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.**

In [40]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```

```

Colab Link

Include a link to your colab file here

Colab Link: https://drive.google.com/file/d/1F4ZQQ29SGgB1YZLfQ_-efaJUtpwqSBOi/view?usp=sharing (https://drive.google.com/file/d/1F4ZQQ29SGgB1YZLfQ_-efaJUtpwqSBOi/view?usp=sharing)

In [4]:
```
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 [5]:

```
##############################################################################
# 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 (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


Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Part (a) -- 1 pt

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

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

In [4]:
```
import matplotlib.pyplot as plt

k = 0
for images, labels in train_loader:
    # since batch_size = 1, there is only 1 image in `images`
    image = images[0]
    # place the colour channel at the end, instead of at the beginning
    img = np.transpose(image, [1,2,0])
    # normalize pixel intensity values to [0, 1]
    img = img / 2 + 0.5
    plt.subplot(3, 5, k+1)
    plt.axis('off')
    plt.imshow(img)

    k += 1
    if k > 14:
        break
```

Part (b) -- 3 pt

How many training examples do we have for the combined `cat` and `dog` classes? What about validation examples? What about test examples?

In [9]:
```
training_examples = len(train_loader)
training_examples = "The number of training examples is {}".format(training_examples)
print(training_examples)
validation_examples = len(val_loader)
validation_examples = "The number of validation examples is {}".format(validation_exa
mples)
print(validation_examples)
test_examples = len(test_loader)
test_examples = "The number of test examples is {}".format(test_examples)
print(test_examples)
```

```
The number of training examples is 8000
The number of validation examples is 2000
The number of test examples is 2000
```

Part (c) -- 3pt

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

Validation Set is used to track the validation accuracy in the training curve and it is used to make decisions about model architecture (i.e hyperparameters). This dataset is important because if we do not have one we would track test loss/accuracy in our training curve and we may make decisions about the ANN architecture using the test accuracy. This is not good because the final test accuracy will not be a realistic estimate of how our model will perform on a new data set.

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.

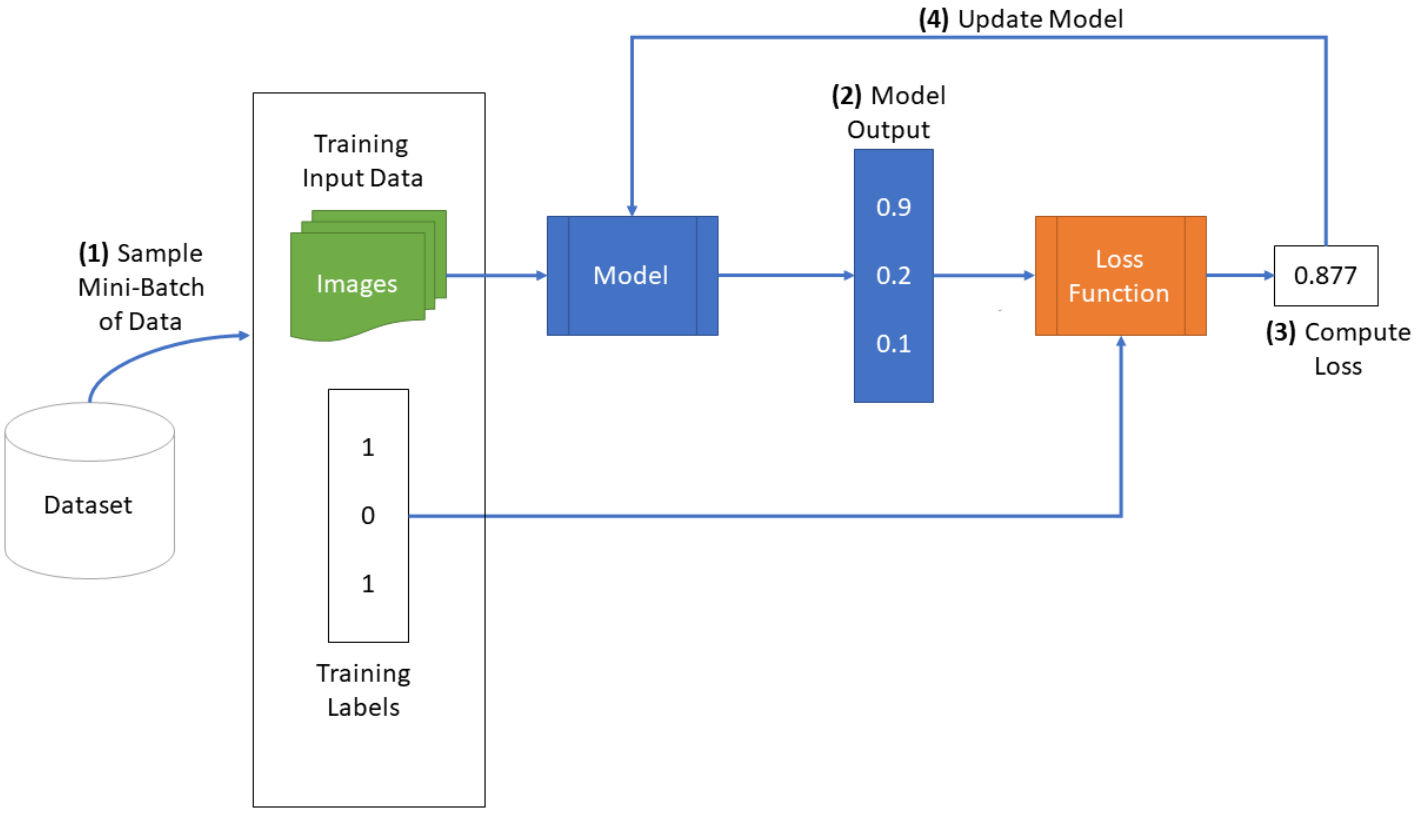
What is the total number of parameters in `small_net` and in `large_net` ? (Hint: how many numbers are in each tensor?)

In [9]:
```
small = 0
large = 0

for param in small_net.parameters():
    small = small + param.numel()
result_small = "The total number of parameters in the SMALL net is {}".format(small)
print(result_small)

for param in large_net.parameters():
    large = large + param.numel()
result_large = "The total number of parameters in the LARGE net is {}".format(large)
print(result_large)
```

```
The total number of parameters in the SMALL net is 386
The total number of parameters in the LARGE net is 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 [11]:
```
current_numbers = "batch_size=64, learning_rate=0.01, num_epochs=30"
print(current_numbers)
```

```
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.

Disk files: model_small_bs64_lr0.01_epoch0, model_small_bs64_lr0.01_epoch1, model_small_bs64_lr0.01_epoch2, model_small_bs64_lr0.01_epoch3, model_small_bs64_lr0.01_epoch4

Each of these files saves the current version of the ANN that is developed in each epoch. Since we have 5 epochs we have 5 different files.

CSV Files: model_small_bs64_lr0.01_epoch4_train_err.csv, model_small_bs64_lr0.01_epoch4_train_loss.csv, model_small_bs64_lr0.01_epoch4_val_err.csv, model_small_bs64_lr0.01_epoch4_val_loss.csv

These files contain the validation and train error/loss data that is generated when we call and train for 5 times smallNnet

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 [13]:
```
# 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 [14]:

```
import time
# Training small net and recording time
start_time_small = time.time()
training_small_net = train_net(small_net, batch_size=64, learning_rate=0.01, num_epoc
hs=30)
end_time_small = time.time()
diff_small = end_time_small - start_time_small

# Training large net and recording time
start_time_large = time.time()
training_large_net = train_net(large_net, batch_size=64, learning_rate=0.01, num_epoc
hs=30)
end_time_large = time.time()
diff_large = end_time_large - start_time_large

# Printing results
time_small = "Total time elapsed for training SMALL net: {}".format(diff_small)
print(time_small)
time_large = "Total time elapsed for training LARGE net: {}".format(diff_large)
print(time_large)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
```

```
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
Epoch 1: Train err: 0.4285, Train loss: 0.6737444496154785 |Validation err: 0.394, V
alidation loss: 0.661311786621809
Epoch 2: Train err: 0.375625, Train loss: 0.6509897251129151 |Validation err: 0.397
5, Validation loss: 0.667868260294199
Epoch 3: Train err: 0.358375, Train loss: 0.6383585271835327 |Validation err: 0.356
5, Validation loss: 0.6331131644546986
Epoch 4: Train err: 0.34325, Train loss: 0.6242320523262024 |Validation err: 0.3625,
Validation loss: 0.6297156233340502
Epoch 5: Train err: 0.33875, Train loss: 0.6175081329345703 |Validation err: 0.3405,
Validation loss: 0.6229404639452696
Epoch 6: Train err: 0.32875, Train loss: 0.60698197722435 |Validation err: 0.3315, V
alidation loss: 0.6183240935206413
Epoch 7: Train err: 0.329375, Train loss: 0.5999680271148682 |Validation err: 0.336,
Validation loss: 0.6100611127912998
Epoch 8: Train err: 0.315875, Train loss: 0.5909323053359985 |Validation err: 0.322,
Validation loss: 0.6025840975344181
Epoch 9: Train err: 0.3105, Train loss: 0.5849576377868653 |Validation err: 0.315, V
alidation loss: 0.5990326162427664
Epoch 10: Train err: 0.3005, Train loss: 0.5755252487659455 |Validation err: 0.309,
Validation loss: 0.590354185551405
Epoch 11: Train err: 0.298125, Train loss: 0.5702860262393952 |Validation err: 0.31
4, Validation loss: 0.5894987024366856
Epoch 12: Train err: 0.294375, Train loss: 0.5643640620708466 |Validation err: 0.32
3, Validation loss: 0.5901558063924313
Epoch 13: Train err: 0.294, Train loss: 0.5658284819126129 |Validation err: 0.3115,
Validation loss: 0.5870114732533693
Epoch 14: Train err: 0.28625, Train loss: 0.5587620766162872 |Validation err: 0.32,
Validation loss: 0.595278830267489
Epoch 15: Train err: 0.288, Train loss: 0.5557496762275695 |Validation err: 0.3085,
Validation loss: 0.5903786523267627
Epoch 16: Train err: 0.297125, Train loss: 0.5609493856430053 |Validation err: 0.305
5, Validation loss: 0.5980571266263723
Epoch 17: Train err: 0.28275, Train loss: 0.5514924840927125 |Validation err: 0.31,
Validation loss: 0.5813328130170703
Epoch 18: Train err: 0.28475, Train loss: 0.5493546750545502 |Validation err: 0.305
5, Validation loss: 0.5795436426997185
Epoch 19: Train err: 0.279875, Train loss: 0.5476849675178528 |Validation err: 0.30
9, Validation loss: 0.5848589139059186
Epoch 20: Train err: 0.284375, Train loss: 0.5463380517959595 |Validation err: 0.302
5, Validation loss: 0.5923437122255564
Epoch 21: Train err: 0.2785, Train loss: 0.5475168673992157 |Validation err: 0.299,
Validation loss: 0.5744361057877541
Epoch 22: Train err: 0.28175, Train loss: 0.5460540273189545 |Validation err: 0.308,
Validation loss: 0.5964024644345045
Epoch 23: Train err: 0.277625, Train loss: 0.5462742958068848 |Validation err: 0.29
9, Validation loss: 0.5824156841263175
Epoch 24: Train err: 0.27475, Train loss: 0.5433128743171692 |Validation err: 0.302
5, Validation loss: 0.5865197079256177
Epoch 25: Train err: 0.277, Train loss: 0.542465491771698 |Validation err: 0.3045, V
alidation loss: 0.5851405439898372
Epoch 26: Train err: 0.277125, Train loss: 0.5422690949440002 |Validation err: 0.29
3, Validation loss: 0.5749385189265013
Epoch 27: Train err: 0.278125, Train loss: 0.5429343717098236 |Validation err: 0.297
5, Validation loss: 0.5917576737701893
Epoch 28: Train err: 0.275375, Train loss: 0.5438227601051331 |Validation err: 0.299
5, Validation loss: 0.5844159498810768
Epoch 29: Train err: 0.28125, Train loss: 0.5405371904373169 |Validation err: 0.304,
Validation loss: 0.5913862539455295
Epoch 30: Train err: 0.2795, Train loss: 0.5433933699131012 |Validation err: 0.2995,
Validation loss: 0.5834077307954431
Finished Training
Total time elapsed: 116.49 seconds
Files already downloaded and verified
```

```
Files already downloaded and verified
Epoch 1: Train err: 0.489125, Train loss: 0.6925681066513062 |Validation err: 0.448
5, Validation loss: 0.6900179274380207
Epoch 2: Train err: 0.444375, Train loss: 0.6852903761863709 |Validation err: 0.425,
Validation loss: 0.6820392441004515
Epoch 3: Train err: 0.40425, Train loss: 0.6690924935340882 |Validation err: 0.376,
Validation loss: 0.6533196624368429
Epoch 4: Train err: 0.364125, Train loss: 0.6446126594543456 |Validation err: 0.396,
Validation loss: 0.6504593770951033
Epoch 5: Train err: 0.3425, Train loss: 0.624798083782196 |Validation err: 0.3365, V
alidation loss: 0.6147323381155729
Epoch 6: Train err: 0.33125, Train loss: 0.6078093810081482 |Validation err: 0.3355,
Validation loss: 0.6197773087769747
Epoch 7: Train err: 0.319875, Train loss: 0.5997646200656891 |Validation err: 0.31,
Validation loss: 0.5996882002800703
Epoch 8: Train err: 0.3075, Train loss: 0.5805826849937439 |Validation err: 0.324, V
alidation loss: 0.5944555858150125
Epoch 9: Train err: 0.296875, Train loss: 0.574046513080597 |Validation err: 0.3125,
Validation loss: 0.5872895633801818
Epoch 10: Train err: 0.29225, Train loss: 0.5596894879341126 |Validation err: 0.305
5, Validation loss: 0.5904460856691003
Epoch 11: Train err: 0.28225, Train loss: 0.5485861487388611 |Validation err: 0.297
5, Validation loss: 0.5737436469644308
Epoch 12: Train err: 0.27125, Train loss: 0.5359470489025115 |Validation err: 0.293
5, Validation loss: 0.5707168793305755
Epoch 13: Train err: 0.262375, Train loss: 0.5239814901351929 |Validation err: 0.299
5, Validation loss: 0.5754214525222778
Epoch 14: Train err: 0.252375, Train loss: 0.509468279838562 |Validation err: 0.297,
Validation loss: 0.5688865380361676
Epoch 15: Train err: 0.244875, Train loss: 0.5028299853801728 |Validation err: 0.28
6, Validation loss: 0.5623254124075174
Epoch 16: Train err: 0.253125, Train loss: 0.5033284947872162 |Validation err: 0.302
5, Validation loss: 0.5797877255827188
Epoch 17: Train err: 0.240625, Train loss: 0.4894884715080261 |Validation err: 0.28
5, Validation loss: 0.5648692278191447
Epoch 18: Train err: 0.233125, Train loss: 0.47896629238128663 |Validation err: 0.29
65, Validation loss: 0.5824996493756771
Epoch 19: Train err: 0.227, Train loss: 0.46910765624046324 |Validation err: 0.2845,
Validation loss: 0.5658214306458831
Epoch 20: Train err: 0.222, Train loss: 0.45774364709854126 |Validation err: 0.2945,
Validation loss: 0.5992406290024519
Epoch 21: Train err: 0.208, Train loss: 0.44547772097587585 |Validation err: 0.2815,
Validation loss: 0.5670876493677497
Epoch 22: Train err: 0.213875, Train loss: 0.44888710165023804 |Validation err: 0.29
45, Validation loss: 0.5934171453118324
Epoch 23: Train err: 0.208875, Train loss: 0.44089851570129396 |Validation err: 0.29
45, Validation loss: 0.5745788244530559
Epoch 24: Train err: 0.1935, Train loss: 0.4198567171096802 |Validation err: 0.288,
Validation loss: 0.6123547237366438
Epoch 25: Train err: 0.18725, Train loss: 0.40493392205238343 |Validation err: 0.28,
Validation loss: 0.5853556795045733
Epoch 26: Train err: 0.18, Train loss: 0.388867591381073 |Validation err: 0.2995, Va
lidation loss: 0.6144224535673857
Epoch 27: Train err: 0.1705, Train loss: 0.37735116958618165 |Validation err: 0.291
5, Validation loss: 0.6300702635198832
Epoch 28: Train err: 0.170875, Train loss: 0.3708683269023895 |Validation err: 0.287
5, Validation loss: 0.6313940044492483
Epoch 29: Train err: 0.163625, Train loss: 0.35624870014190674 |Validation err: 0.30
9, Validation loss: 0.7689840383827686
Epoch 30: Train err: 0.1575, Train loss: 0.34705414056777956 |Validation err: 0.297,
Validation loss: 0.6537422640249133
Finished Training
Total time elapsed: 130.21 seconds
Total time elapsed for training SMALL net: 137.28496098518372
Total time elapsed for training LARGE net: 146.4998118877411
```

The large network took almost 10 seconds more than the small network. This is due to the fact that the large network has more parameters from its convolutiona neural network dimensions and therefore need more computations than the small net to reach completion.

Part (e) - 2pt

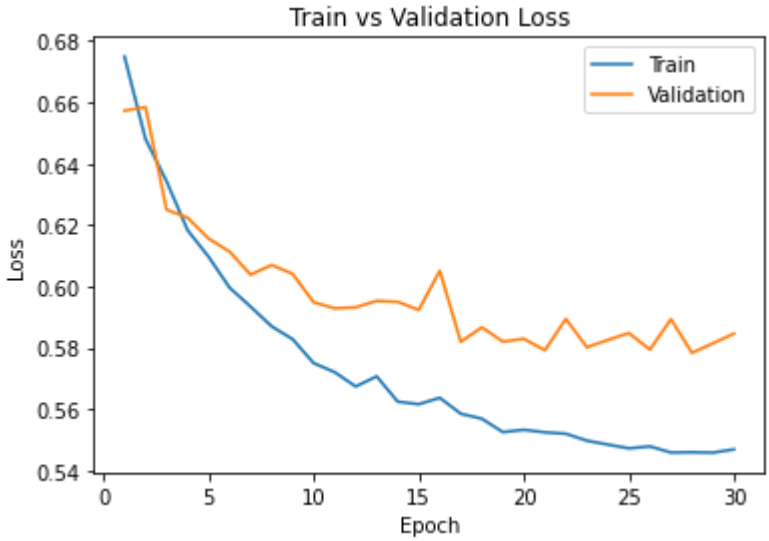
Use the function `plot_training_curve` to display the trajectory of the training/validation error and the training/validation loss. You will need to use the function `get_model_name` to generate the argument to the `plot_training_curve` function.

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

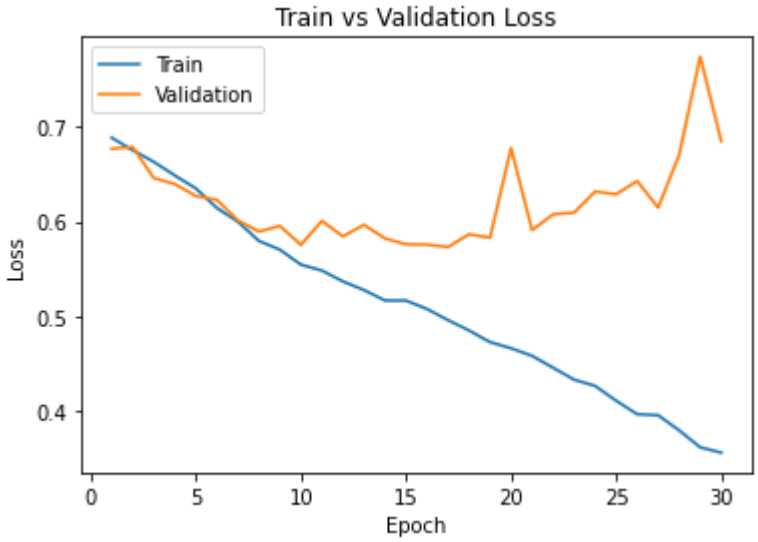
In [29]:

```
small_model_path = get_model_name("small", batch_size=64, learning_rate=0.01, epoch=29)
large_model_path = get_model_name("large", batch_size=64, learning_rate=0.01, epoch=29)
print("SMALL NN GRAPHS:")
plot_training_curve(small_model_path)
print("LARGE NN GRAPHS:")
plot_training_curve(large_model_path)
```

SMALL NN GRAPHS:



LARGE NN GRAPHS:



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.

Observations:

- Training set (blue lines) --> In the Small Net the error/loss are significantly higher than in the Large Net. E.g. at epoch = 30 error_small ~ 0.28 vs error_large ~ 0.16 and loss_small ~ 0.55 vs loss_large ~ 0.3 Also, graphs show that the training curves are much smooth for the large net than the small one. I suppose this may be because the larger net contains more data and therefore attenuates the "noise".
- Validation set (orange lines) --> Small Net and Large Net have similar error values while loss value is significantly higher for the large NN than the small one. E.g. at epoch = 30 error_small ~ 0.31 vs error_large ~ 0.32 and loss_small ~ 0.59 vs loss_large ~ 0.7

The large divergence between the training and validation set in the large neural network afer ~ 7 epoches makes us understand that this model is overfit to the training data. Indeed, after the model is trained, we see at epoch 30 that the training set has very high accuracy/low error but the validation set gets very high error/loss values. The two sets have different characteristics but since the validation set is the one better represents a "real world" dataset, the large net model will not perform well.

Part 3. Optimization Parameters [12 pt]

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

Part (a) - 3pt

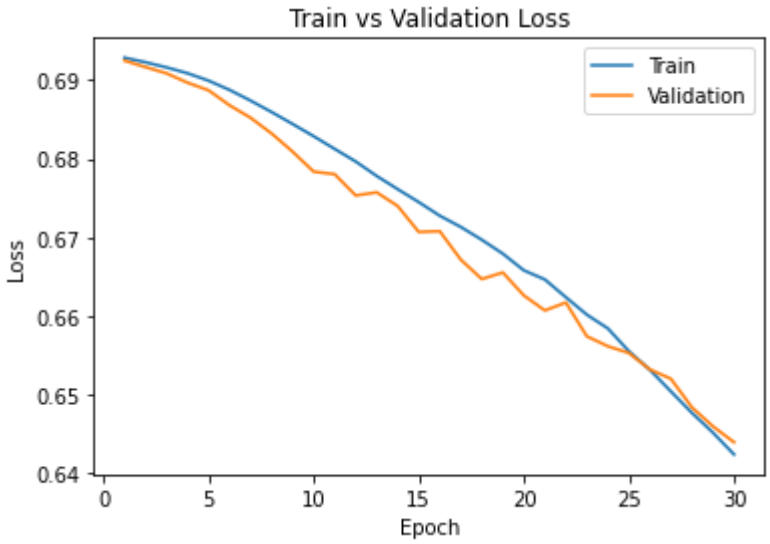
Train `large_net` with all default parameters, except set `learning_rate=0.001` . Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *lowering* the learning rate.

