Lab 2: Cats vs Dogs

In this lab, you will train a convolutional neural network to classify an image into one of two classes: "cat" or "dog". The code for the neural networks you train will be written for you, and you are not (yet!) expected to understand all provided code. However, by the end of the lab, you should be able to:

1. Understand at a high level the training loop for a machine learning model.
2. Understand the distinction between training, validation, and test data.
3. The concepts of overfitting and underfitting.
4. Investigate how different hyperparameters, such as learning rate and batch size, affect the success of training.
5. Compare an ANN (aka Multi-Layer Perceptron) with a CNN.

What to submit

Submit a PDF file containing all your code, outputs, and write-up from parts 1-5. You can produce a PDF of your Google Colab file by going to **File > Print** and then save as PDF. The Colab instructions has more information.

Do not submit any other files produced by your code.

Include a link to your colab file in your submission.

Please use Google Colab to complete this assignment. If you want to use Jupyter Notebook, please complete the assignment and upload your Jupyter Notebook file to Google Colab for submission.

With Colab, you can export a PDF file using the menu option `File -> Print` and save as PDF file. **Adjust the scaling to ensure that the text is not cutoff at the margins.**

Colab Link

Include a link to your colab file here

Colab Link: https://colab.research.google.com/drive/1ainIox2xzXZFMkwQxSiXzf5lWaAwfDyO?usp=drive_link

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 classes
                        Should be a subset of the 'classes'
    Returns:
        indices: list of indices that have labels corresponding to one of the
                 target classes
    """
    indices = []
    for i in range(len(dataset)):
        # Check if the label is in the target classes
        label_index = dataset[i][1] # ex: 3
        label_class = classes[label_index] # ex: 'cat'
        if label_class in target_classes:
            indices.append(i)
    return indices

def get_data_loader(target_classes, batch_size):
    """ Loads images of cats and dogs, splits the data into training, validation
    and testing datasets. Returns data loaders for the three preprocessed datasets.

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

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

    classes = ('plane', 'car', 'bird', 'cat',
               'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
    ########################################################################
    # The output of torchvision datasets are PILImage images of range [0, 1].
    # We transform them to Tensors of normalized range [-1, 1].
    transform = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
    # Load CIFAR10 training data
    trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                            download=True, transform=transform)
    # Get the list of indices to sample from
    relevant_indices = get_relevant_indices(trainset, classes, target_classes)

    # Split into train and validation
    np.random.seed(1000) # Fixed numpy random seed for reproducible shuffling
    np.random.shuffle(relevant_indices)
    split = int(len(relevant_indices) * 0.8) #split at 80%

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

###############################################################################
# Training
def get_model_name(name, batch_size, learning_rate, epoch):
    """ Generate a name for the model consisting of all the hyperparameter values

    Args:
        config: Configuration object containing the hyperparameters
    Returns:
        path: A string with the hyperparameter name and value concatenated
    """
    path = "model_{0}_bs{1}_lr{2}_epoch{3}".format(name,
                                                   batch_size,
                                                   learning_rate,
                                                   epoch)
    return path

def normalize_label(labels):
    """
    Given a tensor containing 2 possible values, normalize this to 0/1

    Args:
        labels: a 1D tensor containing two possible scalar values
    Returns:
        A tensor normalize to 0/1 value
    """
    max_val = torch.max(labels)
    min_val = torch.min(labels)
    norm_labels = (labels - min_val)/(max_val - min_val)
    return norm_labels

def evaluate(net, loader, criterion):
    """ Evaluate the network on the validation set.

     Args:
         net: PyTorch neural network object
         loader: PyTorch data loader for the validation set
         criterion: The loss function
     Returns:
         err: A scalar for the avg classification error over the validation set
         loss: A scalar for the average loss function over the validation set
     """
    total_loss = 0.0
    total_err = 0.0
    total_epoch = 0
    for i, data in enumerate(loader, 0):
        inputs, labels = data
        labels = normalize_label(labels)  # Convert labels to 0/1
        outputs = net(inputs)
        loss = criterion(outputs, labels.float())
        corr = (outputs > 0.0).squeeze().long() != labels
        total_err += int(corr.sum())
        total_loss += loss.item()
        total_epoch += len(labels)
    err = float(total_err) / total_epoch
    loss = float(total_loss) / (i + 1)
    return err, loss

###############################################################################
# Training Curve
def plot_training_curve(path):
    """ Plots the training curve for a model run, given the csv files
    containing the train/validation error/loss.

    Args:
        path: The base path of the csv files produced during training
    """
    import matplotlib.pyplot as plt
    train_err = np.loadtxt("{}_train_err.csv".format(path))
    val_err = np.loadtxt("{}_val_err.csv".format(path))
    train_loss = np.loadtxt("{}_train_loss.csv".format(path))
    val_loss = np.loadtxt("{}_val_loss.csv".format(path))
    plt.title("Train vs Validation Error")
    n = len(train_err) # number of epochs
    plt.plot(range(1,n+1), train_err, label="Train")
    plt.plot(range(1,n+1), val_err, label="Validation")
    plt.xlabel("Epoch")
    plt.ylabel("Error")
    plt.legend(loc='best')
    plt.show()
    plt.title("Train vs Validation Loss")
    plt.plot(range(1,n+1), train_loss, label="Train")
    plt.plot(range(1,n+1), val_loss, label="Validation")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.legend(loc='best')
    plt.show()
```

Part 1. Visualizing the Data [7 pt]

We will make use of some of the CIFAR-10 data set, which consists of colour images of size 32x32 pixels belonging to 10 categories. You can find out more about the dataset at https://www.cs.toronto.edu/~kriz/cifar.html

For this assignment, we will only be using the cat and dog categories. We have included code that automatically downloads the dataset the first time that the main script is run.

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
```

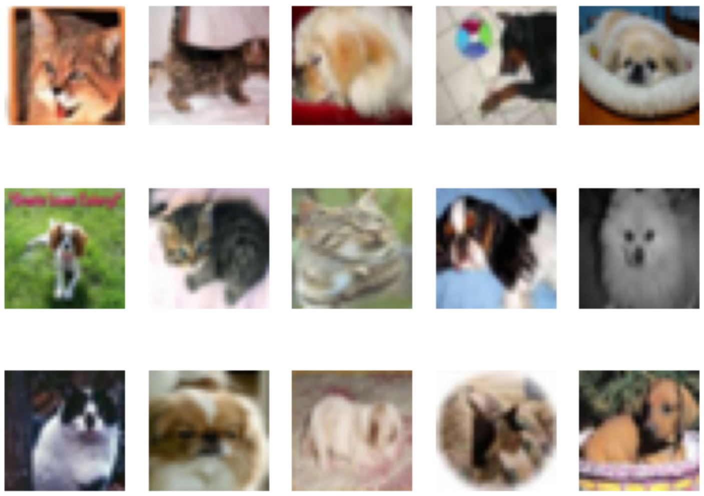
```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:02<00:00, 77616183.33it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Part (a) -- 1 pt

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

(You don't need to submit anything else.)

In [4]:
```python
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]:
```python
sets = [train_loader, test_loader, val_loader]
for i in sets:
  print(len(i))

#From the output below we can see we have 8000 training examples, 2000 testing examples and 2000 validation examples
```

```
8000
2000
2000
```

From the output above we can see we have 8000 training examples, 2000 testing examples and 2000 validation examples in the combined dog and cat classes.

Part (c) -- 3pt

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

Since we get our model to learn the dataset based on the training examples, there is a higher likelihood of the model being overfitted to the data producing low loss/error as compared to when it would be run on examples it hasn't seen before (like those in the validation set). The validation set also provides us an opportunity to explore the true performance capabilities of the model when it is exposed to new data similar to a real world setting and tune the hyperparameters accordingly to increase accuracy.

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]:
```
#Using numel from Lab 1 to find number of elements (parameters) in tensor (network)
total_paramters_small = 0
for param in small_net.parameters():
    print(param.shape)
    parameters_small = param.numel()
    total_paramters_small += parameters_small
print("Total paramters in small_net : ", total_paramters_small)

total_paramters_large = 0
for param in large_net.parameters():
    print(param.shape)
    parameters_large = param.numel()
    total_paramters_large += parameters_large
print("Total paramters in large_net : ", total_paramters_large)
```

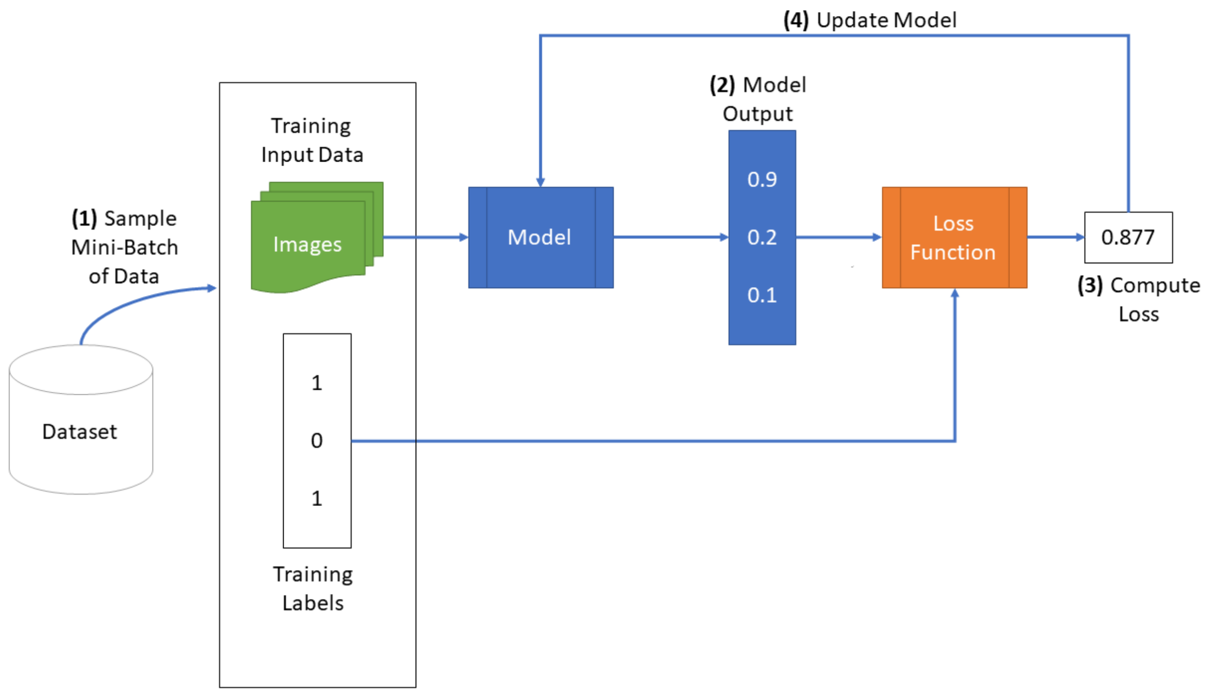
```
torch.Size([5, 3, 3, 3])
torch.Size([5])
torch.Size([1, 245])
torch.Size([1])
Total paramters in small_net :  386
torch.Size([5, 3, 5, 5])
torch.Size([5])
torch.Size([10, 5, 5, 5])
torch.Size([10])
torch.Size([32, 250])
torch.Size([32])
torch.Size([1, 32])
torch.Size([1])
Total paramters in large_net :  9705
```

Total paramters in small_net : 386

Total paramters in large_net : 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 [10]:
```
def train_net(net, batch_size=64, learning_rate=0.01, num_epochs=30):
    ########################################################################
    # Train a classifier on cats vs dogs
    target_classes = ["cat", "dog"]
    ########################################################################
    # Fixed PyTorch random seed for reproducible result
    torch.manual_seed(1000)
    ########################################################################
    # Obtain the PyTorch data loader objects to load batches of the datasets
    train_loader, val_loader, test_loader, classes = get_data_loader(
            target_classes, batch_size)
    ########################################################################
    # Define the Loss function and optimizer
    # The loss function will be Binary Cross Entropy (BCE). In this case we
    # will use the BCEWithLogitsLoss which takes unnormalized output from
    # the neural network and scalar label.
    # Optimizer will be SGD with Momentum.
    criterion = nn.BCEWithLogitsLoss()
    optimizer = optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
    ########################################################################
    # Set up some numpy arrays to store the training/test loss/erruracy
    train_err = np.zeros(num_epochs)
    train_loss = np.zeros(num_epochs)
    val_err = np.zeros(num_epochs)
    val_loss = np.zeros(num_epochs)
    ########################################################################
    # Train the network
    # Loop over the data iterator and sample a new batch of training data
    # Get the output from the network, and optimize our loss function.
    start_time = time.time()
    for epoch in range(num_epochs):  # loop over the dataset multiple times
        total_train_loss = 0.0
        total_train_err = 0.0
        total_epoch = 0
        for i, data in enumerate(train_loader, 0):
            # Get the inputs
            inputs, labels = data
            labels = normalize_label(labels) # Convert labels to 0/1
            # Zero the parameter gradients
            optimizer.zero_grad()
            # Forward pass, backward pass, and optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels.float())
            loss.backward()
            optimizer.step()
            # Calculate the statistics
            corr = (outputs > 0.0).squeeze().long() != labels
            total_train_err += int(corr.sum())
            total_train_loss += loss.item()
            total_epoch += len(labels)
        train_err[epoch] = float(total_train_err) / total_epoch
        train_loss[epoch] = float(total_train_loss) / (i+1)
        val_err[epoch], val_loss[epoch] = evaluate(net, val_loader, criterion)
        print(("Epoch {}: Train err: {}, Train loss: {} |"+
               "Validation err: {}, Validation loss: {}").format(
                   epoch + 1,
                   train_err[epoch],
                   train_loss[epoch],
                   val_err[epoch],
                   val_loss[epoch]))
        # Save the current model (checkpoint) to a file
        model_path = get_model_name(net.name, batch_size, learning_rate, epoch)
        torch.save(net.state_dict(), model_path)
    print('Finished Training')
    end_time = time.time()
    elapsed_time = end_time - start_time
    print("Total time elapsed: {:.2f} seconds".format(elapsed_time))
    # Write the train/test loss/err into CSV file for plotting later
    epochs = np.arange(1, num_epochs + 1)
    np.savetxt("{}_train_err.csv".format(model_path), train_err)
    np.savetxt("{}_train_loss.csv".format(model_path), train_loss)
    np.savetxt("{}_val_err.csv".format(model_path), val_err)
    np.savetxt("{}_val_loss.csv".format(model_path), val_loss)
```

Part (b) -- 1pt

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

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

In [ ]:
```
#The function definition contains all the answers for this question in the line below
#def train_net(net, batch_size=64, learning_rate=0.01, num_epochs=30):

#The default values are, batch_size = 64, learning_rate = 0.01, num_epochs = 30.
```

The default values are, batch_size = 64, learning_rate = 0.01, num_epochs = 30.

Part (c) -- 3 pt

What files are written to disk when we call `train_net` with `small_net`, and train for 5 epochs? Provide a list of all the files written to disk, and what information the files contain.

In [11]:
```
train_net(small_net, 64, 0.01, 5)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.416125, Train loss: 0.6714658460617066 |Validation err: 0.373, Validation loss: 0.6541851703077555
Epoch 2: Train err: 0.36825, Train loss: 0.6417813982963562 |Validation err: 0.3745, Validation loss: 0.6538403276354074
Epoch 3: Train err: 0.34275, Train loss: 0.6244836616516113 |Validation err: 0.3405, Validation loss: 0.6199461929500103
Epoch 4: Train err: 0.329875, Train loss: 0.6090887966156006 |Validation err: 0.3605, Validation loss: 0.6276408210396767
Epoch 5: Train err: 0.317625, Train loss: 0.6000747032165528 |Validation err: 0.322, Validation loss: 0.6144496481865644
Finished Training
Total time elapsed: 22.39 seconds
```

Files written to disk during execution-

model_small_bs64_lr0.01_epoch0 - model saved post 1st epoch

model_small_bs64_lr0.01_epoch1 - model saved post 2nd epoch

model_small_bs64_lr0.01_epoch2 - model saved post 3rd epoch

model_small_bs64_lr0.01_epoch3 - model saved post 4th epoch

model_small_bs64_lr0.01_epoch4 - model saved post 5th epoch

model_small_bs64_lr0.01_epoch4_train_err.csv - training error for all epochs

model_small_bs64_lr0.01_epoch4_train_loss.csv - training loss for all epochs model_small_bs64_lr0.01_epoch4_val_err.csv - validation error for all epochs

model_small_bs64_lr0.01_epoch4_val_loss.csv - validation loss for all epochs

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 [12]:
```
# Since the function writes files to disk, you will need to mount
# your Google Drive. If you are working on the lab locally, you
# can comment out this code.

from google.colab import drive
drive.mount('/content/gdrive')
```

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

In [15]:
```
small_net = SmallNet()
large_net = LargeNet()

#using default parameters to train small_net
train_net(small_net, 64, 0.01, 30)

#using default parameters to train large_net
train_net(large_net, 64, 0.01, 30)

#Training times
#small_net -> Total time elapsed: 147.74 seconds
#large_net -> Total time elapsed: 162.37 seconds

#The large_net network seems to take a little longer than the small_net network as a result of the increased number of training pa
#this section of the lab. large_net parameters = 9705 as compared to small_net parameters = 386.
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.422375, Train loss: 0.6716891074180603 |Validation err: 0.3735, Validation loss: 0.6535317860543728
Epoch 2: Train err: 0.362875, Train loss: 0.6403266997337341 |Validation err: 0.3635, Validation loss: 0.6488472539931536
Epoch 3: Train err: 0.34425, Train loss: 0.6254545783996582 |Validation err: 0.3455, Validation loss: 0.6203189175575972
Epoch 4: Train err: 0.331125, Train loss: 0.6105487518310547 |Validation err: 0.3525, Validation loss: 0.6239588130265474
Epoch 5: Train err: 0.3255, Train loss: 0.6031277022361755 |Validation err: 0.333, Validation loss: 0.6160122845321894
Epoch 6: Train err: 0.3135, Train loss: 0.5945649976730347 |Validation err: 0.3305, Validation loss: 0.6118946485221386
Epoch 7: Train err: 0.316625, Train loss: 0.5917581415176392 |Validation err: 0.3335, Validation loss: 0.6110464800149202
Epoch 8: Train err: 0.309125, Train loss: 0.58613898229599 |Validation err: 0.325, Validation loss: 0.6062229610979557
Epoch 9: Train err: 0.301625, Train loss: 0.5845262587070466 |Validation err: 0.3235, Validation loss: 0.6043402254581451
Epoch 10: Train err: 0.2975, Train loss: 0.5766231815814972 |Validation err: 0.3165, Validation loss: 0.598608729429543
Epoch 11: Train err: 0.299, Train loss: 0.5764271333217621 |Validation err: 0.3155, Validation loss: 0.5942966658622026
Epoch 12: Train err: 0.29175, Train loss: 0.5694889471530914 |Validation err: 0.3215, Validation loss: 0.5985538912937045
Epoch 13: Train err: 0.292125, Train loss: 0.5697676210403443 |Validation err: 0.3145, Validation loss: 0.5932627320289612
Epoch 14: Train err: 0.291625, Train loss: 0.5650441377162934 |Validation err: 0.332, Validation loss: 0.6044320520013571
Epoch 15: Train err: 0.29075, Train loss: 0.56288951587677 |Validation err: 0.31, Validation loss: 0.5882089519873261
Epoch 16: Train err: 0.296625, Train loss: 0.5671735005378723 |Validation err: 0.321, Validation loss: 0.6049329144880176
Epoch 17: Train err: 0.285875, Train loss: 0.5626897213459015 |Validation err: 0.31, Validation loss: 0.5887375781312585
Epoch 18: Train err: 0.289625, Train loss: 0.5576352522373199 |Validation err: 0.3155, Validation loss: 0.5866723582148552
Epoch 19: Train err: 0.285, Train loss: 0.5553905045986176 |Validation err: 0.315, Validation loss: 0.6009101774543524
Epoch 20: Train err: 0.28225, Train loss: 0.5546711716651916 |Validation err: 0.3115, Validation loss: 0.5948053719475865
Epoch 21: Train err: 0.2875, Train loss: 0.5556718516349792 |Validation err: 0.3175, Validation loss: 0.5871756924316287
Epoch 22: Train err: 0.286625, Train loss: 0.5551399085521698 |Validation err: 0.3145, Validation loss: 0.5935489051043987
Epoch 23: Train err: 0.2885, Train loss: 0.5531882934570312 |Validation err: 0.3145, Validation loss: 0.5896968441084027
Epoch 24: Train err: 0.279875, Train loss: 0.5511337373256683 |Validation err: 0.31, Validation loss: 0.5985220922157168
Epoch 25: Train err: 0.282625, Train loss: 0.5493824903964997 |Validation err: 0.3065, Validation loss: 0.5853728735819459
Epoch 26: Train err: 0.279, Train loss: 0.5483519983291626 |Validation err: 0.3105, Validation loss: 0.5852275434881449
Epoch 27: Train err: 0.281, Train loss: 0.5491738488674164 |Validation err: 0.3165, Validation loss: 0.5980016440153122
Epoch 28: Train err: 0.278, Train loss: 0.5491111464500428 |Validation err: 0.316, Validation loss: 0.5854913359507918
Epoch 29: Train err: 0.282875, Train loss: 0.5492837340831757 |Validation err: 0.3115, Validation loss: 0.5953847551718354
Epoch 30: Train err: 0.281125, Train loss: 0.5498382887840271 |Validation err: 0.315, Validation loss: 0.5942998509854078
Finished Training
Total time elapsed: 147.74 seconds
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.449875, Train loss: 0.6902574715614319 |Validation err: 0.4305, Validation loss: 0.6840793807059526
Epoch 2: Train err: 0.4215, Train loss: 0.6773116540908813 |Validation err: 0.45, Validation loss: 0.6902624610811472
Epoch 3: Train err: 0.39075, Train loss: 0.6618121542930603 |Validation err: 0.3755, Validation loss: 0.6502665914595127
Epoch 4: Train err: 0.361625, Train loss: 0.6404139580726623 |Validation err: 0.3695, Validation loss: 0.6479013208299875
Epoch 5: Train err: 0.35, Train loss: 0.6295771174430848 |Validation err: 0.3505, Validation loss: 0.6315131261944771
Epoch 6: Train err: 0.334875, Train loss: 0.6146058330535888 |Validation err: 0.3395, Validation loss: 0.6242715623229742
Epoch 7: Train err: 0.330375, Train loss: 0.6070960855484009 |Validation err: 0.345, Validation loss: 0.6143617928028107
Epoch 8: Train err: 0.323375, Train loss: 0.5923470008373226 |Validation err: 0.336, Validation loss: 0.611571753397584
Epoch 9: Train err: 0.319375, Train loss: 0.5912202508449554 |Validation err: 0.3185, Validation loss: 0.6016443874686956
Epoch 10: Train err: 0.299125, Train loss: 0.5732340822219849 |Validation err: 0.316, Validation loss: 0.597239569760859
Epoch 11: Train err: 0.290875, Train loss: 0.566786470413208 |Validation err: 0.316, Validation loss: 0.5986843183636665
Epoch 12: Train err: 0.282375, Train loss: 0.5541278166770935 |Validation err: 0.305, Validation loss: 0.5887974593788385
Epoch 13: Train err: 0.273125, Train loss: 0.545344172000885 |Validation err: 0.3, Validation loss: 0.5794570930302143
Epoch 14: Train err: 0.269875, Train loss: 0.5330307495594024 |Validation err: 0.294, Validation loss: 0.5820369338616729
Epoch 15: Train err: 0.2625, Train loss: 0.5268569860458374 |Validation err: 0.3015, Validation loss: 0.5762819591909647
Epoch 16: Train err: 0.26525, Train loss: 0.5270128128528595 |Validation err: 0.3025, Validation loss: 0.5798417991027236
Epoch 17: Train err: 0.2565, Train loss: 0.511244300365448 |Validation err: 0.288, Validation loss: 0.5834593530744314
Epoch 18: Train err: 0.240125, Train loss: 0.4939929814338684 |Validation err: 0.289, Validation loss: 0.565758460201323
Epoch 19: Train err: 0.24, Train loss: 0.4928543953895569 |Validation err: 0.282, Validation loss: 0.5820385720580816
Epoch 20: Train err: 0.237, Train loss: 0.482390189409256 |Validation err: 0.299, Validation loss: 0.6357136806473136
Epoch 21: Train err: 0.23475, Train loss: 0.4752623991966248 |Validation err: 0.2765, Validation loss: 0.5641676662489772
Epoch 22: Train err: 0.22425, Train loss: 0.4657190887928009 |Validation err: 0.2865, Validation loss: 0.583379671908915
Epoch 23: Train err: 0.215875, Train loss: 0.45512766695022583 |Validation err: 0.281, Validation loss: 0.5743518304079771
Epoch 24: Train err: 0.212125, Train loss: 0.4448171682357788 |Validation err: 0.296, Validation loss: 0.5907860016450286
Epoch 25: Train err: 0.2035, Train loss: 0.43331602501869204 |Validation err: 0.2905, Validation loss: 0.5954966181889176
Epoch 26: Train err: 0.20325, Train loss: 0.42598249769210816 |Validation err: 0.2885, Validation loss: 0.6136195780709386
Epoch 27: Train err: 0.193375, Train loss: 0.4185446243286133 |Validation err: 0.28, Validation loss: 0.6154750669375062
Epoch 28: Train err: 0.18325, Train loss: 0.4065200546979904 |Validation err: 0.299, Validation loss: 0.6189063107594848
Epoch 29: Train err: 0.18225, Train loss: 0.3986688561439514 |Validation err: 0.2965, Validation loss: 0.6553325271233916
Epoch 30: Train err: 0.174125, Train loss: 0.3825919407606125 |Validation err: 0.299, Validation loss: 0.6410978380590677
Finished Training
Total time elapsed: 162.37 seconds
```

small_net -> Total time elapsed: 147.74 seconds

large_net -> Total time elapsed: 162.37 seconds

The large_net network seems to take a little longer than the small_net network as a result of the increased number of training parameters as found in part (a) of this section of the lab. large_net parameters = 9705 as compared to small_net parameters = 386.

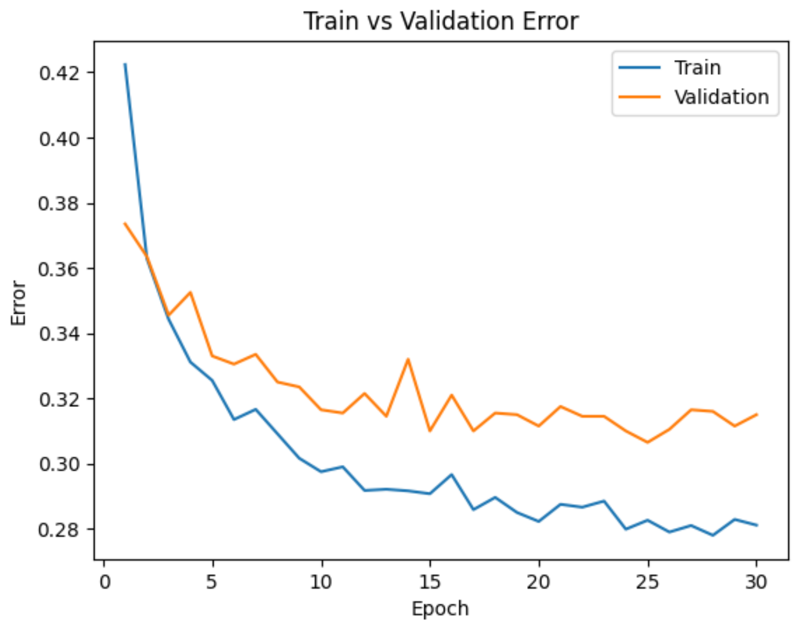
Part (e) - 2pt

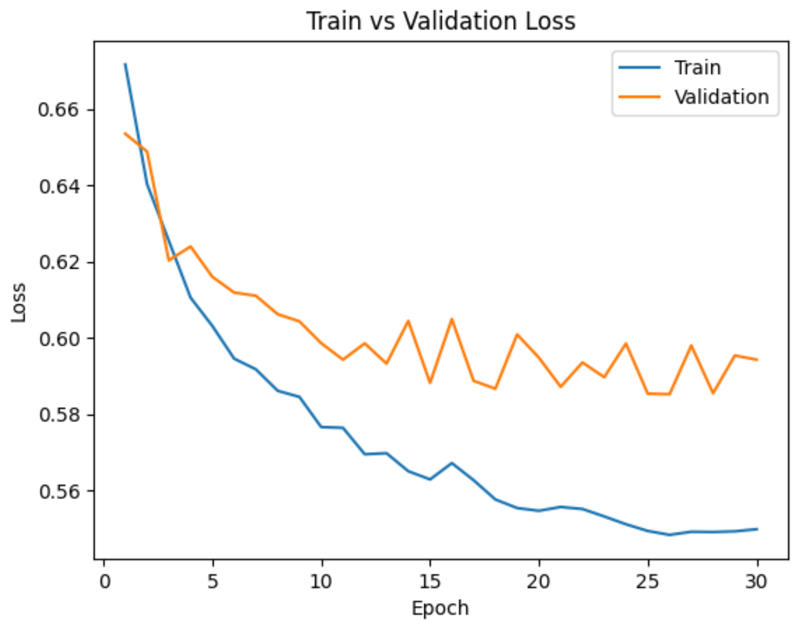
Use the function `plot_training_curve` to display the trajectory of the training/validation error and the training/validation loss. You will need to use the function `get_model_name` to generate the argument to the `plot_training_curve` function.

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

Small_Net Curve Plots -

```
In [16]: #Trajectory of training/validation error and training/validation loss in small_net
model_path = get_model_name("small", batch_size=64, learning_rate=0.01, epoch=29)
print(plot_training_curve(model_path))
```

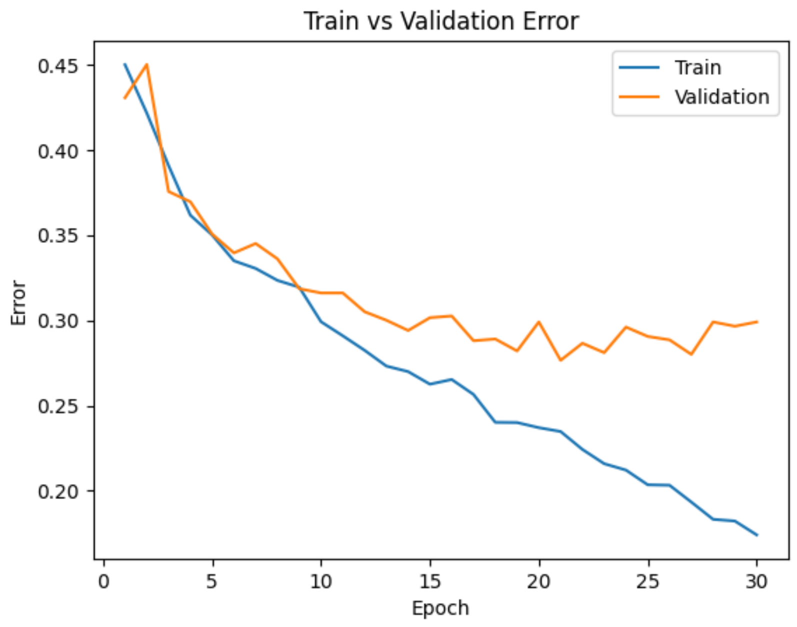


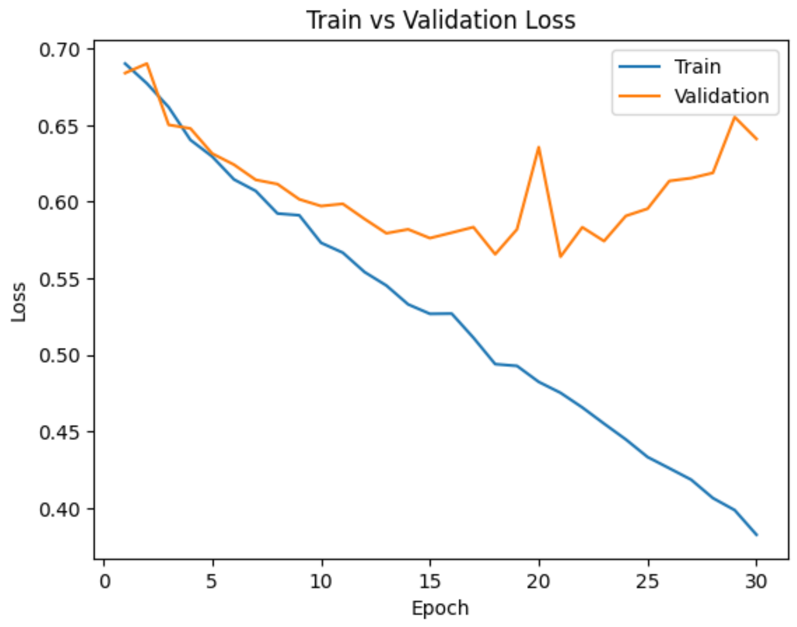


```
None
```

Large_Net Curve Plots -

```
In [17]: #Trajectory of training/validation error and training/validation loss in large_net
model_path = get_model_name("large", batch_size=64, learning_rate=0.01, epoch=29)
print(plot_training_curve(model_path))
```





```
None
```

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.

Mentioned below are a set of differences observed between the training and validation curves of the two models -

The training error in the case of small_net is initially larger than that of large_net for the first 5-7 epochs, but this error reduces a lot quicker until both small and large net produce equivalent error at 8-10 epochs. After this initial rapid error lowering rate, it then decreases a lot slower than in the case of small_net (only going from 0.2975 to 0.28115 between epochs 10 and 30), as compared to large_net which decreases uniformly post the 10th epoch all the way down to 0.174125 in the 30th epoch.

It can also be noted that the validation error curves for both models follow a similar pattern with a slight offset and end up at a difference of 0.02 post the 30th epoch, with large_net prodcuing a lower validation error.

There is a case of underfitting in small_net as there is a decrease in both the training and validation error and loss curves until close to the 15th epoch, after which the loss values fluctuate but don't actually reduce that much, seeming to be held within constant bounds. This fluctuation can also be attributed to underfitting but is a lot less obvious than before the 15th epoch.

As for large_net, the training and validation error and loss seem to curve downwards until the 20th epoch where the validation loss seems to suddenly jump up and start increasing again while the validation error seems to stabalize. Up until the 20th epoch, this is once again a case of underfitting similar to the small_net case. However, post the 20th epoch this is a severe case of overfitting as the training error and loss keep decreasing further but the validation loss jumps back up, and the validation error also starts to be constant post that point.

Part 3. Optimization Parameters [12 pt]

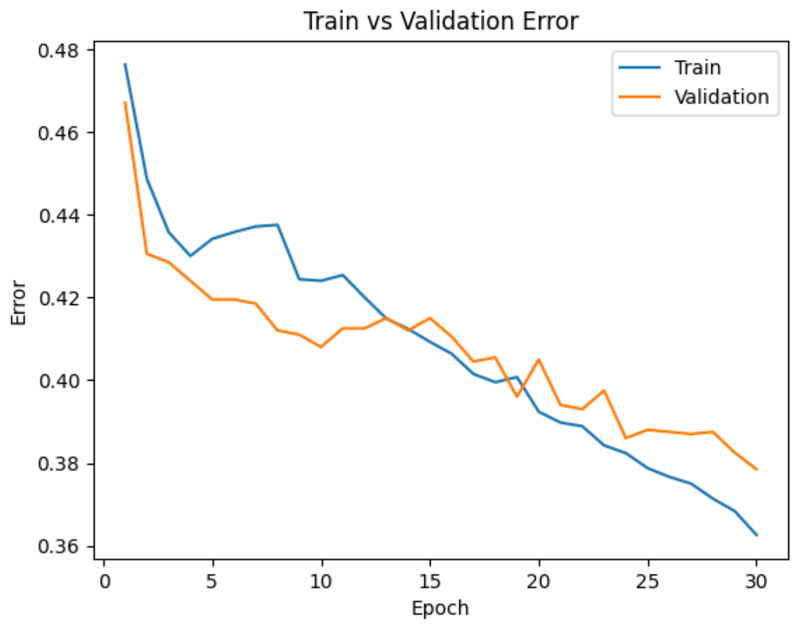
For this section, we will work with `large_net` only.

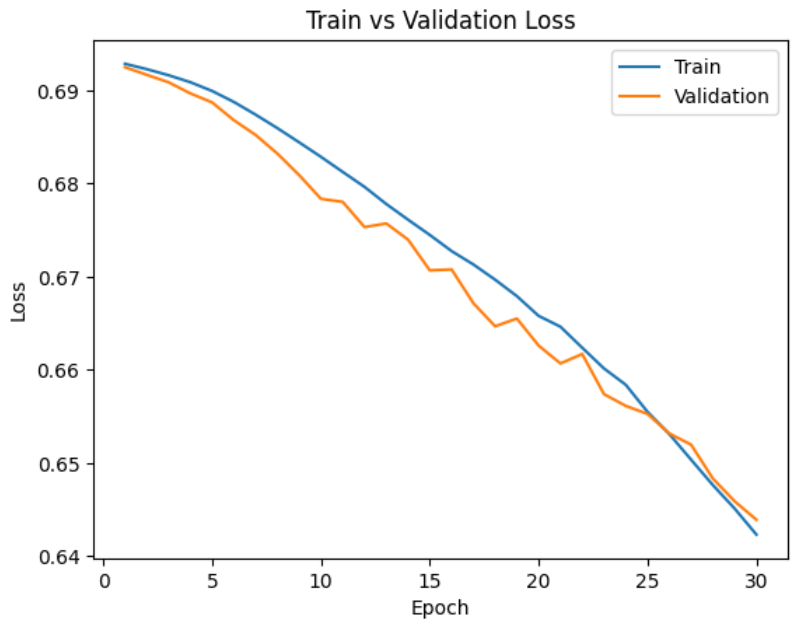
Part (a) - 3pt

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, 64, 0.001, 30)
model_path = get_model_name("large", batch_size=64, learning_rate=0.001, epoch=29)
print(plot_training_curve(model_path))
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.47625, Train loss: 0.6928360013961792 |Validation err: 0.467, Validation loss: 0.6924686580896378
Epoch 2: Train err: 0.448625, Train loss: 0.6922589712142945 |Validation err: 0.4305, Validation loss: 0.691649341955781
Epoch 3: Train err: 0.43575, Train loss: 0.6916067280769348 |Validation err: 0.4285, Validation loss: 0.690854424610734
Epoch 4: Train err: 0.43, Train loss: 0.690861343383789 |Validation err: 0.424, Validation loss: 0.6896595880389214
Epoch 5: Train err: 0.434125, Train loss: 0.6899195008277893 |Validation err: 0.4195, Validation loss: 0.6886935643851757
Epoch 6: Train err: 0.43575, Train loss: 0.6887411961555481 |Validation err: 0.4195, Validation loss: 0.6867824867367744
Epoch 7: Train err: 0.437125, Train loss: 0.6873774147033691 |Validation err: 0.4185, Validation loss: 0.6851982977241278
Epoch 8: Train err: 0.4375, Train loss: 0.6859278454780579 |Validation err: 0.412, Validation loss: 0.683199780061841
Epoch 9: Train err: 0.424375, Train loss: 0.6844058036804199 |Validation err: 0.411, Validation loss: 0.6808880660682917
Epoch 10: Train err: 0.424, Train loss: 0.6828502931594849 |Validation err: 0.408, Validation loss: 0.6783502567559481
Epoch 11: Train err: 0.425375, Train loss: 0.6812348766326904 |Validation err: 0.4125, Validation loss: 0.6780214440077543
Epoch 12: Train err: 0.42, Train loss: 0.6796319708824158 |Validation err: 0.4125, Validation loss: 0.6753159202635288
Epoch 13: Train err: 0.414875, Train loss: 0.6777918744087219 |Validation err: 0.415, Validation loss: 0.6757059413939714
Epoch 14: Train err: 0.412375, Train loss: 0.6761112003326416 |Validation err: 0.412, Validation loss: 0.6739734839648008
Epoch 15: Train err: 0.40925, Train loss: 0.674472680568695 |Validation err: 0.415, Validation loss: 0.6706762500107288
Epoch 16: Train err: 0.406375, Train loss: 0.6727448840141297 |Validation err: 0.4105, Validation loss: 0.6707733049988747
Epoch 17: Train err: 0.4015, Train loss: 0.6713076601028443 |Validation err: 0.4045, Validation loss: 0.6671545393764973
Epoch 18: Train err: 0.3995, Train loss: 0.6696742882728577 |Validation err: 0.4055, Validation loss: 0.6646782550960779
Epoch 19: Train err: 0.40075, Train loss: 0.6679086356163025 |Validation err: 0.396, Validation loss: 0.6655019577592611
Epoch 20: Train err: 0.392375, Train loss: 0.665787980556488 |Validation err: 0.405, Validation loss: 0.6626011095941067
Epoch 21: Train err: 0.38975, Train loss: 0.6646300601959229 |Validation err: 0.394, Validation loss: 0.660687854513526
Epoch 22: Train err: 0.388875, Train loss: 0.6623730587959229 |Validation err: 0.393, Validation loss: 0.6616998575627804
Epoch 23: Train err: 0.38425, Train loss: 0.6601516346931458 |Validation err: 0.3975, Validation loss: 0.6573981791734695
Epoch 24: Train err: 0.382375, Train loss: 0.6584009389877319 |Validation err: 0.386, Validation loss: 0.6561364810913801
Epoch 25: Train err: 0.37875, Train loss: 0.6554971766471863 |Validation err: 0.388, Validation loss: 0.6552744228392839
Epoch 26: Train err: 0.376625, Train loss: 0.6531173253059387 |Validation err: 0.3875, Validation loss: 0.6531743723899126
Epoch 27: Train err: 0.375, Train loss: 0.6503696331977844 |Validation err: 0.387, Validation loss: 0.6519789285957813
Epoch 28: Train err: 0.371375, Train loss: 0.6476435809135437 |Validation err: 0.3875, Validation loss: 0.6483502741903067
Epoch 29: Train err: 0.368375, Train loss: 0.6451257643699646 |Validation err: 0.3825, Validation loss: 0.6459067314863205
Epoch 30: Train err: 0.362625, Train loss: 0.6423329524993896 |Validation err: 0.3785, Validation loss: 0.6439237017184496
Finished Training
Total time elapsed: 165.11 seconds
```





```
None
```

Lowering the learning rate to 0.001 from 0.01 causes the model to take 3 seconds longer (165 sec vs 162 sec) as compared to the default case.

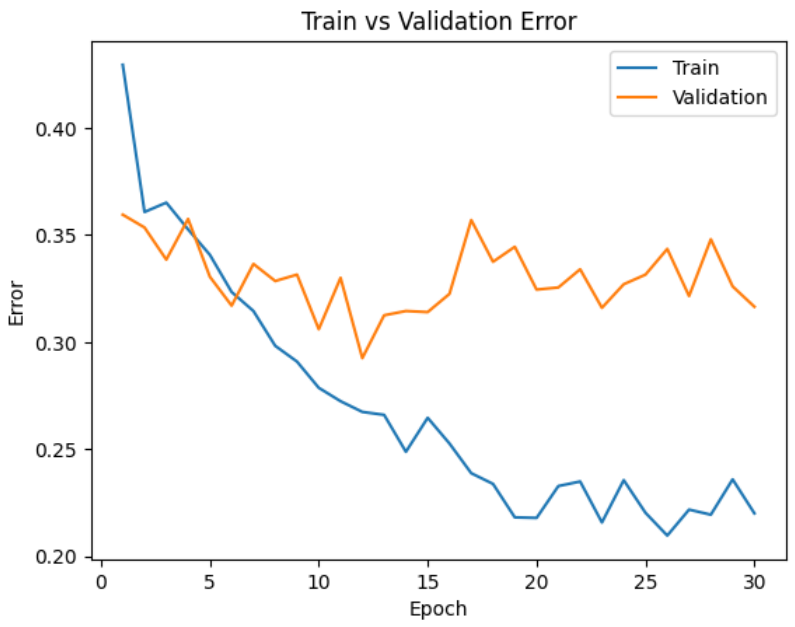
The main difference from the default case is that there is no more overfitting all through the 30 epochs as both the training and validation error and loss curves follow a simialr downward curve. However, it is important to note that both the training error and validation errors are now significantly higher than the default case; increasing from 0.17 to 0.36 in the case of training, and 0.30 to 0.38 in the case of validation after all 30 epochs. The losses are also higher for training comapred to the default case (0.38 vs 0.64), but validation loss remains the same after 30 epochs. All this suggests that 0.001 maybe to low of a learning rate.

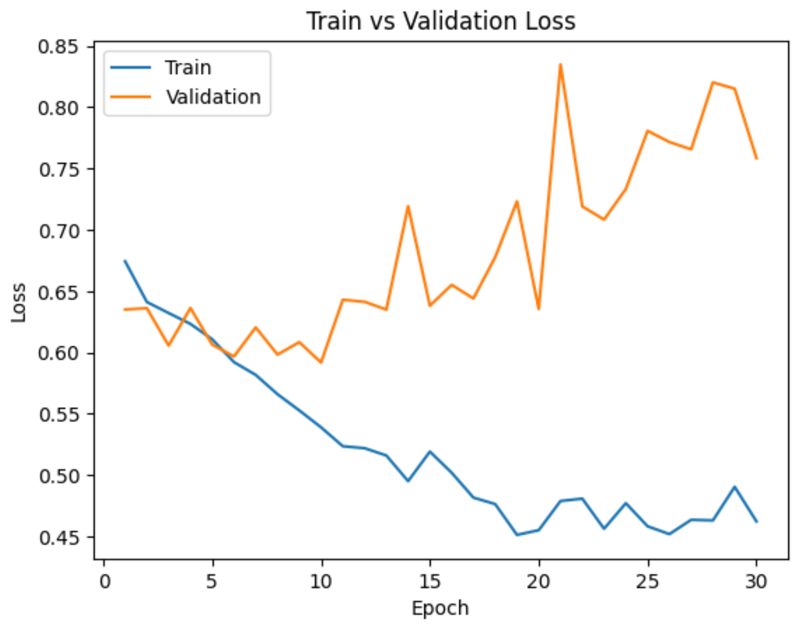
Part (b) - 3pt

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, 64, 0.1, 30)
model_path = get_model_name("large", batch_size=64, learning_rate=0.1, epoch=29)
print(plot_training_curve(model_path))
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.4295, Train loss: 0.67437779712677 |Validation err: 0.3595, Validation loss: 0.6350857093930244
Epoch 2: Train err: 0.36075, Train loss: 0.6411805458068848 |Validation err: 0.3535, Validation loss: 0.6361209936439991
Epoch 3: Train err: 0.365125, Train loss: 0.6321813461780548 |Validation err: 0.3385, Validation loss: 0.6056603882461786
Epoch 4: Train err: 0.352625, Train loss: 0.6233456182479858 |Validation err: 0.3575, Validation loss: 0.6362800188362598
Epoch 5: Train err: 0.34075, Train loss: 0.6108013873100281 |Validation err: 0.3305, Validation loss: 0.6064918786287308
Epoch 6: Train err: 0.323375, Train loss: 0.5921835997104645 |Validation err: 0.317, Validation loss: 0.5967769594863057
Epoch 7: Train err: 0.3145, Train loss: 0.5817317583560944 |Validation err: 0.3365, Validation loss: 0.6204487886279821
Epoch 8: Train err: 0.29825, Train loss: 0.5660300073623658 |Validation err: 0.3285, Validation loss: 0.5983372200280428
Epoch 9: Train err: 0.290875, Train loss: 0.552809501171112 |Validation err: 0.3315, Validation loss: 0.6084455158561468
Epoch 10: Train err: 0.278625, Train loss: 0.539032607793808 |Validation err: 0.306, Validation loss: 0.5918631898239255
Epoch 11: Train err: 0.272375, Train loss: 0.5236025826931 |Validation err: 0.33, Validation loss: 0.6430060230195522
Epoch 12: Train err: 0.267375, Train loss: 0.5220149435997009 |Validation err: 0.2925, Validation loss: 0.6413561534136534
Epoch 13: Train err: 0.266, Train loss: 0.5160510110855102 |Validation err: 0.3125, Validation loss: 0.6349832843989134
Epoch 14: Train err: 0.24875, Train loss: 0.4951590054035187 |Validation err: 0.3145, Validation loss: 0.7193072671070695
Epoch 15: Train err: 0.264625, Train loss: 0.519231944322586 |Validation err: 0.314, Validation loss: 0.6381420725956559
Epoch 16: Train err: 0.252625, Train loss: 0.5020012385845184 |Validation err: 0.3225, Validation loss: 0.6551959458738565
Epoch 17: Train err: 0.23875, Train loss: 0.481714787364006 |Validation err: 0.357, Validation loss: 0.6440742611885071
Epoch 18: Train err: 0.23375, Train loss: 0.47645506453514097 |Validation err: 0.3375, Validation loss: 0.6777342790737748
Epoch 19: Train err: 0.218125, Train loss: 0.45134368968009947 |Validation err: 0.3445, Validation loss: 0.7232250478118658
Epoch 20: Train err: 0.217875, Train loss: 0.45516350817680357 |Validation err: 0.3245, Validation loss: 0.6354950983077288
Epoch 21: Train err: 0.23275, Train loss: 0.47897080445289614 |Validation err: 0.3255, Validation loss: 0.8348110988736153
Epoch 22: Train err: 0.234875, Train loss: 0.4808810565471649 |Validation err: 0.334, Validation loss: 0.7191346418112516
Epoch 23: Train err: 0.21575, Train loss: 0.4563647754192352 |Validation err: 0.316, Validation loss: 0.7083508176729083
Epoch 24: Train err: 0.2355, Train loss: 0.47718250966072084 |Validation err: 0.327, Validation loss: 0.7333047650754452
Epoch 25: Train err: 0.22025, Train loss: 0.4583414270877838 |Validation err: 0.3315, Validation loss: 0.7806987538933754
Epoch 26: Train err: 0.209625, Train loss: 0.4519626965522766 |Validation err: 0.3435, Validation loss: 0.7715998776257038
Epoch 27: Train err: 0.22175, Train loss: 0.4636160457134247 |Validation err: 0.3215, Validation loss: 0.7656293725594878
Epoch 28: Train err: 0.219375, Train loss: 0.46314777398109436 |Validation err: 0.348, Validation loss: 0.8202023077756166
Epoch 29: Train err: 0.235875, Train loss: 0.49053542733192446 |Validation err: 0.326, Validation loss: 0.8150460105389357
Epoch 30: Train err: 0.22, Train loss: 0.4623157248497009 |Validation err: 0.3165, Validation loss: 0.7585078496485949
Finished Training
Total time elapsed: 165.51 seconds
```





```
None
```

Increasing the learning rate to 0.01 from 0.1 causes the model to take 3 seconds longer (165 sec vs 162 sec) as compared to the default case.

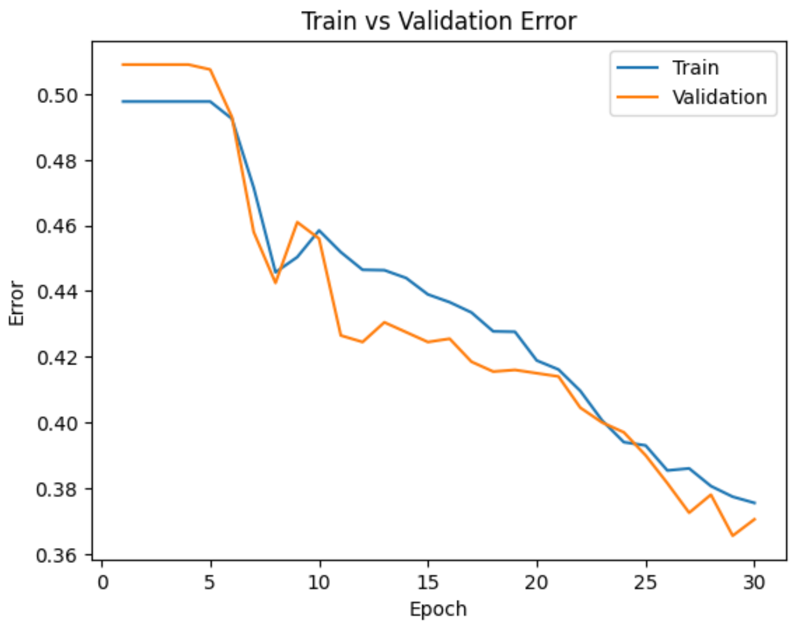
The main difference from the default case is that there is now more overfitting, starting as early as the 10th epoch as both the training error and loss curves follow a simialr downward curve but the validation error seems to stop decreasing and only fluctuate within bounds, and the validation loss starts going higher post the 10th epoch. It is also important to note that both the training error and validation errors are now slightly higher than the default case; increasing from 0.17 to 0.22 in the case of training, and 0.30 to 0.32 in the case of validation after all 30 epochs. The losses are also higher for training comapred to the default case (0.46 vs 0.64), and the validation loss is extremely high compared to default after 30 epochs, increasing from 0.64 to 0.76. All this suggests that 0.1 maybe too high of a learning rate.

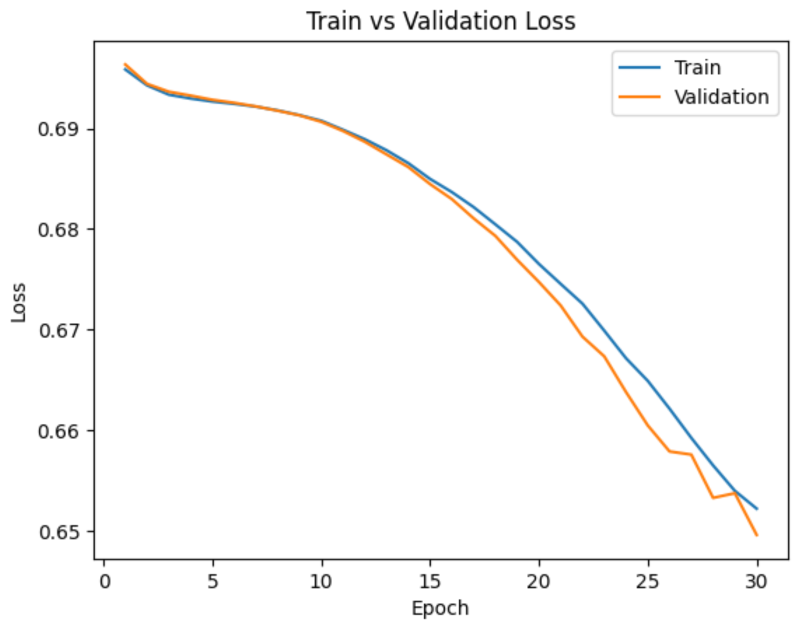
Part (c) - 3pt

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 [24]:
```
large_net = LargeNet()
train_net(large_net, 512, 0.01, 30)
model_path = get_model_name("large", batch_size=512, learning_rate=0.01, epoch=29)
print(plot_training_curve(model_path))
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.49775, Train loss: 0.6958557292819023 |Validation err: 0.509, Validation loss: 0.6963492184877396
Epoch 2: Train err: 0.49775, Train loss: 0.694276686757803 |Validation err: 0.509, Validation loss: 0.6944219917058945
Epoch 3: Train err: 0.49775, Train loss: 0.6933476105332375 |Validation err: 0.509, Validation loss: 0.6936414539813995
Epoch 4: Train err: 0.49775, Train loss: 0.6929657645523548 |Validation err: 0.509, Validation loss: 0.6932569891214371
Epoch 5: Train err: 0.49775, Train loss: 0.6926678568124771 |Validation err: 0.5075, Validation loss: 0.6928388327360153
Epoch 6: Train err: 0.4925, Train loss: 0.6924372650682926 |Validation err: 0.493, Validation loss: 0.6925397664308548
Epoch 7: Train err: 0.4715, Train loss: 0.6921608597040176 |Validation err: 0.458, Validation loss: 0.6921813935041428
Epoch 8: Train err: 0.44575, Train loss: 0.6917873173952103 |Validation err: 0.4425, Validation loss: 0.691753700375557
Epoch 9: Train err: 0.450375, Train loss: 0.6913183219730854 |Validation err: 0.461, Validation loss: 0.6912958174943924
Epoch 10: Train err: 0.4585, Train loss: 0.6907608844339848 |Validation err: 0.456, Validation loss: 0.6906521767377853
Epoch 11: Train err: 0.451875, Train loss: 0.6898587495088577 |Validation err: 0.4265, Validation loss: 0.6897406578063965
Epoch 12: Train err: 0.4465, Train loss: 0.6889194026589394 |Validation err: 0.4245, Validation loss: 0.6886679083108902
Epoch 13: Train err: 0.446375, Train loss: 0.6878230422735214 |Validation err: 0.4305, Validation loss: 0.6874092817306519
Epoch 14: Train err: 0.444, Train loss: 0.6865388303995132 |Validation err: 0.4275, Validation loss: 0.686152458190918
Epoch 15: Train err: 0.439, Train loss: 0.6849737018346786 |Validation err: 0.4245, Validation loss: 0.6844606250524521
Epoch 16: Train err: 0.436625, Train loss: 0.6836653836071491 |Validation err: 0.4255, Validation loss: 0.6829700917005539
Epoch 17: Train err: 0.4335, Train loss: 0.6821726821362972 |Validation err: 0.4185, Validation loss: 0.6810538470745087
Epoch 18: Train err: 0.42775, Train loss: 0.6804557368159294 |Validation err: 0.4155, Validation loss: 0.6793098151683807
Epoch 19: Train err: 0.427625, Train loss: 0.6787110194563866 |Validation err: 0.416, Validation loss: 0.6769335269927979
Epoch 20: Train err: 0.418875, Train loss: 0.6765205599367619 |Validation err: 0.415, Validation loss: 0.6747279018163681
Epoch 21: Train err: 0.416125, Train loss: 0.6745374836027622 |Validation err: 0.414, Validation loss: 0.6723902970552444
Epoch 22: Train err: 0.409625, Train loss: 0.672575306147337 |Validation err: 0.4045, Validation loss: 0.6692968308925629
Epoch 23: Train err: 0.400625, Train loss: 0.6698938198387623 |Validation err: 0.4, Validation loss: 0.6673406064510345
Epoch 24: Train err: 0.394, Train loss: 0.6671492792665958 |Validation err: 0.397, Validation loss: 0.6637893319129944
Epoch 25: Train err: 0.393, Train loss: 0.664900541305542 |Validation err: 0.39, Validation loss: 0.6604697555303574
Epoch 26: Train err: 0.385375, Train loss: 0.6621338650584221 |Validation err: 0.3815, Validation loss: 0.6578838229179382
Epoch 27: Train err: 0.386, Train loss: 0.6592315025627613 |Validation err: 0.3725, Validation loss: 0.6575706899166107
Epoch 28: Train err: 0.380625, Train loss: 0.6564991027116776 |Validation err: 0.378, Validation loss: 0.6532704085111618
Epoch 29: Train err: 0.377375, Train loss: 0.6539665870368481 |Validation err: 0.3655, Validation loss: 0.6537327319383621
Epoch 30: Train err: 0.3755, Train loss: 0.6522011868655682 |Validation err: 0.3705, Validation loss: 0.649590864777565
Finished Training
Total time elapsed: 148.34 seconds
```





```
None
```

Increasing the batch size from 64 to 512 causes the model to take 14 seconds less (148 sec vs 162 sec) as compared to the default case, this might be becuase of the reduced number of iterations required to go through in each epoch.

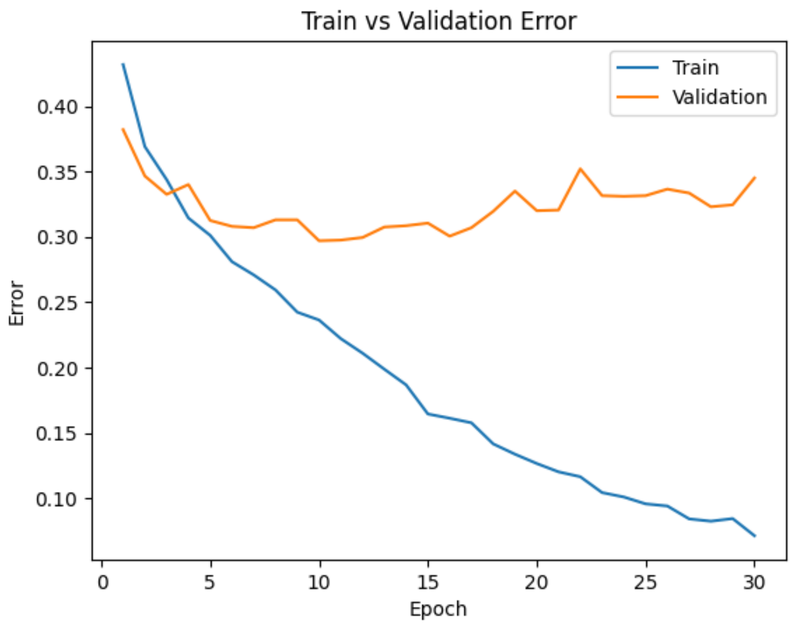
The main difference from the default case is that there is no more overfitting all through the 30 epochs as both the training and validation error and loss curves follow a simialr downward curve. However, it is important to note that both the training error and validation errors are now significantly higher; increasing from 0.17 to 0.37 in the case of training, and 0.30 to 0.37 in the case of validation after all 30 epochs. The losses are also higher for training comapred to the default case (0.38 vs 0.65), but validation loss remains almost the same (even a small bit lower infact) after 30 epochs.

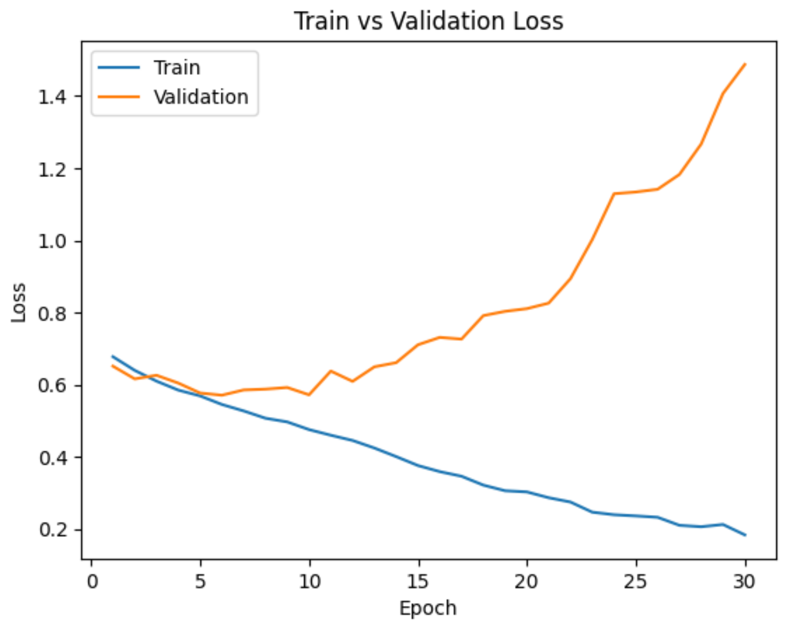
Part (d) - 3pt

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, 16, 0.01, 30)
model_path = get_model_name("large", batch_size=16, learning_rate=0.01, epoch=29)
print(plot_training_curve(model_path))
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.43175, Train loss: 0.6774994022846222 |Validation err: 0.382, Validation loss: 0.6513170118331909
Epoch 2: Train err: 0.369, Train loss: 0.639639899969101 |Validation err: 0.3465, Validation loss: 0.6161113576889038
Epoch 3: Train err: 0.34375, Train loss: 0.6098222947120666 |Validation err: 0.3325, Validation loss: 0.6260210764408112
Epoch 4: Train err: 0.314375, Train loss: 0.5849691489338875 |Validation err: 0.34, Validation loss: 0.6044013917446136
Epoch 5: Train err: 0.301125, Train loss: 0.5689119303822517 |Validation err: 0.3125, Validation loss: 0.576918310880661
Epoch 6: Train err: 0.281, Train loss: 0.5452213581204415 |Validation err: 0.308, Validation loss: 0.5708447456359863
Epoch 7: Train err: 0.270875, Train loss: 0.5272981298565864 |Validation err: 0.307, Validation loss: 0.5854293291568756
Epoch 8: Train err: 0.259375, Train loss: 0.5070905526578426 |Validation err: 0.313, Validation loss: 0.5877130818367005
Epoch 9: Train err: 0.242375, Train loss: 0.4968344421982765 |Validation err: 0.313, Validation loss: 0.5922425072193146
Epoch 10: Train err: 0.236375, Train loss: 0.4756101597249508 |Validation err: 0.297, Validation loss: 0.5718690166473389
Epoch 11: Train err: 0.222125, Train loss: 0.4599769461452961 |Validation err: 0.2975, Validation loss: 0.6376970833539963
Epoch 12: Train err: 0.211, Train loss: 0.4454492371380329 |Validation err: 0.2995, Validation loss: 0.609202565908432
Epoch 13: Train err: 0.19875, Train loss: 0.4245421719551086 |Validation err: 0.3075, Validation loss: 0.6494987765550614
Epoch 14: Train err: 0.18675, Train loss: 0.4007472907453775 |Validation err: 0.3085, Validation loss: 0.6610016552209854
Epoch 15: Train err: 0.1645, Train loss: 0.3759974058121443 |Validation err: 0.3105, Validation loss: 0.7106090537309646
Epoch 16: Train err: 0.16125, Train loss: 0.3591455406397581 |Validation err: 0.3005, Validation loss: 0.7310364942550659
Epoch 17: Train err: 0.15775, Train loss: 0.3463234790861607 |Validation err: 0.307, Validation loss: 0.7263009325265884
Epoch 18: Train err: 0.141625, Train loss: 0.32175366275012496 |Validation err: 0.3195, Validation loss: 0.7913952842950821
Epoch 19: Train err: 0.13375, Train loss: 0.30618105667084455 |Validation err: 0.335, Validation loss: 0.8032052783966065
Epoch 20: Train err: 0.126625, Train loss: 0.3029071792438626 |Validation err: 0.32, Validation loss: 0.8106685240268707
Epoch 21: Train err: 0.12025, Train loss: 0.28682796490937473 |Validation err: 0.3205, Validation loss: 0.8259474284648896
Epoch 22: Train err: 0.1165, Train loss: 0.27489088076353074 |Validation err: 0.352, Validation loss: 0.8937610774040222
Epoch 23: Train err: 0.104375, Train loss: 0.2467898527495563 |Validation err: 0.3315, Validation loss: 1.0021928198337555
Epoch 24: Train err: 0.101, Train loss: 0.23970085787773132 |Validation err: 0.331, Validation loss: 1.1290796399116516
Epoch 25: Train err: 0.09575, Train loss: 0.23643119425699116 |Validation err: 0.3315, Validation loss: 1.1338514368534087
Epoch 26: Train err: 0.094125, Train loss: 0.2325953512713313 |Validation err: 0.3365, Validation loss: 1.1414263204336166
Epoch 27: Train err: 0.08425, Train loss: 0.21040759468451142 |Validation err: 0.3335, Validation loss: 1.1823678107261657
Epoch 28: Train err: 0.0825, Train loss: 0.20643112615589052 |Validation err: 0.323, Validation loss: 1.266836181640625
Epoch 29: Train err: 0.0845, Train loss: 0.21273409337876364 |Validation err: 0.3245, Validation loss: 1.406717705130577
Epoch 30: Train err: 0.071375, Train loss: 0.18387044295761734 |Validation err: 0.345, Validation loss: 1.4871552000045776
Finished Training
Total time elapsed: 235.51 seconds
```





```
None
```

Decreasing the batch size from 64 to 16 causes the model to take 73 seconds longer (235 sec vs 162 sec) as compared to the default case, this might be becuase of the increased number of iterations required to go through in each epoch.

The main difference from the default case is that there is now more overfitting, starting as early as the 10th epoch as both the training error and loss curves follow a simialr downward curve but the validation error seems to stop decreasing and only fluctuate within bounds, and the validation loss starts going higher post the 10th epoch. It is also important to note that the training error is significantly lower comapred to default, decreasing from 0.17 to 0.07, but the validation error is now a lot higher than the default case increasing from 0.17 to 0.34 after all 30 epochs. The loss is also lower for training comapred to the default case (0.18 vs 0.46), but once again the validation loss is extremely high compared to default after 30 epochs, increasing from 0.64 to 1.48.

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.

Based on the results of all the different hyperparameter combinations tried in the parts above it seems that the large_net model is a better choice than small_net as it produces lower training and validation errors. It also produces a lower validation loss, suggesting lesser signs of overfitting when compared with other error/loss curves.

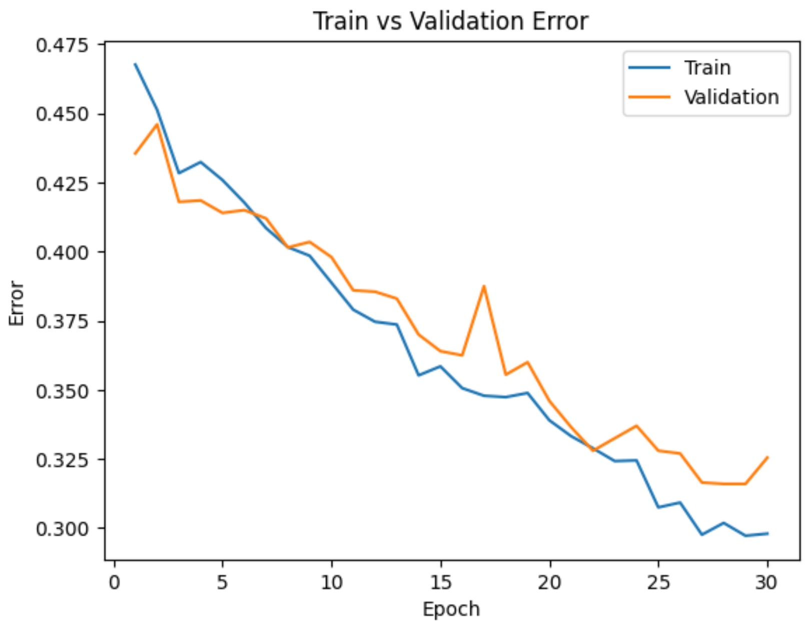
A common problem with the large_net model was the presence of overfitting with default hyperparameter settings. A good plan would be to pick 0.01 as the learning rate as it does not overfit to the training data like 0.1 and also doesn't undefit as much as 0.001. It also seems like a batch_size of 512 eliminates the overfitting problem that might be caused as a result of choosing large_net, so that would be my choice over 64 or 16 batch_sizes as they too overfit to the training data, but I think the optimal balance would be achieved by using a batch_size of 256. I would keep the epochs to 30 as it seems to be a good balance that allows getting past the underfitting stage but also preventing overfitting stages. So, large_net, learning rate = 0.01, batch_size = 256, epochs = 30 would be my startegy to reduce the validation error while keeping the validation loss in check, to improve the validation accuracy.

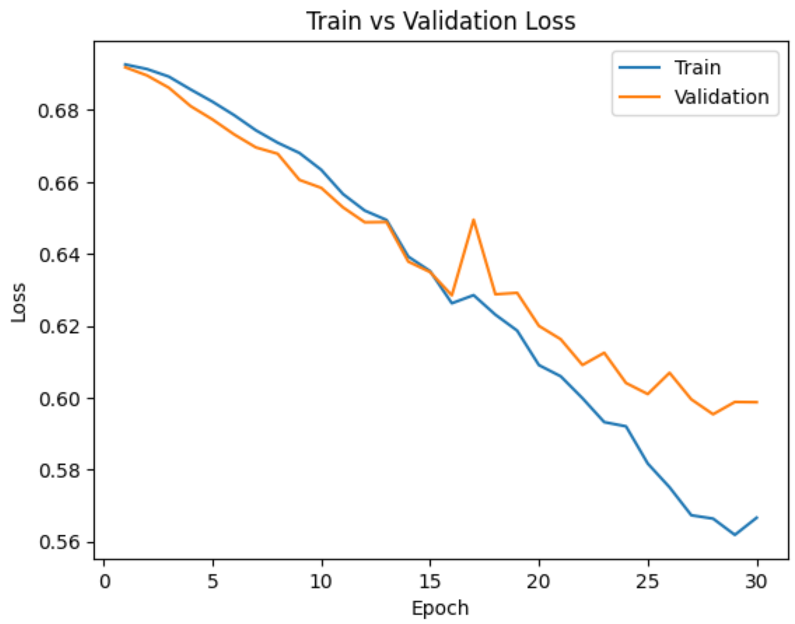
Part (b) - 1pt

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

