogitsLoss()
error, loss = evaluate(pigeon, test_loader, criterion)
print("The test classification error is", error, "and the test loss is", los
```

```
Epoch 1: Train err: 0.495, Train loss: 0.6956628561019897 |Validation err:
0.489, Validation loss: 0.6934726536273956
Epoch 2: Train err: 0.4905, Train loss: 0.6944922097027302 |Validation err:
0.484, Validation loss: 0.692015990614891
Epoch 3: Train err: 0.48775, Train loss: 0.6930558905005455 |Validation err:
0.481, Validation loss: 0.6906565278768539
Epoch 4: Train err: 0.481375, Train loss: 0.6915821395814419 |Validation er
r: 0.4745, Validation loss: 0.6892245709896088
Epoch 5: Train err: 0.47425, Train loss: 0.6903515309095383 |Validation err:
0.47, Validation loss: 0.6881945729255676
Epoch 6: Train err: 0.466, Train loss: 0.6890573538839817 |Validation err:
0.4645, Validation loss: 0.6870233118534088
Epoch 7: Train err: 0.460125, Train loss: 0.6880327202379704 |Validation er
r: 0.4565, Validation loss: 0.686014324426651
Epoch 8: Train err: 0.4545, Train loss: 0.6869523748755455 |Validation err:
0.448, Validation loss: 0.6849831193685532
Epoch 9: Train err: 0.447125, Train loss: 0.6860044673085213 |Validation er
r: 0.44, Validation loss: 0.6841146647930145
Epoch 10: Train err: 0.443375, Train loss: 0.6850729286670685 |Validation er
r: 0.4345, Validation loss: 0.6832539588212967
Epoch 11: Train err: 0.439375, Train loss: 0.6841644942760468 |Validation er
r: 0.4345, Validation loss: 0.6823029667139053
Epoch 12: Train err: 0.4385, Train loss: 0.6831399314105511 |Validation err:
0.4325, Validation loss: 0.681540384888649
Epoch 13: Train err: 0.43425, Train loss: 0.6824091412127018 |Validation er
r: 0.433, Validation loss: 0.6808141022920609
Epoch 14: Train err: 0.4315, Train loss: 0.6816442981362343 |Validation err:
0.432, Validation loss: 0.6800456643104553
Epoch 15: Train err: 0.42675, Train loss: 0.6807287782430649 |Validation er
r: 0.434, Validation loss: 0.6794317960739136
Epoch 16: Train err: 0.4245, Train loss: 0.6802511289715767 |Validation err:
0.4335, Validation loss: 0.6787526458501816
Epoch 17: Train err: 0.42125, Train loss: 0.6795667223632336 |Validation er
r: 0.4315, Validation loss: 0.6779935210943222
Epoch 18: Train err: 0.42075, Train loss: 0.6789167262613773 |Validation er
r: 0.4345, Validation loss: 0.677673265337944
Epoch 19: Train err: 0.42075, Train loss: 0.678250040858984 |Validation err:
0.437, Validation loss: 0.6771111488342285
Epoch 20: Train err: 0.41925, Train loss: 0.6773593351244926 |Validation er
r: 0.44, Validation loss: 0.676514744758606
Epoch 21: Train err: 0.417625, Train loss: 0.6770113930106163 |Validation er
r: 0.4395, Validation loss: 0.6760795265436172
Epoch 22: Train err: 0.41675, Train loss: 0.6767516359686852 |Validation er
r: 0.439, Validation loss: 0.6756591945886612
Epoch 23: Train err: 0.417, Train loss: 0.6761059165000916 |Validation err:
0.4385, Validation loss: 0.6752630323171616
Epoch 24: Train err: 0.416, Train loss: 0.6754485666751862 |Validation err:
0.437, Validation loss: 0.674946665763855
Epoch 25: Train err: 0.415, Train loss: 0.6748479828238487 |Validation err:
0.438, Validation loss: 0.6745251566171646
Epoch 26: Train err: 0.41525, Train loss: 0.6745449155569077 |Validation er
r: 0.44, Validation loss: 0.6740157753229141
Epoch 27: Train err: 0.41425, Train loss: 0.6739039160311222 |Validation er
r: 0.44, Validation loss: 0.6736737340688705
Epoch 28: Train err: 0.41425, Train loss: 0.6738136187195778 |Validation er
r: 0.4395, Validation loss: 0.6734024584293365
```

```
Epoch 29: Train err: 0.413125, Train loss: 0.6729271076619625 |Validation er
r: 0.441, Validation loss: 0.67282934486866
Epoch 30: Train err: 0.4135, Train loss: 0.672724787145853 |Validation err:
0.4385, Validation loss: 0.6726945787668228
Finished Training
Total time elapsed: 120.91 seconds
```






```
The test classification error is 0.409 and the test loss is 0.67120306193828
58
```

From the previous part, the CNN had a test classification error of 0.343

This ANN shows a test classification error of 0.409

therefore, the cnn has lower error

In [37]:
```
%%shell
jupyter nbconvert --to html /content/Lab2CatsvsDogs.ipynb
```

```
[NbConvertApp] Converting notebook /content/Lab2CatsvsDogs.ipynb to html
[NbConvertApp] Writing 366663 bytes to /content/Lab2CatsvsDogs.html
```

Out[37]: