Lab 2: Cats vs Dogs

Deadline: Feb 01, 5:00pm

Late Penalty: There is a penalty-free grace period of one hour past the deadline. Any work that is submitted between 1 hour and 24 hours past the deadline will receive a 20% grade deduction. No other late work is accepted. Quercus submission time will be used, not your local computer time. You can submit your labs as many times as you want before the deadline, so please submit often and early.

Marking TA: Tinglin (Francis) Duan

This lab is partially based on an assignment developed by Prof. Jonathan Rose and Harris Chan.

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/15M59eBaLjYy_X1m-Yriu3iuEtHZu5USZ/view?usp=sharing (https://drive.google.com/file/d/15M59eBaLjYy_X1m-Yriu3iuEtHZu5USZ/view?usp=sharing)

In [26]:
```
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 [27]:
```
#################################################################################
#
# 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, validati
on
    and testing datasets. Returns data loaders for the three preprocessed data
sets.

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

    Returns:
        train_loader: iterable training dataset organized according to batch s
ize
        val_loader: iterable validation dataset organized according to batch s
ize
        test_loader: iterable testing dataset organized according to batch siz
e
        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)),transforms.Gra
yscale(num_output_channels=1)])
```

```
    # 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], r
elevant_indices[split:]
    train_sampler = SubsetRandomSampler(relevant_train_indices)
    train_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size
,
                                               num_workers=1, sampler=train_sa
mpler)
    val_sampler = SubsetRandomSampler(relevant_val_indices)
    val_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                             num_workers=1, sampler=val_sampl
er)
    # Load CIFAR10 testing data
    testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                           download=True, transform=transform)
    # Get the list of indices to sample from
    relevant_test_indices = get_relevant_indices(testset, classes, target_clas
ses)
    test_sampler = SubsetRandomSampler(relevant_test_indices)
    test_loader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                              num_workers=1, sampler=test_sampl
er)
    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 val
ues

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

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

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

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

     Args:
         net: PyTorch neural network object
         loader: PyTorch data loader for the validation set
         criterion: The loss function
     Returns:
         err: A scalar for the avg classification error over the validation se
t
         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 (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 [28]:
```
# This will download the CIFAR-10 dataset to a folder called "data"
# the first time you run this code.
train_loader, val_loader, test_loader, classes = get_data_loader(
    target_classes=["cat", "dog"],
    batch_size=1) # One image per batch
```

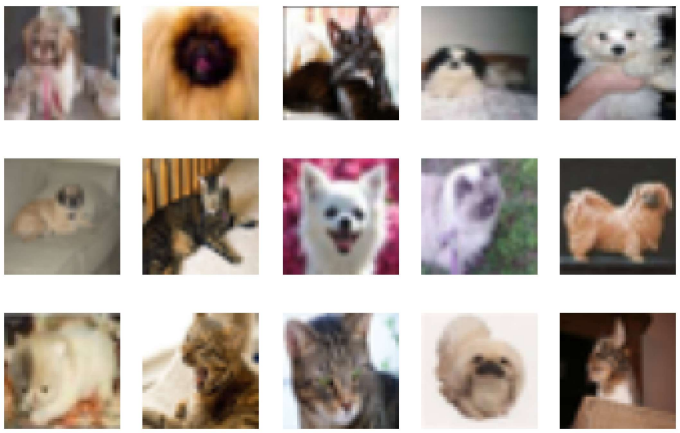
```
Files already downloaded and verified
Files already downloaded and verified
```

Part (a) -- 1 pt

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

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

In [14]:
```
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 [12]:
```
print("There are", len(train_loader),"training examples.")
print("There are", len(val_loader),"validation examples.")
print("There are", len(test_loader),"test examples.")
```

```
There are 8000 training examples.
There are 2000 validation examples.
There are 2000 test examples.
```

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?

In [16]:
```
# We need a validation set because we want to set aside a separate data set to
track validation accuracy in the training curve and make decisions
# about the model architecture (hyperparameters) using the validation set.
# Judging the performance of the model based on the training set can cause ove
rfitting problem that will cause the trained model to
# overly reply on the training data and thus lose the ability to generalize to
all possible futrue data. In another word, the errors will increase
# drastically when the model is tested with 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 [29]:
```
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 [30]:
```
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 [31]:
```
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.

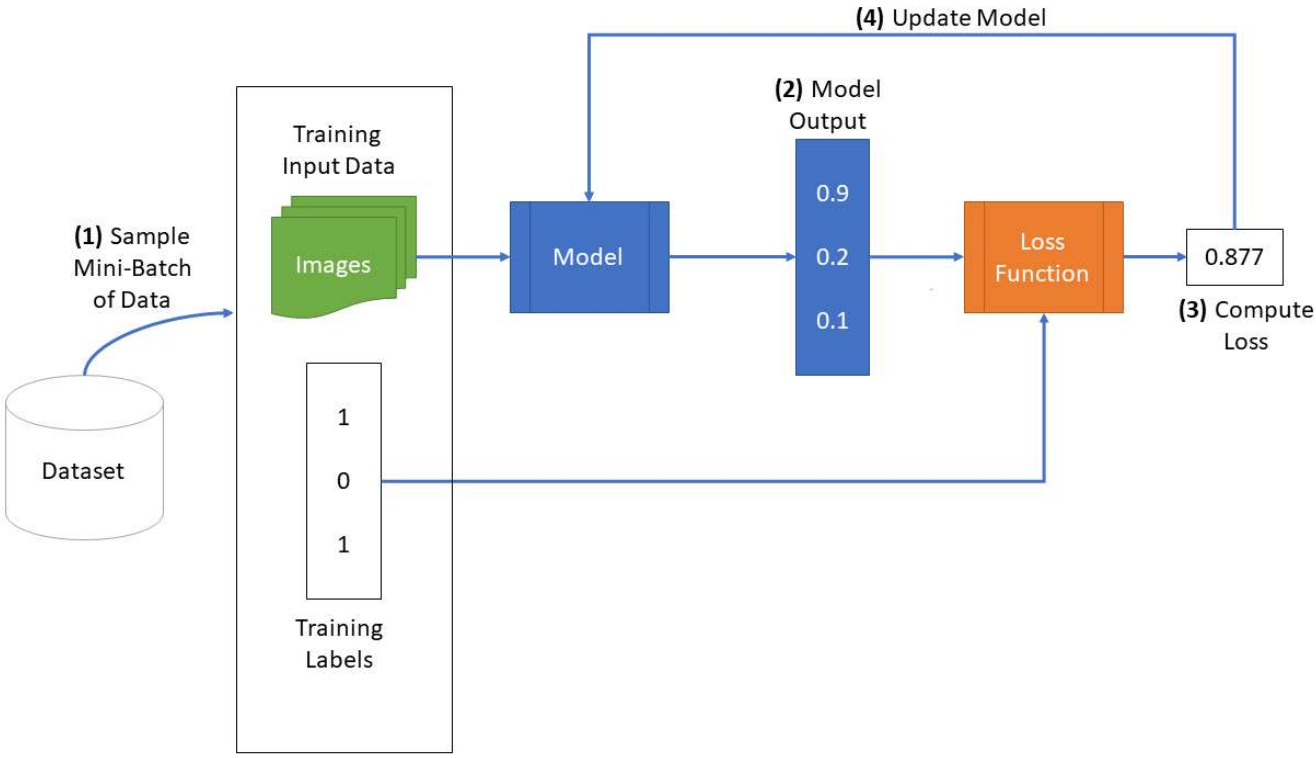
What is the total number of parameters in `small_net` and in `large_net` ? (Hint: how many numbers are in each tensor?)

In [33]:
```
print("Small_net:")
for param in small_net.parameters():
    print(param.shape)

print("Large_net:")
for param in large_net.parameters():
    print(param.shape)

print(5*3*3*3 + 5 +245+1,"parameters for small net")
print(5*3*5*5 + 5 + 10*5*5*5 + 10 + 32*250 +32 +32+1,"parameters for large ne
t")
```

```
Small_net:
torch.Size([5, 3, 3, 3])
torch.Size([5])
torch.Size([1, 245])
torch.Size([1])
Large_net:
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])
386 parameters for small net
9705 parameters for large net
```

The function train_net

The function `train_net` below takes an untrained neural network (like `small_net` and `large_net`) and several other parameters. You should be able to understand how this function works. The figure below shows the high level training loop for a machine learning model:



In [18]:
```
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 [28]:
```
print("The parameters and their default values are:\n","batch_size=64, learnin
g_rate=0.01, num_epochs=30")
```

```
The parameters and their 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 [29]:
```
train_net(small_net, batch_size=64, learning_rate=0.01, num_epochs=5)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.4295, Train loss: 0.6745695013999939 |Validation err:
0.378, Validation loss: 0.6548647321760654
Epoch 2: Train err: 0.363, Train loss: 0.6436164474487305 |Validation err: 0.
371, Validation loss: 0.6547714974731207
Epoch 3: Train err: 0.349875, Train loss: 0.6314334268569947 |Validation err:
0.3505, Validation loss: 0.6241133566945791
Epoch 4: Train err: 0.338125, Train loss: 0.6180711879730224 |Validation err:
0.3545, Validation loss: 0.624964639544487
Epoch 5: Train err: 0.335375, Train loss: 0.6125645656585693 |Validation err:
0.344, Validation loss: 0.6192154120653868
Finished Training
Total time elapsed: 20.85 seconds
```

The following non-csv files contain the loss and error data for the validation and train set calculated in each epoch. model_small_bs64_lr0.01_epoch0 contains model_small_bs64_lr0.01_epoch1 model_small_bs64_lr0.01_epoch2 model_small_bs64_lr0.01_epoch3 model_small_bs64_lr0.01_epoch4
model_small_bs64_lr0.01_epoch4_train_err.csv contains the calculated train errors after each epoch
model_small_bs64_lr0.01_epoch4_train_loss.csv contains the calculated train loss after each epoch
model_small_bs64_lr0.01_epoch4_val_err.csv contains the calculated validation errors after each epoch
model_small_bs64_lr0.01_epoch4_val_loss.csv contains the calculated validation loss after each 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 [30]:
```
# 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 [31]:
```
train_net(small_net)
train_net(large_net)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.3295, Train loss: 0.6051668682098389 |Validation err:
0.3365, Validation loss: 0.6178892850875854
Epoch 2: Train err: 0.32275, Train loss: 0.6000635211467743 |Validation err:
0.352, Validation loss: 0.6331383101642132
Epoch 3: Train err: 0.321, Train loss: 0.5951693964004516 |Validation err: 0.
3335, Validation loss: 0.6046995678916574
Epoch 4: Train err: 0.305375, Train loss: 0.5859285674095154 |Validation err:
0.3385, Validation loss: 0.608858710154891
Epoch 5: Train err: 0.301875, Train loss: 0.5791746554374695 |Validation err:
0.318, Validation loss: 0.5989876892417669
Epoch 6: Train err: 0.2955, Train loss: 0.5711853466033936 |Validation err:
0.3255, Validation loss: 0.6121932277455926
Epoch 7: Train err: 0.2955, Train loss: 0.5706627750396729 |Validation err:
0.332, Validation loss: 0.5922151263803244
Epoch 8: Train err: 0.28825, Train loss: 0.5634026775360107 |Validation err:
0.3075, Validation loss: 0.5886584529653192
Epoch 9: Train err: 0.29, Train loss: 0.5666917026042938 |Validation err: 0.3
075, Validation loss: 0.5838038353249431
Epoch 10: Train err: 0.281, Train loss: 0.557926882982254 |Validation err: 0.
303, Validation loss: 0.5805829223245382
Epoch 11: Train err: 0.2835, Train loss: 0.5577404987812042 |Validation err:
0.316, Validation loss: 0.5888388101011515
Epoch 12: Train err: 0.279125, Train loss: 0.5506557452678681 |Validation er
r: 0.317, Validation loss: 0.5950328893959522
Epoch 13: Train err: 0.27775, Train loss: 0.5515842080116272 |Validation err:
0.3155, Validation loss: 0.5925341928377748
Epoch 14: Train err: 0.277875, Train loss: 0.5462580037117004 |Validation er
r: 0.3075, Validation loss: 0.5853533670306206
Epoch 15: Train err: 0.27575, Train loss: 0.5462239012718201 |Validation err:
0.297, Validation loss: 0.5736525971442461
Epoch 16: Train err: 0.277125, Train loss: 0.551639577627182 |Validation err:
0.302, Validation loss: 0.5878525460138917
Epoch 17: Train err: 0.27975, Train loss: 0.5470677170753478 |Validation err:
0.2995, Validation loss: 0.5761091131716967
Epoch 18: Train err: 0.27625, Train loss: 0.5428227627277374 |Validation err:
0.31, Validation loss: 0.5843090489506721
Epoch 19: Train err: 0.27125, Train loss: 0.541426822900772 |Validation err:
0.307, Validation loss: 0.6029842030256987
Epoch 20: Train err: 0.27125, Train loss: 0.5405812273025513 |Validation err:
0.303, Validation loss: 0.5875138659030199
Epoch 21: Train err: 0.27075, Train loss: 0.5428693442344665 |Validation err:
0.2955, Validation loss: 0.5701606124639511
Epoch 22: Train err: 0.277125, Train loss: 0.5425244340896607 |Validation er
r: 0.307, Validation loss: 0.587181924842298
Epoch 23: Train err: 0.269, Train loss: 0.5398602492809296 |Validation err:
0.294, Validation loss: 0.5744608044624329
Epoch 24: Train err: 0.268, Train loss: 0.537346331357956 |Validation err: 0.
302, Validation loss: 0.5754719506949186
Epoch 25: Train err: 0.26825, Train loss: 0.535331017255783 |Validation err:
0.295, Validation loss: 0.5719010233879089
Epoch 26: Train err: 0.271125, Train loss: 0.5344478268623352 |Validation er
r: 0.294, Validation loss: 0.572728592902422
Epoch 27: Train err: 0.269, Train loss: 0.5345914597511292 |Validation err:
0.287, Validation loss: 0.5790974702686071
Epoch 28: Train err: 0.270625, Train loss: 0.5365882682800293 |Validation er
```

```
r: 0.2985, Validation loss: 0.5716164456680417
Epoch 29: Train err: 0.27575, Train loss: 0.5361437737941742 |Validation err:
0.2965, Validation loss: 0.5736283138394356
Epoch 30: Train err: 0.27075, Train loss: 0.5378967094421386 |Validation err:
0.299, Validation loss: 0.5808860855177045
Finished Training
Total time elapsed: 120.84 seconds
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.468375, Train loss: 0.6902052545547486 |Validation err:
0.4425, Validation loss: 0.6822461802512407
Epoch 2: Train err: 0.42825, Train loss: 0.6800277309417725 |Validation err:
0.44, Validation loss: 0.6846732944250107
Epoch 3: Train err: 0.4045, Train loss: 0.6681363906860351 |Validation err:
0.3745, Validation loss: 0.6540922746062279
Epoch 4: Train err: 0.379, Train loss: 0.6518011727333068 |Validation err: 0.
366, Validation loss: 0.6501812729984522
Epoch 5: Train err: 0.35875, Train loss: 0.6370277795791626 |Validation err:
0.362, Validation loss: 0.6354033537209034
Epoch 6: Train err: 0.341125, Train loss: 0.6213076267242431 |Validation err:
0.3465, Validation loss: 0.6279278136789799
Epoch 7: Train err: 0.334125, Train loss: 0.6129170560836792 |Validation err:
0.333, Validation loss: 0.6166572999209166
Epoch 8: Train err: 0.322625, Train loss: 0.5983357875347137 |Validation err:
0.335, Validation loss: 0.6113663204014301
Epoch 9: Train err: 0.31525, Train loss: 0.5913089616298676 |Validation err:
0.341, Validation loss: 0.6096827443689108
Epoch 10: Train err: 0.306875, Train loss: 0.5773086700439453 |Validation er
r: 0.321, Validation loss: 0.60216651763767
Epoch 11: Train err: 0.2975, Train loss: 0.5657535645961761 |Validation err:
0.323, Validation loss: 0.6017451956868172
Epoch 12: Train err: 0.286625, Train loss: 0.5542769396305084 |Validation er
r: 0.32, Validation loss: 0.5968012986704707
Epoch 13: Train err: 0.28, Train loss: 0.5438401865959167 |Validation err: 0.
3065, Validation loss: 0.5928338002413511
Epoch 14: Train err: 0.269, Train loss: 0.5355554623603821 |Validation err:
0.304, Validation loss: 0.6048325402662158
Epoch 15: Train err: 0.2695, Train loss: 0.5283188669681549 |Validation err:
0.304, Validation loss: 0.5960234487429261
Epoch 16: Train err: 0.260625, Train loss: 0.5178001472949981 |Validation er
r: 0.3095, Validation loss: 0.5898392861708999
Epoch 17: Train err: 0.2545, Train loss: 0.5143085143566132 |Validation err:
0.2995, Validation loss: 0.582975503988564
Epoch 18: Train err: 0.245, Train loss: 0.4961880655288696 |Validation err:
0.3005, Validation loss: 0.5904396735131741
Epoch 19: Train err: 0.242, Train loss: 0.4892804160118103 |Validation err:
0.2925, Validation loss: 0.5892045367509127
Epoch 20: Train err: 0.239625, Train loss: 0.4830480873584747 |Validation er
r: 0.302, Validation loss: 0.5947847319766879
Epoch 21: Train err: 0.22775, Train loss: 0.4712683465480804 |Validation err:
0.3015, Validation loss: 0.5941134421154857
Epoch 22: Train err: 0.225625, Train loss: 0.4627283730506897 |Validation er
r: 0.3195, Validation loss: 0.6105640884488821
Epoch 23: Train err: 0.211125, Train loss: 0.44887502121925354 |Validation er
r: 0.304, Validation loss: 0.5987749015912414
Epoch 24: Train err: 0.20625, Train loss: 0.4345599703788757 |Validation err:
0.3085, Validation loss: 0.657028628513217
```

```
Epoch 25: Train err: 0.200125, Train loss: 0.4246109311580658 |Validation er
r: 0.3025, Validation loss: 0.628796218894422
Epoch 26: Train err: 0.188875, Train loss: 0.40659145891666415 |Validation er
r: 0.3185, Validation loss: 0.6543826060369611
Epoch 27: Train err: 0.185375, Train loss: 0.39725749456882475 |Validation er
r: 0.3155, Validation loss: 0.6580766281113029
Epoch 28: Train err: 0.180375, Train loss: 0.3898140594959259 |Validation er
r: 0.3045, Validation loss: 0.6532812416553497
Epoch 29: Train err: 0.169625, Train loss: 0.37194368374347686 |Validation er
r: 0.314, Validation loss: 0.7486753845587373
Epoch 30: Train err: 0.15975, Train loss: 0.3547302296161652 |Validation err:
0.3185, Validation loss: 0.704806069843471
Finished Training
Total time elapsed: 135.62 seconds
```

The total time elapsed to train the small_net network is 120.84 seconds and the total time elapsed to train the large_net network is 135.62 seconds. The large_net network took loger time to train because the large_net has more neurons and layers to train.