```
In [26]: large_net = LargeNet()
train_net(large_net, 256, 0.01, 30)
model_path = get_model_name("large", batch_size=256, learning_rate=0.01, epoch=29)
print(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: 0.6918838396668434
Epoch 2: Train err: 0.45125, Train loss: 0.6913921367377043 |Validation err: 0.446, Validation loss: 0.6896143183112144
Epoch 3: Train err: 0.428375, Train loss: 0.6892956402152777 |Validation err: 0.418, Validation loss: 0.6862075850367546
Epoch 4: Train err: 0.432375, Train loss: 0.6857246868312359 |Validation err: 0.4185, Validation loss: 0.6810905262827873
Epoch 5: Train err: 0.425875, Train loss: 0.6823424641042948 |Validation err: 0.414, Validation loss: 0.6774001196026802
Epoch 6: Train err: 0.41775, Train loss: 0.6785750649869442 |Validation err: 0.415, Validation loss: 0.6732205599546432
Epoch 7: Train err: 0.4085, Train loss: 0.6743601486086845 |Validation err: 0.412, Validation loss: 0.6696006879210472
Epoch 8: Train err: 0.401625, Train loss: 0.6709103956818581 |Validation err: 0.4015, Validation loss: 0.6678383350372314
Epoch 9: Train err: 0.3985, Train loss: 0.668053587898612 |Validation err: 0.4035, Validation loss: 0.660559818148613
Epoch 10: Train err: 0.38875, Train loss: 0.663382263854146 |Validation err: 0.398, Validation loss: 0.658350445330143
Epoch 11: Train err: 0.379, Train loss: 0.6566288601607084 |Validation err: 0.386, Validation loss: 0.652928002178669
Epoch 12: Train err: 0.374625, Train loss: 0.6520253773778677 |Validation err: 0.3855, Validation loss: 0.6487779468297958
Epoch 13: Train err: 0.373625, Train loss: 0.6494111102074385 |Validation err: 0.383, Validation loss: 0.6488563939929008
Epoch 14: Train err: 0.35525, Train loss: 0.6392229404300451 |Validation err: 0.37, Validation loss: 0.6378394290804863
Epoch 15: Train err: 0.3585, Train loss: 0.635199349373579 |Validation err: 0.364, Validation loss: 0.6349294185638428
Epoch 16: Train err: 0.350625, Train loss: 0.6262455694377422 |Validation err: 0.3625, Validation loss: 0.6284593492746353
Epoch 17: Train err: 0.347875, Train loss: 0.6285011693835258 |Validation err: 0.3875, Validation loss: 0.6494947820901871
Epoch 18: Train err: 0.347375, Train loss: 0.6230689510703087 |Validation err: 0.3555, Validation loss: 0.6287511587142944
Epoch 19: Train err: 0.348875, Train loss: 0.6186420768499374 |Validation err: 0.36, Validation loss: 0.6291449218988419
Epoch 20: Train err: 0.339, Train loss: 0.6090399734675884 |Validation err: 0.346, Validation loss: 0.6199502795934677
Epoch 21: Train err: 0.33325, Train loss: 0.6059376578778028 |Validation err: 0.3365, Validation loss: 0.6162464991211891
Epoch 22: Train err: 0.328875, Train loss: 0.5998399555683136 |Validation err: 0.328, Validation loss: 0.609074093401432
Epoch 23: Train err: 0.32425, Train loss: 0.5931875444948673 |Validation err: 0.3325, Validation loss: 0.6124721989035606
Epoch 24: Train err: 0.3245, Train loss: 0.5920102782547474 |Validation err: 0.337, Validation loss: 0.6040716990828514
Epoch 25: Train err: 0.3075, Train loss: 0.5816512629389763 |Validation err: 0.328, Validation loss: 0.6009927093982697
Epoch 26: Train err: 0.30925, Train loss: 0.575066328048706 |Validation err: 0.327, Validation loss: 0.6069098189473152
Epoch 27: Train err: 0.297625, Train loss: 0.5672740656882524 |Validation err: 0.3165, Validation loss: 0.5995183810591698
Epoch 28: Train err: 0.301875, Train loss: 0.5663426201790571 |Validation err: 0.316, Validation loss: 0.5953697860240936
Epoch 29: Train err: 0.29725, Train loss: 0.5618185754865408 |Validation err: 0.316, Validation loss: 0.5987934470176697
Epoch 30: Train err: 0.298, Train loss: 0.5665905568748713 |Validation err: 0.3255, Validation loss: 0.5987277328968048
Finished Training
Total time elapsed: 148.47 seconds
```





None

Part (c) - 2pt

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

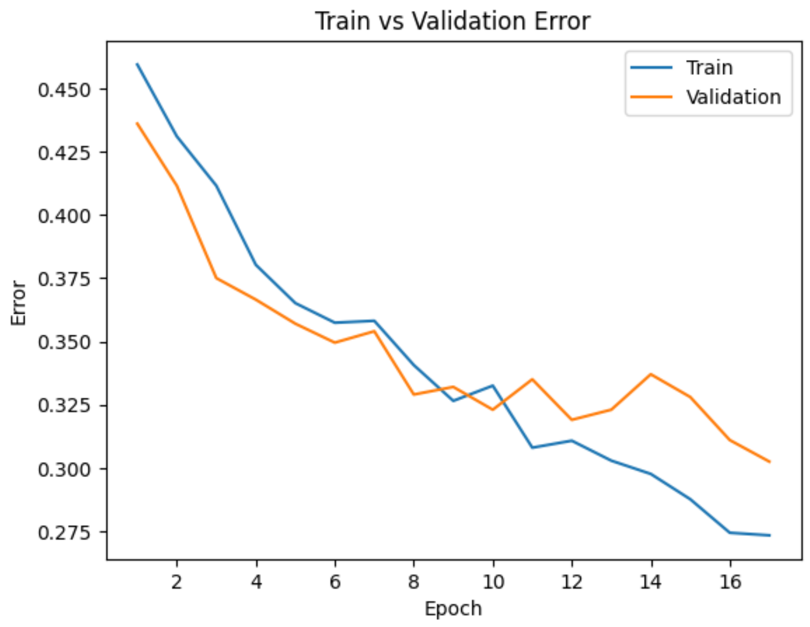
It seems from the results above that even though a batch_size of 256 did a good job of reducing the validation error, there is an instance of the beginnings of a overfitting stage at epoch 17, so I would reduce the epochs to 17 now. I also think the learning rate of 0.01 is a bit too low and not decreasing the validation loss quick enough, so I will now change that to 0.025 in order to initiate a quicker decrease in the loss, preventing any overfitting that might be left. So the new set of hyperparameters are now large_net, batch_size = 256, learning_rate = 0.025, epochs = 17.

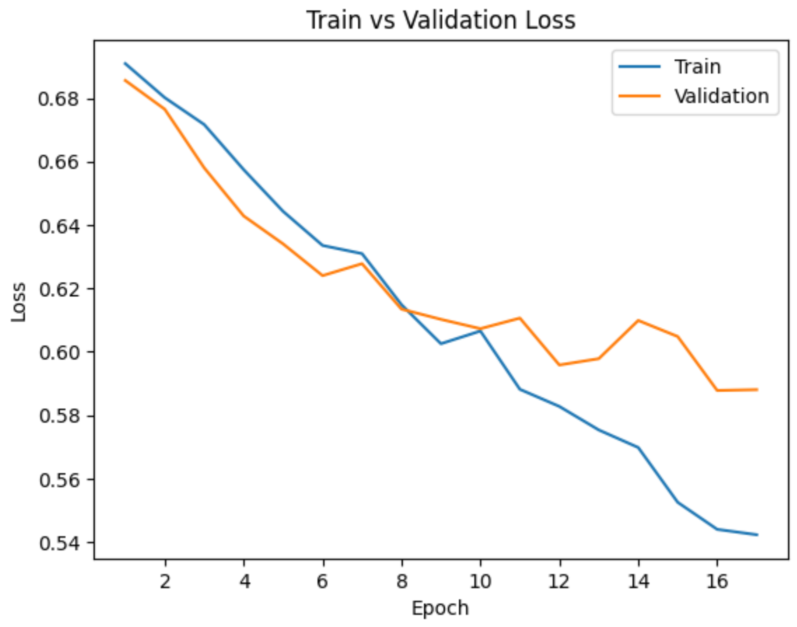
Part (d) - 1pt

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

In [28]:
```
large_net = LargeNet()
train_net(large_net, 256, 0.025, 17)
model_path = get_model_name("large", batch_size=256, learning_rate=0.025, epoch=16)
print(plot_training_curve(model_path))
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.459375, Train loss: 0.690877053886652 |Validation err: 0.436, Validation loss: 0.6855586245656013
Epoch 2: Train err: 0.431, Train loss: 0.6801417786628008 |Validation err: 0.4115, Validation loss: 0.6764916107058525
Epoch 3: Train err: 0.4115, Train loss: 0.6716584824025631 |Validation err: 0.375, Validation loss: 0.6580018103122711
Epoch 4: Train err: 0.38025, Train loss: 0.6574407164007425 |Validation err: 0.3665, Validation loss: 0.6428290829062462
Epoch 5: Train err: 0.365125, Train loss: 0.6442770082503557 |Validation err: 0.357, Validation loss: 0.634015865623951
Epoch 6: Train err: 0.357375, Train loss: 0.6335296276956797 |Validation err: 0.3495, Validation loss: 0.6240116059780121
Epoch 7: Train err: 0.358125, Train loss: 0.6309494897723198 |Validation err: 0.354, Validation loss: 0.6277857348322868
Epoch 8: Train err: 0.340625, Train loss: 0.614841278642416 |Validation err: 0.329, Validation loss: 0.6134492456912994
Epoch 9: Train err: 0.3265, Train loss: 0.6025369837880135 |Validation err: 0.332, Validation loss: 0.6102249175310135
Epoch 10: Train err: 0.3325, Train loss: 0.6065482012927532 |Validation err: 0.323, Validation loss: 0.6072827726602554
Epoch 11: Train err: 0.308, Train loss: 0.5881668906658888 |Validation err: 0.335, Validation loss: 0.6106195002794266
Epoch 12: Train err: 0.31075, Train loss: 0.5827658642083406 |Validation err: 0.319, Validation loss: 0.5958363711833954
Epoch 13: Train err: 0.302875, Train loss: 0.5753433723002672 |Validation err: 0.323, Validation loss: 0.5978229641914368
Epoch 14: Train err: 0.297625, Train loss: 0.5698130149394274 |Validation err: 0.337, Validation loss: 0.6098785027861595
Epoch 15: Train err: 0.287625, Train loss: 0.5525306053459644 |Validation err: 0.328, Validation loss: 0.6048072800040245
Epoch 16: Train err: 0.274375, Train loss: 0.5440043527632952 |Validation err: 0.311, Validation loss: 0.5878000482916832
Epoch 17: Train err: 0.273375, Train loss: 0.5423074718564749 |Validation err: 0.3025, Validation loss: 0.5880512148141861
Finished Training
Total time elapsed: 82.27 seconds
```





```
None
```

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 [29]:
```
net = LargeNet()
model_path = get_model_name(net.name, batch_size=256, learning_rate=0.025, epoch=16)
state = torch.load(model_path)
net.load_state_dict(state)
```

Out[29]:
```
<All keys matched successfully>
```

In [ ]:

Part (b) - 2pt

Justify your choice of model from part (a).

Based on the results from Part 2,3,4 above I chose large_net as it prodcues lower validation error and loss results as compared to small_net as seen in part 2d, and even though it overfits a bit more than small_net, this can be avoided by the hyperparameter values mentioned below. Thus a better starting point overall.

I chose a batch_size of 256 as it did a good job of reducing the validation error/loss while ensuring that there was no obvious over/underfitting as seen in part 4c,d.

I chose 17 epochs as there is an instance of the beginnings of a overfitting stage at epoch 18 as seen in part 4a,b, so reducing to 17 epochs helps avoid this.

I chose a learning rate of 0.025 in order to initiate a quicker decrease in the loss, preventing any overfitting that might be left as seen in part 4c,d.

As observed in the results, the validation error is very similar to the default model and the validation loss is a lot lower (0.58 vs 0.64), thus avoiding overfitting and increasing validation accuracy.

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 [31]:
```
# 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=256)

#using the code from Lab 1 part 5 and tutorial notebooks
criterion = nn.BCEWithLogitsLoss()
testing_error, testing_loss = evaluate(net, test_loader, criterion)
print ("Testing error : ", testing_error)
print ("Testing loss : ", testing_loss)
```

```
Files already downloaded and verified
Files already downloaded and verified
Testing error :  0.304
Testing loss :  0.5875863283872604
```

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 [32]:
```
# 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=256)

#using the code from Lab 1 part 5
criterion = nn.BCEWithLogitsLoss()
validation_error, validation_loss = evaluate(net, val_loader, criterion)
print ("Validation error : ", validation_error)
print ("Validation loss : ", validation_loss)
```

```
Files already downloaded and verified
Files already downloaded and verified
Validation error :  0.3025
Validation loss :  0.5869087502360344
```

The test classification error of 0.304 is slightly higher than the validation error of 0.3025. This is because the way we tuned our hyperparameters was to ensure that the validation error was lowered as much as possible based on the various results seen in the previous parts. Also the model has continuously been exposed to the training and validation data in an attempt to maximize its accuracy on unseen data like the testing data. This has allowed the model to familiarize itself with the images in the training and validation sets a lot more than it has with the testing set which it has only seen once. The testing error is a real indication of the model performance in a real world setting, and thus it would be slightly higher than the validation error which the model has based its predictions on.

Part (e) - 2pt

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

The test data was only used at the very end to ensure that the model did not familiarize itself to it. This is an accurate measure of how the model would perform in a real world setting on data it has never seen before, allowing us to get the true performance. Doing so also allows us to not tune the hyperparameters in such a way that they overfit themselves to the testing data while training the model and produce biased results, giving us an inaccurate measure of the true model performance.

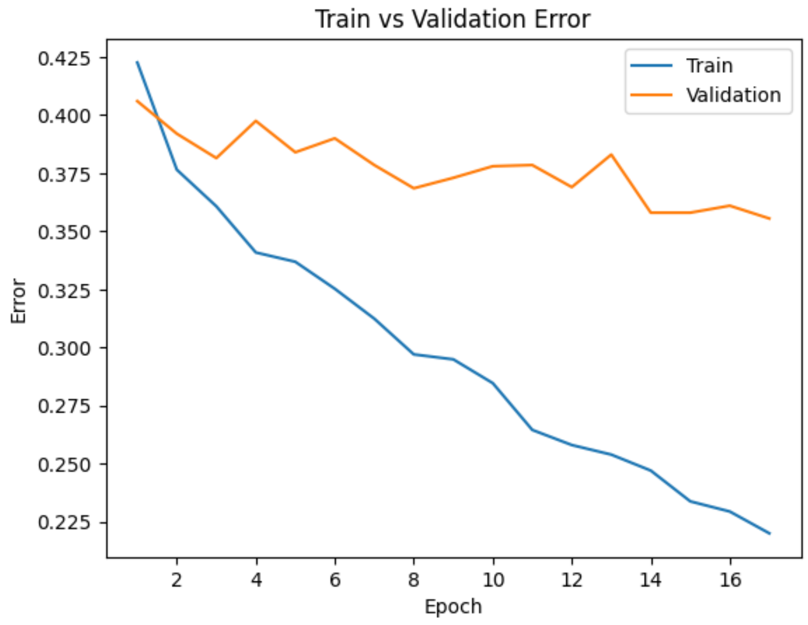
Part (f) - 5pt

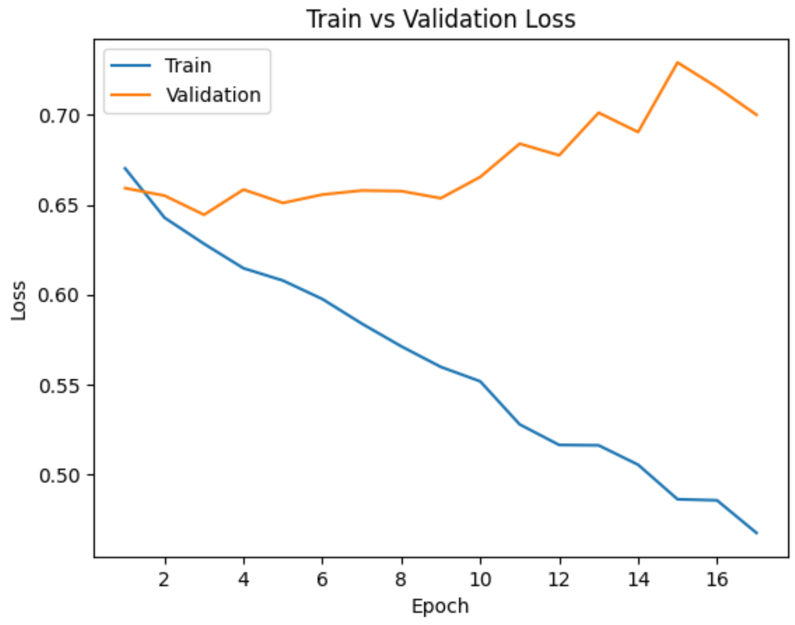
How does the your best CNN model compare with an 2-layer ANN model (no convolutional layers) on classifying cat and dog images. You can use a 2-layer ANN architecture similar to what you used in Lab 1. You should explore different hyperparameter settings to determine how well you can do on the validation 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 [39]:
```
#using code from Lab 1 to compare CNN model to 2-layer ANN model (Pigeon)

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

# define a 2-layer artificial neural network
class Pigeon(nn.Module):
    def __init__(self):
        super(Pigeon, self).__init__()
        self.name = "pigeon_ANN"
        #flattening the RGB scale tensors
        self.layer1 = nn.Linear(32*32*3, 30)
        self.layer2 = nn.Linear(30, 1)
    def forward(self, img):
       #flattening the RGB scale tensors
        flattened = img.view(-1, 32*32*3)
        activation1 = self.layer1(flattened)
        activation1 = F.relu(activation1)
        activation2 = self.layer2(activation1)
        #using the squeeze function on tensors to concatenate them for the ANN
        activation2 = activation2.squeeze(1)
        return activation2

pigeon_ANN = Pigeon()

train_net(pigeon_ANN, 256, 0.025, 17)
model_path = get_model_name("pigeon_ANN", batch_size=256, learning_rate=0.025, epoch=16)
plot_training_curve(model_path)

# 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=256)

#using the code from Lab 1 part 5
criterion = nn.BCEWithLogitsLoss()
testing_error, testing_loss = evaluate(pigeon_ANN, test_loader, criterion)
print ("Testing error : ", testing_error)
print ("Testing loss : ", testing_loss)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.422625, Train loss: 0.6703135017305613 |Validation err: 0.406, Validation loss: 0.6593176648020744
Epoch 2: Train err: 0.3765, Train loss: 0.6428378708660603 |Validation err: 0.392, Validation loss: 0.6551310867071152
Epoch 3: Train err: 0.36075, Train loss: 0.6283104512840509 |Validation err: 0.3815, Validation loss: 0.6444747820496559
Epoch 4: Train err: 0.340875, Train loss: 0.6146906968206167 |Validation err: 0.3975, Validation loss: 0.6585051268339157
Epoch 5: Train err: 0.336875, Train loss: 0.6078445799648762 |Validation err: 0.384, Validation loss: 0.6510740369558334
Epoch 6: Train err: 0.32525, Train loss: 0.597529336810112 |Validation err: 0.39, Validation loss: 0.6557098105549812
Epoch 7: Train err: 0.312375, Train loss: 0.5838396530598402 |Validation err: 0.3785, Validation loss: 0.6579889133572578
Epoch 8: Train err: 0.297, Train loss: 0.571203300729394 |Validation err: 0.3685, Validation loss: 0.6576461419463158
Epoch 9: Train err: 0.294875, Train loss: 0.5597567521035671 |Validation err: 0.373, Validation loss: 0.6536862179636955
Epoch 10: Train err: 0.284625, Train loss: 0.551705090329051 |Validation err: 0.378, Validation loss: 0.6655291989445686
Epoch 11: Train err: 0.2645, Train loss: 0.5277302311733365 |Validation err: 0.3785, Validation loss: 0.6840528920292854
Epoch 12: Train err: 0.258, Train loss: 0.5162823498249054 |Validation err: 0.369, Validation loss: 0.6775994822382927
Epoch 13: Train err: 0.253875, Train loss: 0.5160815734416246 |Validation err: 0.383, Validation loss: 0.7012975439429283
Epoch 14: Train err: 0.247, Train loss: 0.5052963802590966 |Validation err: 0.358, Validation loss: 0.6905441582202911
Epoch 15: Train err: 0.23375, Train loss: 0.4860480474308133 |Validation err: 0.358, Validation loss: 0.7292773947119713
Epoch 16: Train err: 0.229375, Train loss: 0.4854626450687647 |Validation err: 0.361, Validation loss: 0.7155099511146545
Epoch 17: Train err: 0.22, Train loss: 0.46732200402766466 |Validation err: 0.3555, Validation loss: 0.7001607939600945
Finished Training
Total time elapsed: 64.36 seconds
```





```
Files already downloaded and verified
Files already downloaded and verified
Testing error :  0.3595
Testing loss :  0.7043357640504837
```

The 2-layer ANN model perfoms worse than the CNN model as it produces a higher testing error (0.304 vs 0.3595) and also a higher testing loss (0.58 vs 0.70). These classification readings indicate that there is a presence of overfitting on validation data, as is also confirmed by the increase in validation loss with increasing epochs even when the error is reducing. Thus the CNN model performs significantly better and should be the best choice for the dogs vs cats classification dataset.