Part (e) - 2pt

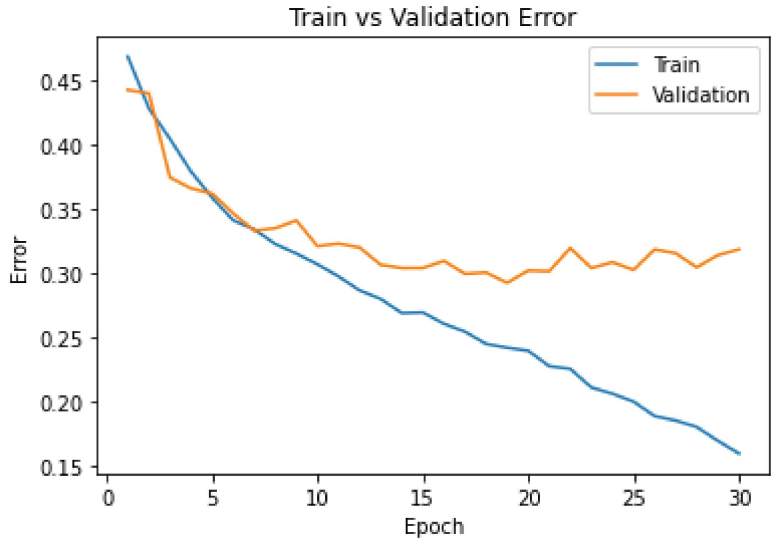
Use the function `plot_training_curve` to display the trajectory of the training/validation error and the training/validation loss. You will need to use the function `get_model_name` to generate the argument to the `plot_training_curve` function.

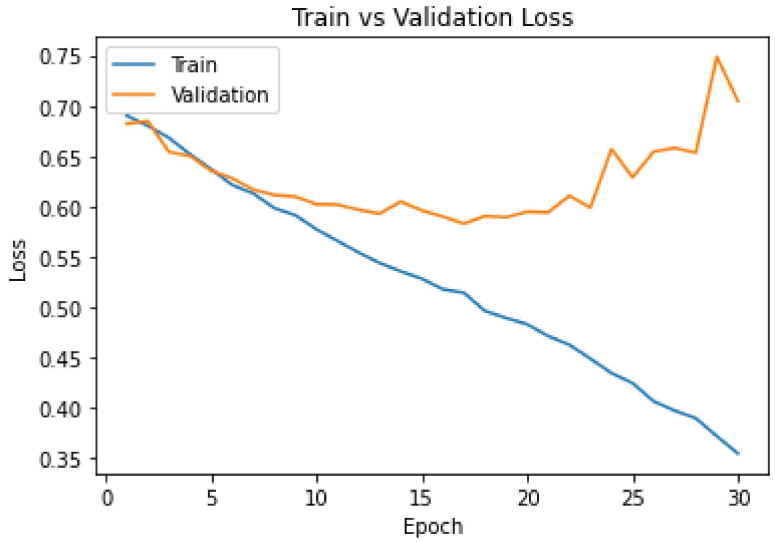
Do this for both the small network and the large network. Include both plots in your writeup.

In [40]:
```
model_path_small = get_model_name("small",batch_size=64, learning_rate=0.01, e
poch=29)
model_path_large = get_model_name("large",batch_size=64, learning_rate=0.01, e
poch=29)
print("Small model:")
plot_training_curve(model_path_small)
print("Large model:")
plot_training_curve(model_path_large)
```

Small model:

Large model:





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.

For the loss and error for the small model, both train and validation loss/error graphs follow a similiar trend respectively but the validation loss/error graphs are higher than the train loss/error graphs by a certain shift. For the loss and error curve for the large model, both train and validation loss/error graphs initially are very close to each other, but the validation loss/error graphs gradually deviates away from the train loss/error graphs. Underfitting occurs for the small_net as the model's errors and loss level seem to fluctuate a lot and therefore shows that it is not a suitable model for both new and traninng data. Overfitting occurs for the large-net and small-net as the validation error eventually diverges dramatically from the train error. This has shown that although the model can fit the training data fairly accurately, it doesn't generalize to new data set (validation data) as the model overly relies on the training data. Overall speaking large_net data fit the model better than the small_net, as it suffers less underfitting problem and the overfitting problem can be reduced later.

Part 3. Optimization Parameters [12 pt]

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

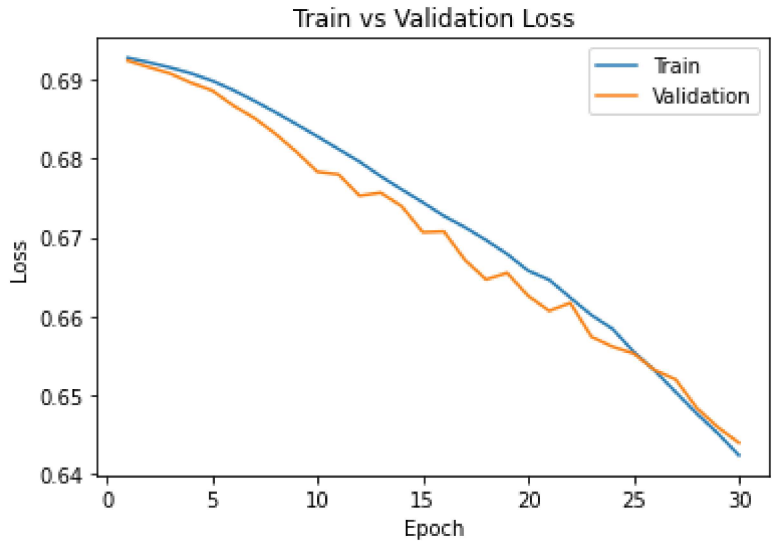
Part (a) - 3pt

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 [45]:
```
# 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,batch_size=64, learning_rate=0.001, num_epochs=30)
model_path_large = get_model_name("large",batch_size=64, learning_rate=0.001,
epoch=29)
print("Large model:")
plot_training_curve(model_path_large)
```

```
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: 0.42
4, 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 er
r: 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 er
r: 0.415, Validation loss: 0.6757059413939714
Epoch 14: Train err: 0.412375, Train loss: 0.6761112003326416 |Validation er
r: 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 er
r: 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: 0.662373058795929 |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 er
r: 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 er
r: 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 er
```

```
r: 0.3875, Validation loss: 0.6483502741903067
Epoch 29: Train err: 0.368375, Train loss: 0.6451257643699646 |Validation er
r: 0.3825, Validation loss: 0.6459067314863205
Epoch 30: Train err: 0.362625, Train loss: 0.6423329524993896 |Validation er
r: 0.3785, Validation loss: 0.6439237017184496
Finished Training
Total time elapsed: 136.30 seconds
Large model:
```



The model took longer time to train (136.3>135.62). Lowering the learning rate increases the time to train due to smaller changes in adjusting the weights in minimizing the loss function. Additionally, when learning rate is lowered, train error and validation error as well as losses deviate from each other much less, and the overfitting and underfitting problem are significantly reduced. However, the average losses and errors are increased at each epoch.

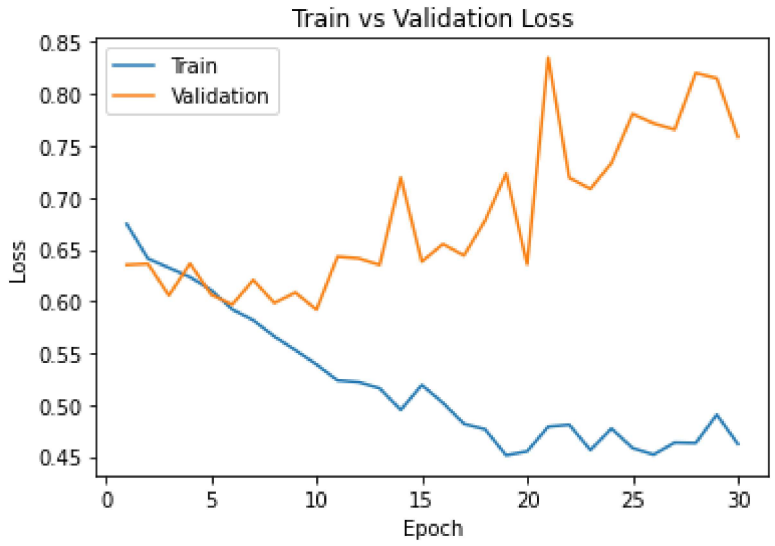
Part (b) - 3pt

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

In [49]:
```
large_net = LargeNet()
train_net(large_net,batch_size=64, learning_rate=0.1, num_epochs=30)
model_path_large = get_model_name("large",batch_size=64, learning_rate=0.1, ep
och=29)
print("Large model:")
plot_training_curve(model_path_large)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.4295, Train loss: 0.67437779712677 |Validation err: 0.3
595, Validation loss: 0.6350857093930244
Epoch 2: Train err: 0.36075, Train loss: 0.6411805458068848 |Validation err:
0.3535, Validation loss: 0.6361209936439991
Epoch 3: Train err: 0.365125, Train loss: 0.6321813461780548 |Validation err:
0.3385, Validation loss: 0.6056603882461786
Epoch 4: Train err: 0.352625, Train loss: 0.6233456182479858 |Validation err:
0.3575, Validation loss: 0.6362800188362598
Epoch 5: Train err: 0.34075, Train loss: 0.6108013873100281 |Validation err:
0.3305, Validation loss: 0.6064918786287308
Epoch 6: Train err: 0.323375, Train loss: 0.5921835997104645 |Validation err:
0.317, Validation loss: 0.5967769594863057
Epoch 7: Train err: 0.3145, Train loss: 0.5817317583560944 |Validation err:
0.3365, Validation loss: 0.6204487886279821
Epoch 8: Train err: 0.29825, Train loss: 0.5660300073623658 |Validation err:
0.3285, Validation loss: 0.5983372200280428
Epoch 9: Train err: 0.290875, Train loss: 0.552809501171112 |Validation err:
0.3315, Validation loss: 0.6084455158561468
Epoch 10: Train err: 0.278625, Train loss: 0.539032607793808 |Validation err:
0.306, Validation loss: 0.5918631898239255
Epoch 11: Train err: 0.272375, Train loss: 0.5236025826931 |Validation err:
0.33, Validation loss: 0.6430060230195522
Epoch 12: Train err: 0.267375, Train loss: 0.5220149435997009 |Validation er
r: 0.2925, Validation loss: 0.6413561534136534
Epoch 13: Train err: 0.266, Train loss: 0.5160510110855102 |Validation err:
0.3125, Validation loss: 0.6349832843989134
Epoch 14: Train err: 0.24875, Train loss: 0.4951590054035187 |Validation err:
0.3145, Validation loss: 0.7193072671070695
Epoch 15: Train err: 0.264625, Train loss: 0.519231944322586 |Validation err:
0.314, Validation loss: 0.6381420725956559
Epoch 16: Train err: 0.252625, Train loss: 0.5020012385845184 |Validation er
r: 0.3225, Validation loss: 0.6551959458738565
Epoch 17: Train err: 0.23875, Train loss: 0.481714787364006 |Validation err:
0.357, Validation loss: 0.6440742611885071
Epoch 18: Train err: 0.23375, Train loss: 0.47645506453514097 |Validation er
r: 0.3375, Validation loss: 0.6777342790737748
Epoch 19: Train err: 0.218125, Train loss: 0.45134368968009947 |Validation er
r: 0.3445, Validation loss: 0.7232250478118658
Epoch 20: Train err: 0.217875, Train loss: 0.45516350817680357 |Validation er
r: 0.3245, Validation loss: 0.6354950983077288
Epoch 21: Train err: 0.23275, Train loss: 0.47897080445289614 |Validation er
r: 0.3255, Validation loss: 0.8348110988736153
Epoch 22: Train err: 0.234875, Train loss: 0.4808810565471649 |Validation er
r: 0.334, Validation loss: 0.7191346418112516
Epoch 23: Train err: 0.21575, Train loss: 0.4563647754192352 |Validation err:
0.316, Validation loss: 0.7083508176729083
Epoch 24: Train err: 0.2355, Train loss: 0.47718250966072084 |Validation err:
0.327, Validation loss: 0.7333047650754452
Epoch 25: Train err: 0.22025, Train loss: 0.4583414270877838 |Validation err:
0.3315, Validation loss: 0.7806987538933754
Epoch 26: Train err: 0.209625, Train loss: 0.4519626965522766 |Validation er
r: 0.3435, Validation loss: 0.7715998776257038
Epoch 27: Train err: 0.22175, Train loss: 0.4636160457134247 |Validation err:
0.3215, Validation loss: 0.7656293725594878
Epoch 28: Train err: 0.219375, Train loss: 0.46314777398109436 |Validation er
```

```
r: 0.348, Validation loss: 0.8202023077756166
Epoch 29: Train err: 0.235875, Train loss: 0.49053542733192446 |Validation er
r: 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: 133.08 seconds
Large model:
```



The model took shorter time to train (133.08<135.62). Increasing the learning rate reduces the time to train due to larger changes in adjusting the weights in minimizing the loss function. Additionally, when learning rate is increased, train error and validation error as well as losses deviate from each other much more and the overfitting problem is worsen. The underfitting problem is more severe as well, and the error and loss graphs are more noisy. The average training error and loss are reduced.

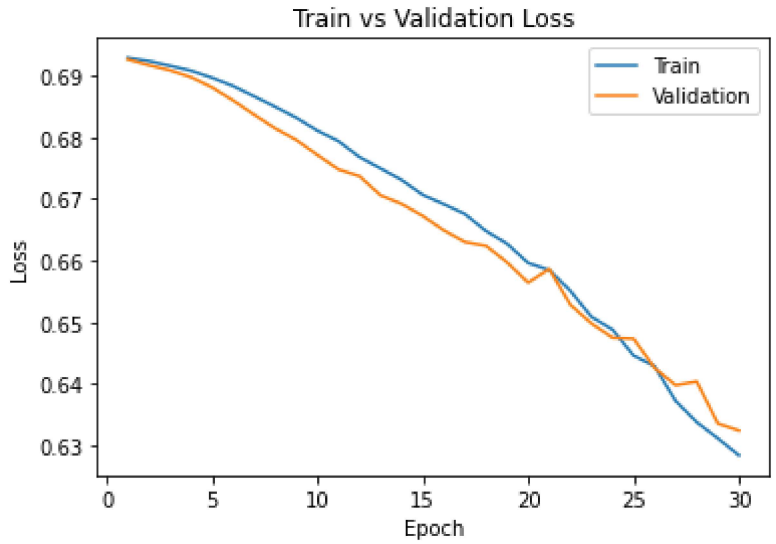
Part (c) - 3pt

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 [47]:
```
large_net = LargeNet()
train_net(large_net,batch_size=512, learning_rate=0.01, num_epochs=30)
model_path_large = get_model_name("large",batch_size=512, learning_rate=0.01,
epoch=29)
print("Large model:")
plot_training_curve(model_path_large)
```

```
Files already downloaded and verified
Files already downloaded and verified
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: 0.6924104019999504 |Validation err:
0.434, Validation loss: 0.6917425245046616
Epoch 3: Train err: 0.437, Train loss: 0.6916500590741634 |Validation err: 0.
4265, Validation loss: 0.6909129917621613
Epoch 4: Train err: 0.433625, Train loss: 0.6908449940383434 |Validation err:
0.424, Validation loss: 0.6897870451211929
Epoch 5: Train err: 0.434, Train loss: 0.6896935552358627 |Validation err: 0.
424, Validation loss: 0.6881355047225952
Epoch 6: Train err: 0.438, Train loss: 0.688353206962347 |Validation err: 0.4
285, Validation loss: 0.686011865735054
Epoch 7: Train err: 0.439375, Train loss: 0.6866871677339077 |Validation err:
0.426, Validation loss: 0.6836968809366226
Epoch 8: Train err: 0.43525, Train loss: 0.6849770769476891 |Validation err:
0.4115, Validation loss: 0.6814671903848648
Epoch 9: Train err: 0.42375, Train loss: 0.6832009293138981 |Validation err:
0.414, Validation loss: 0.679591491818428
Epoch 10: Train err: 0.421, Train loss: 0.6811089366674423 |Validation err:
0.416, Validation loss: 0.6771548539400101
Epoch 11: Train err: 0.420875, Train loss: 0.6794026419520378 |Validation er
r: 0.4095, Validation loss: 0.6748111099004745
Epoch 12: Train err: 0.41475, Train loss: 0.6768048219382763 |Validation err:
0.412, Validation loss: 0.6737060546875
Epoch 13: Train err: 0.4105, Train loss: 0.6749702803790569 |Validation err:
0.412, Validation loss: 0.6706101596355438
Epoch 14: Train err: 0.407125, Train loss: 0.6730880849063396 |Validation er
r: 0.4125, Validation loss: 0.6692148000001907
Epoch 15: Train err: 0.4005, Train loss: 0.6706806942820549 |Validation err:
0.4105, Validation loss: 0.667252704501152
Epoch 16: Train err: 0.397625, Train loss: 0.6691771410405636 |Validation er
r: 0.405, Validation loss: 0.6649097055196762
Epoch 17: Train err: 0.393875, Train loss: 0.6675694733858109 |Validation er
r: 0.401, Validation loss: 0.6630224883556366
Epoch 18: Train err: 0.393, Train loss: 0.6648042872548103 |Validation err:
0.3945, Validation loss: 0.6624014377593994
Epoch 19: Train err: 0.38625, Train loss: 0.662746611982584 |Validation err:
0.388, Validation loss: 0.6597220152616501
Epoch 20: Train err: 0.38175, Train loss: 0.6596181839704514 |Validation err:
0.4005, Validation loss: 0.6564337313175201
Epoch 21: Train err: 0.38575, Train loss: 0.6584899798035622 |Validation err:
0.3885, Validation loss: 0.6586423963308334
Epoch 22: Train err: 0.378125, Train loss: 0.655123382806778 |Validation err:
0.3855, Validation loss: 0.6528600305318832
Epoch 23: Train err: 0.372125, Train loss: 0.6508794128894806 |Validation er
r: 0.3835, Validation loss: 0.6497963815927505
Epoch 24: Train err: 0.37675, Train loss: 0.6488028429448605 |Validation err:
0.385, Validation loss: 0.6474899500608444
Epoch 25: Train err: 0.368625, Train loss: 0.6445869170129299 |Validation er
r: 0.382, Validation loss: 0.6473268568515778
Epoch 26: Train err: 0.372625, Train loss: 0.6428566053509712 |Validation er
r: 0.3745, Validation loss: 0.6425703465938568
Epoch 27: Train err: 0.359375, Train loss: 0.6372117549180984 |Validation er
r: 0.379, Validation loss: 0.6397799849510193
Epoch 28: Train err: 0.35425, Train loss: 0.6337667480111122 |Validation err:
```

```
0.3695, Validation loss: 0.6403783112764359
Epoch 29: Train err: 0.3535, Train loss: 0.6311353109776974 |Validation err:
0.366, Validation loss: 0.6335585117340088
Epoch 30: Train err: 0.353, Train loss: 0.6283832415938377 |Validation err:
0.3675, Validation loss: 0.6324127316474915
Finished Training
Total time elapsed: 115.65 seconds
Large model:
```



The model took shorter time to train (115.65<135.62). Increasing the batch size decreases the time to train because that it now takes in more data in one batch and thus reduces the number of steps, which reduces the time to train. Additionally, when the batch size is increased, train error and validation error as well as losses deviate from each other much less and the overfitting and underfitting problem are significantly reduced. However, the losses and errors increase in general compared to before.

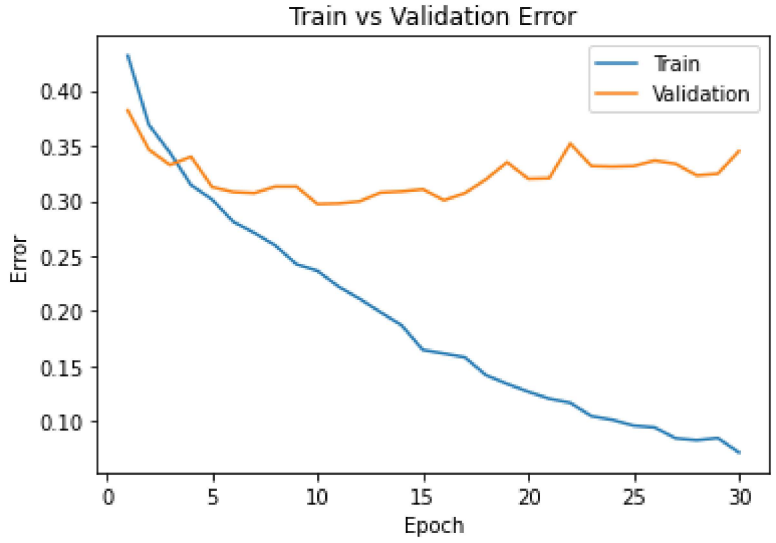
Part (d) - 3pt

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 [48]:
```
large_net = LargeNet()
train_net(large_net,batch_size=16, learning_rate=0.01, num_epochs=30)
model_path_large = get_model_name("large",batch_size=16, learning_rate=0.01, e
poch=29)
print("Large model:")
plot_training_curve(model_path_large)
```

```
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: 0.3
465, Validation loss: 0.6161113576889038
Epoch 3: Train err: 0.34375, Train loss: 0.6098222947120666 |Validation err:
0.3325, Validation loss: 0.6260210764408112
Epoch 4: Train err: 0.314375, Train loss: 0.5849691489338875 |Validation err:
0.34, Validation loss: 0.6044013917446136
Epoch 5: Train err: 0.301125, Train loss: 0.5689119303822517 |Validation err:
0.3125, Validation loss: 0.576918310880661
Epoch 6: Train err: 0.281, Train loss: 0.5452213581204415 |Validation err: 0.
308, Validation loss: 0.5708447456359863
Epoch 7: Train err: 0.270875, Train loss: 0.5272981298565864 |Validation err:
0.307, Validation loss: 0.5854293291568756
Epoch 8: Train err: 0.259375, Train loss: 0.5070905526578426 |Validation err:
0.313, Validation loss: 0.5877130818367005
Epoch 9: Train err: 0.242375, Train loss: 0.4968344421982765 |Validation err:
0.313, Validation loss: 0.5922425072193146
Epoch 10: Train err: 0.236375, Train loss: 0.4756101597249508 |Validation er
r: 0.297, Validation loss: 0.5718690166473389
Epoch 11: Train err: 0.222125, Train loss: 0.4599769461452961 |Validation er
r: 0.2975, Validation loss: 0.6376970833539963
Epoch 12: Train err: 0.211, Train loss: 0.4454492371380329 |Validation err:
0.2995, Validation loss: 0.609202565908432
Epoch 13: Train err: 0.19875, Train loss: 0.4245421719551086 |Validation err:
0.3075, Validation loss: 0.6494987765550614
Epoch 14: Train err: 0.18675, Train loss: 0.4007472907453775 |Validation err:
0.3085, Validation loss: 0.6610016552209854
Epoch 15: Train err: 0.1645, Train loss: 0.3759974058121443 |Validation err:
0.3105, Validation loss: 0.7106090537309646
Epoch 16: Train err: 0.16125, Train loss: 0.3591455406397581 |Validation err:
0.3005, Validation loss: 0.7310364942550659
Epoch 17: Train err: 0.15775, Train loss: 0.3463234790861607 |Validation err:
0.307, Validation loss: 0.7263009325265884
Epoch 18: Train err: 0.141625, Train loss: 0.32175366275012496 |Validation er
r: 0.3195, Validation loss: 0.7913952842950821
Epoch 19: Train err: 0.13375, Train loss: 0.30618105667084455 |Validation er
r: 0.335, Validation loss: 0.8032052783966065
Epoch 20: Train err: 0.126625, Train loss: 0.3029071792438626 |Validation er
r: 0.32, Validation loss: 0.8106685240268707
Epoch 21: Train err: 0.12025, Train loss: 0.28682796490937473 |Validation er
r: 0.3205, Validation loss: 0.8259474284648896
Epoch 22: Train err: 0.1165, Train loss: 0.27489088076353074 |Validation err:
0.352, Validation loss: 0.8937610774040222
Epoch 23: Train err: 0.104375, Train loss: 0.2467898527495563 |Validation er
r: 0.3315, Validation loss: 1.0021928198337555
Epoch 24: Train err: 0.101, Train loss: 0.23970085787773132 |Validation err:
0.331, Validation loss: 1.1290796399116516
Epoch 25: Train err: 0.09575, Train loss: 0.23643119425699116 |Validation er
r: 0.3315, Validation loss: 1.1338514368534087
Epoch 26: Train err: 0.094125, Train loss: 0.2325953512713313 |Validation er
r: 0.3365, Validation loss: 1.1414263204336166
Epoch 27: Train err: 0.08425, Train loss: 0.21040759468451142 |Validation er
r: 0.3335, Validation loss: 1.1823678107261657
Epoch 28: Train err: 0.0825, Train loss: 0.20643112615589052 |Validation err:
```

```
0.323, Validation loss: 1.266836181640625
Epoch 29: Train err: 0.0845, Train loss: 0.21273409337876364 |Validation err:
0.3245, Validation loss: 1.406717705130577
Epoch 30: Train err: 0.071375, Train loss: 0.18387044295761734 |Validation er
r: 0.345, Validation loss: 1.4871552000045776
Finished Training
Total time elapsed: 195.74 seconds
Large model:
```



The model took longer time to train (195.74<135.62). Reducing the batch size increases the time to train because that it now takes in fewer data in one batch and thus increases the number of steps, which increases the time to train. Additionally, when batch size is decreased, train error and validation error as well as losses deviate from each other much more and the overfitting problem is worsen. The validation loss is greatly increased in general compared to before, whereas the train loss and error decrease significantly than before.

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.

My chosen set of values for the hyperparameters are batch_size = 32, learning rate = 0.001,number of epochs = 30. Although reducing the batch size will worsen the overfitting problem, it will decreases the train error and loss significantly. Although reducing the learning rate will increase the training time, I think it will decrease the overfitting problem so that the validation error and loss will be small and similar to the train error and loss.

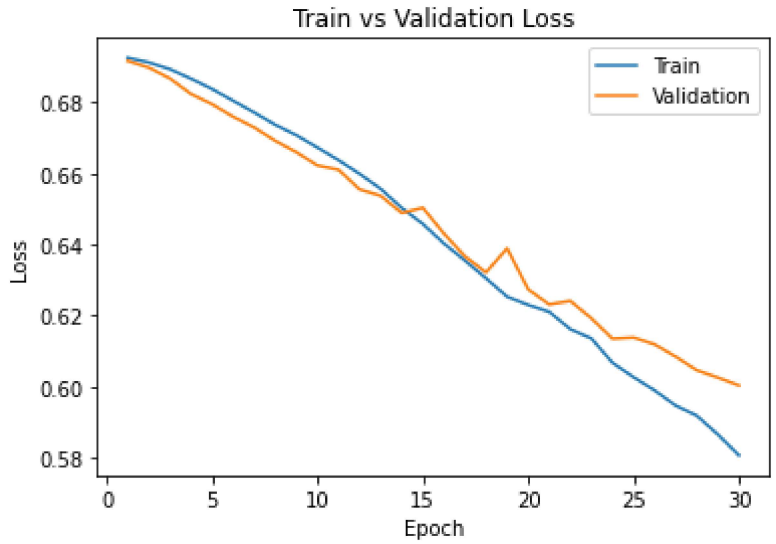
Part (b) - 1pt

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

In [8]:
```
large_net = LargeNet()
train_net(large_net,batch_size=32, learning_rate=0.001, num_epochs=30)
model_path_large = get_model_name("large",batch_size=32, learning_rate=0.001,
epoch=29)
print("Large model:")
plot_training_curve(model_path_large)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.466125, Train loss: 0.6925730059146881 |Validation err:
0.4335, Validation loss: 0.6916963704048641
Epoch 2: Train err: 0.447, Train loss: 0.6912713117599487 |Validation err: 0.
4235, Validation loss: 0.689815480557699
Epoch 3: Train err: 0.43475, Train loss: 0.6893711988925934 |Validation err:
0.418, Validation loss: 0.6867533401837425
Epoch 4: Train err: 0.433, Train loss: 0.6866874279975891 |Validation err: 0.
42, Validation loss: 0.6824120084444681
Epoch 5: Train err: 0.425875, Train loss: 0.6837984304428101 |Validation err:
0.413, Validation loss: 0.6795223591819642
Epoch 6: Train err: 0.41875, Train loss: 0.6804884467124939 |Validation err:
0.4115, Validation loss: 0.6759844375035119
Epoch 7: Train err: 0.412125, Train loss: 0.6771827213764191 |Validation err:
0.4145, Validation loss: 0.6729671150918991
Epoch 8: Train err: 0.409125, Train loss: 0.6737104182243348 |Validation err:
0.41, Validation loss: 0.6691933368879651
Epoch 9: Train err: 0.40275, Train loss: 0.670788322687149 |Validation err:
0.407, Validation loss: 0.6659858311925616
Epoch 10: Train err: 0.39725, Train loss: 0.6673055355548858 |Validation err:
0.4025, Validation loss: 0.6622668078967503
Epoch 11: Train err: 0.392125, Train loss: 0.6637537758350373 |Validation er
r: 0.393, Validation loss: 0.661065537778158
Epoch 12: Train err: 0.3855, Train loss: 0.6599007363319397 |Validation err:
0.3915, Validation loss: 0.6555445279393878
Epoch 13: Train err: 0.380625, Train loss: 0.6557163195610046 |Validation er
r: 0.385, Validation loss: 0.6537485851181878
Epoch 14: Train err: 0.37125, Train loss: 0.6503091659545899 |Validation err:
0.3835, Validation loss: 0.6488832746233258
Epoch 15: Train err: 0.375, Train loss: 0.6458070278167725 |Validation err:
0.3835, Validation loss: 0.6503617924357218
Epoch 16: Train err: 0.365875, Train loss: 0.64028985953331 |Validation err:
0.3825, Validation loss: 0.6430936285427639
Epoch 17: Train err: 0.359, Train loss: 0.6354842091798782 |Validation err:
0.3705, Validation loss: 0.6366607797524285
Epoch 18: Train err: 0.358125, Train loss: 0.6304897521734237 |Validation er
r: 0.3635, Validation loss: 0.6321560589094011
Epoch 19: Train err: 0.349875, Train loss: 0.6253278095722199 |Validation er
r: 0.37, Validation loss: 0.6389134658707513
Epoch 20: Train err: 0.3485, Train loss: 0.6230042055845261 |Validation err:
0.3585, Validation loss: 0.6274004559668284
Epoch 21: Train err: 0.34275, Train loss: 0.6210563819408417 |Validation err:
0.349, Validation loss: 0.6231400597663153
Epoch 22: Train err: 0.343, Train loss: 0.6161815811395646 |Validation err:
0.3455, Validation loss: 0.6241406060400463
Epoch 23: Train err: 0.34, Train loss: 0.6135540924072266 |Validation err: 0.
339, Validation loss: 0.619217766655816
Epoch 24: Train err: 0.33725, Train loss: 0.6067150747776031 |Validation err:
0.3385, Validation loss: 0.6134822652453468
Epoch 25: Train err: 0.334125, Train loss: 0.6026661356687546 |Validation er
r: 0.3455, Validation loss: 0.6138216488891177
Epoch 26: Train err: 0.325875, Train loss: 0.5989329878091812 |Validation er
r: 0.3345, Validation loss: 0.611895415518019
Epoch 27: Train err: 0.3275, Train loss: 0.5946586643457412 |Validation err:
0.3205, Validation loss: 0.6084380878342522
Epoch 28: Train err: 0.322375, Train loss: 0.5918326848745346 |Validation er
```

```
r: 0.3185, Validation loss: 0.6046140610225617
Epoch 29: Train err: 0.314625, Train loss: 0.5865380837917328 |Validation er
r: 0.32, Validation loss: 0.6025549972814227
Epoch 30: Train err: 0.309, Train loss: 0.5806492085456848 |Validation err:
0.321, Validation loss: 0.6003361268649026
Finished Training
Total time elapsed: 148.43 seconds
Large model:
```



Part (c) - 2pt

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

The results from part a) are pretty promising, as the overfitting problem is controlled and the error is relatively low. However, I think the error can be further reduced with reduced batch size. Reducing the learning rate can futher reduce the overfitting problem.

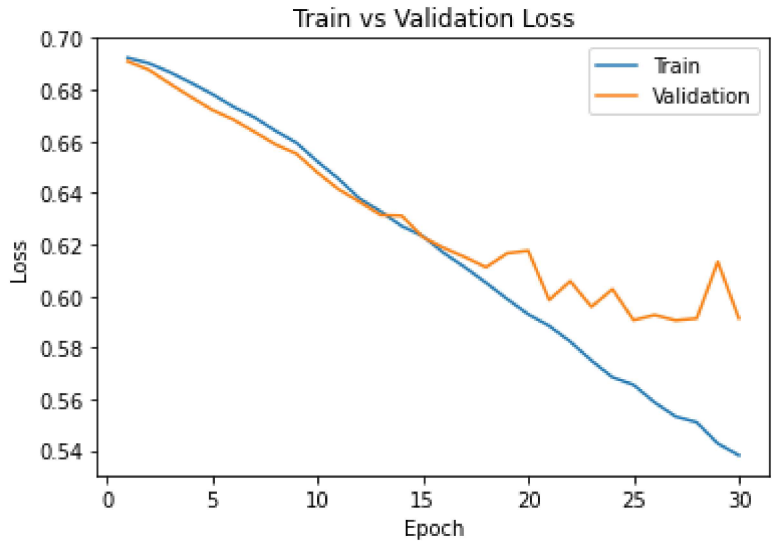
Part (d) - 1pt

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

In [9]:
```
large_net = LargeNet()
train_net(large_net,batch_size=16, learning_rate=0.0007, num_epochs=30)
model_path_large = get_model_name("large",batch_size=16, learning_rate=0.0007,
epoch=29)
print("Large model:")
plot_training_curve(model_path_large)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.45775, Train loss: 0.6923313375711441 |Validation err:
0.429, Validation loss: 0.6909683723449707
Epoch 2: Train err: 0.445125, Train loss: 0.6902284998893737 |Validation err:
0.4165, Validation loss: 0.68767560338974
Epoch 3: Train err: 0.431375, Train loss: 0.6867488778829575 |Validation err:
0.412, Validation loss: 0.6822789363861084
Epoch 4: Train err: 0.422875, Train loss: 0.6826530860662461 |Validation err:
0.4105, Validation loss: 0.6771459794044494
Epoch 5: Train err: 0.41925, Train loss: 0.6783087722063065 |Validation err:
0.411, Validation loss: 0.6721762719154358
Epoch 6: Train err: 0.405875, Train loss: 0.6734160768985749 |Validation err:
0.4105, Validation loss: 0.6684141178131103
Epoch 7: Train err: 0.397625, Train loss: 0.6692936973571777 |Validation err:
0.3995, Validation loss: 0.6636849761009216
Epoch 8: Train err: 0.393375, Train loss: 0.6640850905179977 |Validation err:
0.399, Validation loss: 0.6588062953948974
Epoch 9: Train err: 0.385, Train loss: 0.6594116897583008 |Validation err: 0.
3915, Validation loss: 0.6551446042060852
Epoch 10: Train err: 0.37625, Train loss: 0.6521287958025932 |Validation err:
0.3875, Validation loss: 0.6478536672592163
Epoch 11: Train err: 0.362375, Train loss: 0.6454345623850822 |Validation er
r: 0.379, Validation loss: 0.6413924884796143
Epoch 12: Train err: 0.357125, Train loss: 0.6377665683627128 |Validation er
r: 0.3625, Validation loss: 0.6365272104740143
Epoch 13: Train err: 0.357375, Train loss: 0.6328267571926117 |Validation er
r: 0.3545, Validation loss: 0.6313904032707215
Epoch 14: Train err: 0.352375, Train loss: 0.6270976803302765 |Validation er
r: 0.359, Validation loss: 0.6311080799102783
Epoch 15: Train err: 0.3465, Train loss: 0.6232257344722748 |Validation err:
0.3455, Validation loss: 0.6228464634418488
Epoch 16: Train err: 0.347125, Train loss: 0.6166393537521362 |Validation er
r: 0.34, Validation loss: 0.6186069421768189
Epoch 17: Train err: 0.336875, Train loss: 0.611111298263073 |Validation err:
0.335, Validation loss: 0.615061963558197
Epoch 18: Train err: 0.336125, Train loss: 0.6051417922377587 |Validation er
r: 0.329, Validation loss: 0.6111300423145294
Epoch 19: Train err: 0.326625, Train loss: 0.5988901138901711 |Validation er
r: 0.3375, Validation loss: 0.6165456101894379
Epoch 20: Train err: 0.321875, Train loss: 0.5929287813305855 |Validation er
r: 0.3415, Validation loss: 0.6175643079280854
Epoch 21: Train err: 0.317, Train loss: 0.5884002556204796 |Validation err:
0.3185, Validation loss: 0.5984949963092804
Epoch 22: Train err: 0.314125, Train loss: 0.5823386007547379 |Validation er
r: 0.3305, Validation loss: 0.6057743673324585
Epoch 23: Train err: 0.309625, Train loss: 0.5749758765101433 |Validation er
r: 0.3115, Validation loss: 0.5958208031654358
Epoch 24: Train err: 0.296625, Train loss: 0.5685029901266098 |Validation er
r: 0.3215, Validation loss: 0.6026424686908722
Epoch 25: Train err: 0.297125, Train loss: 0.5655799903273583 |Validation er
r: 0.313, Validation loss: 0.5906458003520966
Epoch 26: Train err: 0.289375, Train loss: 0.5587581843137741 |Validation er
r: 0.3135, Validation loss: 0.592645069360733
Epoch 27: Train err: 0.286375, Train loss: 0.5532734594345092 |Validation er
r: 0.307, Validation loss: 0.5905520958900452
Epoch 28: Train err: 0.287, Train loss: 0.5510365852117538 |Validation err:
```

```
0.3195, Validation loss: 0.5913199999332428
Epoch 29: Train err: 0.28025, Train loss: 0.542787800848484 |Validation err:
0.3195, Validation loss: 0.6132768330574035
Epoch 30: Train err: 0.2765, Train loss: 0.5382332564890384 |Validation err:
0.314, Validation loss: 0.5913634011745453
Finished Training
Total time elapsed: 182.54 seconds
Large model:
```



Part 4. Evaluating the Best Model [15 pt]

Part (a) - 1pt

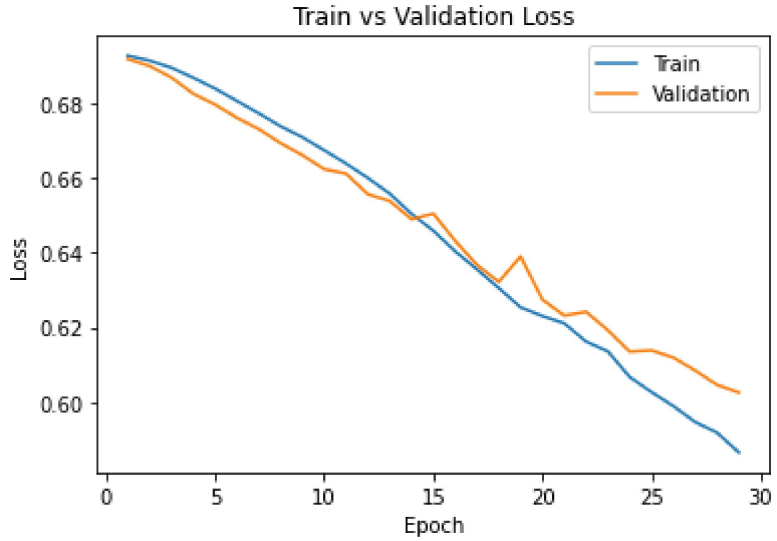
Choose the **best** model that you have so far. This means choosing the best model checkpoint, including the choice of `small_net` vs `large_net`, the `batch_size`, `learning_rate`, **and the epoch number**.

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

In [21]:
```
large_net = LargeNet()
train_net(large_net,batch_size=32, learning_rate=0.001, num_epochs=29)
model_path = get_model_name("large", batch_size=32, learning_rate=0.001, epoch
=28)
plot_training_curve(model_path)
state = torch.load(model_path)
large_net.load_state_dict(state)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.466125, Train loss: 0.6925730059146881 |Validation err:
0.4335, Validation loss: 0.6916963704048641
Epoch 2: Train err: 0.447, Train loss: 0.6912713117599487 |Validation err: 0.
4235, Validation loss: 0.689815480557699
Epoch 3: Train err: 0.43475, Train loss: 0.6893711988925934 |Validation err:
0.418, Validation loss: 0.6867533401837425
Epoch 4: Train err: 0.433, Train loss: 0.6866874279975891 |Validation err: 0.
42, Validation loss: 0.6824120084444681
Epoch 5: Train err: 0.425875, Train loss: 0.6837984304428101 |Validation err:
0.413, Validation loss: 0.6795223591819642
Epoch 6: Train err: 0.41875, Train loss: 0.6804884467124939 |Validation err:
0.4115, Validation loss: 0.6759844375035119
Epoch 7: Train err: 0.412125, Train loss: 0.6771827213764191 |Validation err:
0.4145, Validation loss: 0.6729671150918991
Epoch 8: Train err: 0.409125, Train loss: 0.6737104182243348 |Validation err:
0.41, Validation loss: 0.6691933368879651
Epoch 9: Train err: 0.40275, Train loss: 0.670788322687149 |Validation err:
0.407, Validation loss: 0.6659858311925616
Epoch 10: Train err: 0.39725, Train loss: 0.6673055355548858 |Validation err:
0.4025, Validation loss: 0.6622668078967503
Epoch 11: Train err: 0.392125, Train loss: 0.6637537758350373 |Validation er
r: 0.393, Validation loss: 0.661065537778158
Epoch 12: Train err: 0.3855, Train loss: 0.6599007363319397 |Validation err:
0.3915, Validation loss: 0.6555445279393878
Epoch 13: Train err: 0.380625, Train loss: 0.6557163195610046 |Validation er
r: 0.385, Validation loss: 0.6537485851181878
Epoch 14: Train err: 0.37125, Train loss: 0.6503091659545899 |Validation err:
0.3835, Validation loss: 0.6488832746233258
Epoch 15: Train err: 0.375, Train loss: 0.6458070278167725 |Validation err:
0.3835, Validation loss: 0.6503617924357218
Epoch 16: Train err: 0.365875, Train loss: 0.64028985953331 |Validation err:
0.3825, Validation loss: 0.6430936285427639
Epoch 17: Train err: 0.359, Train loss: 0.6354842091798782 |Validation err:
0.3705, Validation loss: 0.6366607797524285
Epoch 18: Train err: 0.358125, Train loss: 0.6304897521734237 |Validation er
r: 0.3635, Validation loss: 0.6321560589094011
Epoch 19: Train err: 0.349875, Train loss: 0.6253278095722199 |Validation er
r: 0.37, Validation loss: 0.6389134658707513
Epoch 20: Train err: 0.3485, Train loss: 0.6230042055845261 |Validation err:
0.3585, Validation loss: 0.6274004559668284
Epoch 21: Train err: 0.34275, Train loss: 0.6210563819408417 |Validation err:
0.349, Validation loss: 0.6231400597663153
Epoch 22: Train err: 0.343, Train loss: 0.6161815811395646 |Validation err:
0.3455, Validation loss: 0.6241406060400463
Epoch 23: Train err: 0.34, Train loss: 0.6135540924072266 |Validation err: 0.
339, Validation loss: 0.619217766655816
Epoch 24: Train err: 0.33725, Train loss: 0.6067150747776031 |Validation err:
0.3385, Validation loss: 0.6134822652453468
Epoch 25: Train err: 0.334125, Train loss: 0.6026661356687546 |Validation er
r: 0.3455, Validation loss: 0.6138216488891177
Epoch 26: Train err: 0.325875, Train loss: 0.5989329878091812 |Validation er
r: 0.3345, Validation loss: 0.611895415518019
Epoch 27: Train err: 0.3275, Train loss: 0.5946586643457412 |Validation err:
0.3205, Validation loss: 0.6084380878342522
Epoch 28: Train err: 0.322375, Train loss: 0.5918326848745346 |Validation er
```

```
r: 0.3185, Validation loss: 0.6046140610225617
Epoch 29: Train err: 0.314625, Train loss: 0.5865380837917328 |Validation er
r: 0.32, Validation loss: 0.6025549972814227
Finished Training
Total time elapsed: 141.56 seconds
```



Out[21]: <All keys matched successfully>

Part (b) - 2pt

Justify your choice of model from part (a).

The same parameters from 4b were used in part a), with large_net, the batch_size = 32, learning_rate = 0.001, and the epoch number = 29. The validation error decreases significantly until the 28th epoch. Therefore, the epoch number is adjusted slight from 30 to 29.

In this choice of model, the overfitting problem is reduced significantly compared to other models. The validation error is also reduced to 0.32 in the final epoch, relative low compared to other models. The choice from 4d was not chosen because it has severe overfitting problem compared to 4b.

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 [22]:
```
# 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)
err, loss = evaluate(large_net, test_loader, nn.BCEWithLogitsLoss())
print(err)
```

```
Files already downloaded and verified
Files already downloaded and verified
0.332
```

Part (d) - 3pt

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

The test classification error is 0.332 which is higher than the validation error which is 0.32 from part a after 29 epochs. The test error is expected to be higher due to that the test data is the brand new data to the data that the model has never trained on before. As we know that the model is tuned based on the validation data and the train data, therefore the model is expected to be more accurate on the validation data than the general new data that the model has no previous experience.

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?

We only use the test data set at the end because we want to see how realistic our model can perform on a new data set. The test data set must be new to our neural network and therefore is only applied at the very end after we finish training our model. The test data should be used as little as possible because if the test data is used in training the model, the model will have bias towards the particular test data. As a result, the test data will lose its generality and cannot be used to test the performance of the model on new and general data, because the model will remember and have bias in the used test data.

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 [23]:
```
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
import numpy as np

torch.manual_seed(1) # set the random seed

# define a 2-layer artificial neural network
class cat_dog(nn.Module):
    def __init__(self):
        super(cat_dog, self).__init__()
        self.name = 'cat_dog'
        self.layer1 = nn.Linear(32 * 32, 30)
        self.layer2 = nn.Linear(30, 1)


    def forward(self, img):
        flattened = img.view(-1, 32 * 32)
        activation1 = self.layer1(flattened)
        activation1 = F.relu(activation1)
        activation2 = self.layer2(activation1)

        return activation2.squeeze(1)

cat_dog = cat_dog()
train_net(cat_dog,batch_size=32, learning_rate=0.001, num_epochs=29)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.434875, Train loss: 0.6799167640209198 |Validation err:
0.416, Validation loss: 0.6717010679699126
Epoch 2: Train err: 0.405875, Train loss: 0.6659876420497894 |Validation err:
0.4065, Validation loss: 0.6640299331574213
Epoch 3: Train err: 0.39675, Train loss: 0.6592102403640747 |Validation err:
0.3995, Validation loss: 0.6597571212147909
Epoch 4: Train err: 0.3935, Train loss: 0.653987648487091 |Validation err: 0.
398, Validation loss: 0.6574626527135334
Epoch 5: Train err: 0.385, Train loss: 0.6498583860397339 |Validation err: 0.
391, Validation loss: 0.6544982516576373
Epoch 6: Train err: 0.377, Train loss: 0.6453544647693634 |Validation err: 0.
393, Validation loss: 0.6538092550777254
Epoch 7: Train err: 0.374, Train loss: 0.6419072413444519 |Validation err: 0.
397, Validation loss: 0.6521866983837552
Epoch 8: Train err: 0.369875, Train loss: 0.638644228219986 |Validation err:
0.3945, Validation loss: 0.6506842412645855
Epoch 9: Train err: 0.36925, Train loss: 0.6357664341926574 |Validation err:
0.399, Validation loss: 0.6517505447069804
Epoch 10: Train err: 0.36425, Train loss: 0.6325631721019744 |Validation err:
0.391, Validation loss: 0.6479886581027319
Epoch 11: Train err: 0.35925, Train loss: 0.6299666080474854 |Validation err:
0.3955, Validation loss: 0.6496043290410723
Epoch 12: Train err: 0.354625, Train loss: 0.627101879119873 |Validation err:
0.395, Validation loss: 0.6510035622687567
Epoch 13: Train err: 0.350625, Train loss: 0.6244955954551696 |Validation er
r: 0.3975, Validation loss: 0.6508948840792217
Epoch 14: Train err: 0.35125, Train loss: 0.6220047996044159 |Validation err:
0.4005, Validation loss: 0.652888676476857
Epoch 15: Train err: 0.345625, Train loss: 0.6193871252536773 |Validation er
r: 0.3925, Validation loss: 0.6489737847494701
Epoch 16: Train err: 0.342625, Train loss: 0.6163706480264664 |Validation er
r: 0.3945, Validation loss: 0.6487545068301852
Epoch 17: Train err: 0.338625, Train loss: 0.6138603093624115 |Validation er
r: 0.389, Validation loss: 0.6478309631347656
Epoch 18: Train err: 0.336625, Train loss: 0.6112860950231552 |Validation er
r: 0.3935, Validation loss: 0.6466395845488896
Epoch 19: Train err: 0.334, Train loss: 0.6079571443796158 |Validation err:
0.386, Validation loss: 0.6495023388711233
Epoch 20: Train err: 0.330125, Train loss: 0.6052539476156235 |Validation er
r: 0.3935, Validation loss: 0.6476583395685468
Epoch 21: Train err: 0.323, Train loss: 0.6020453892946244 |Validation err:
0.3915, Validation loss: 0.6478906557673499
Epoch 22: Train err: 0.322375, Train loss: 0.5990860463380814 |Validation er
r: 0.383, Validation loss: 0.6497016052405039
Epoch 23: Train err: 0.321, Train loss: 0.5955845173597336 |Validation err:
0.389, Validation loss: 0.6476688829679338
Epoch 24: Train err: 0.317375, Train loss: 0.5925408774614334 |Validation er
r: 0.386, Validation loss: 0.6483988648369199
Epoch 25: Train err: 0.31275, Train loss: 0.5891513917446136 |Validation err:
0.394, Validation loss: 0.6474269051400442
Epoch 26: Train err: 0.306125, Train loss: 0.5857796405553818 |Validation er
r: 0.3885, Validation loss: 0.6481502756239876
Epoch 27: Train err: 0.310375, Train loss: 0.5832679276466369 |Validation er
r: 0.387, Validation loss: 0.6520891889693246
Epoch 28: Train err: 0.30825, Train loss: 0.5793453329801559 |Validation err:
```

```
0.388, Validation loss: 0.6531033005033221
Epoch 29: Train err: 0.30225, Train loss: 0.5751604465246201 |Validation err:
0.385, Validation loss: 0.6583498640665932
Finished Training
Total time elapsed: 110.87 seconds
```

In order to convert and concatenate the RGB images to grayscale images before feeding them to ANN, I modified the get_data_loader function and specifically modified the " transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),transforms.Grayscale(num_output_channels=1)])". to ensure that the processed images are all grayscale images.

I have tuned the parameters several times. I eventually determined to use the same hyperparameter settings as used in my CNN network. The validation errors are shown above. The validation error of the ANN network eventually reaches a minimum of 0.385. I will use this model to evaluate the ANN using the test data. CNN is expected to perform better than the ANN, since the ANN only has two layers of neurons with only 30 hidden units.

In [24]:
```
train_loader, val_loader, test_loader, classes = get_data_loader(
    target_classes=["cat", "dog"],
    batch_size=64)
err, loss = evaluate(cat_dog, test_loader, nn.BCEWithLogitsLoss())
print(err)
```

```
Files already downloaded and verified
Files already downloaded and verified
0.375
```

The test data classification error results have shown that, with the same hyperparameters, the test error of the ANN is 0.375, which is higher than the CNN (0.375>0.332). Therefore, it is confirmed that CNN works better in classifying our dog and cat images.