In [33]:
```
# Note: When we re-construct the model, we start the training
# with *random weights*. If we omit this code, the values of
# the weights will still be the previously trained values.
large_net = LargeNet()
train_net(large_net, learning_rate=0.001)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.47625, Train loss: 0.6928360004425049 |Validation err: 0.467,
Validation loss: 0.692468661814928
Epoch 2: Train err: 0.448625, Train loss: 0.6922589693069457 |Validation err: 0.430
5, Validation loss: 0.6916493456810713
Epoch 3: Train err: 0.43575, Train loss: 0.6916067309379578 |Validation err: 0.4285,
Validation loss: 0.6908544600009918
Epoch 4: Train err: 0.43, Train loss: 0.6908613452911377 |Validation err: 0.424, Val
idation loss: 0.6896595284342766
Epoch 5: Train err: 0.434125, Train loss: 0.6899194993972778 |Validation err: 0.419
5, Validation loss: 0.6886935140937567
Epoch 6: Train err: 0.43575, Train loss: 0.6887412719726562 |Validation err: 0.4195,
Validation loss: 0.686782693490386
Epoch 7: Train err: 0.436875, Train loss: 0.6873777813911438 |Validation err: 0.418
5, Validation loss: 0.6851987447589636
Epoch 8: Train err: 0.437375, Train loss: 0.68592657995224 |Validation err: 0.4115,
Validation loss: 0.6831984482705593
Epoch 9: Train err: 0.4245, Train loss: 0.6844043183326721 |Validation err: 0.411, V
alidation loss: 0.6808853577822447
Epoch 10: Train err: 0.424125, Train loss: 0.6828489475250245 |Validation err: 0.40
8, Validation loss: 0.6783470250666142
Epoch 11: Train err: 0.42525, Train loss: 0.6812356414794922 |Validation err: 0.412
5, Validation loss: 0.6780184432864189
Epoch 12: Train err: 0.420125, Train loss: 0.6796348176002502 |Validation err: 0.412
5, Validation loss: 0.6753157749772072
Epoch 13: Train err: 0.414875, Train loss: 0.6777922549247741 |Validation err: 0.41
5, Validation loss: 0.675711577758193
Epoch 14: Train err: 0.41225, Train loss: 0.6761121478080749 |Validation err: 0.412,
Validation loss: 0.6739676017314196
Epoch 15: Train err: 0.409375, Train loss: 0.6744700741767883 |Validation err: 0.414
5, Validation loss: 0.670675216242671
Epoch 16: Train err: 0.406, Train loss: 0.6727381496429443 |Validation err: 0.4105,
Validation loss: 0.6707698833197355
Epoch 17: Train err: 0.40125, Train loss: 0.6713080592155457 |Validation err: 0.403
5, Validation loss: 0.6671544443815947
Epoch 18: Train err: 0.399375, Train loss: 0.6696773543357849 |Validation err: 0.405
5, Validation loss: 0.6646812092512846
Epoch 19: Train err: 0.40075, Train loss: 0.6679102511405944 |Validation err: 0.396,
Validation loss: 0.6655160505324602
Epoch 20: Train err: 0.39225, Train loss: 0.6657903985977173 |Validation err: 0.404
5, Validation loss: 0.6626022979617119
Epoch 21: Train err: 0.3895, Train loss: 0.6646289625167847 |Validation err: 0.3935,
Validation loss: 0.6606861352920532
Epoch 22: Train err: 0.389, Train loss: 0.6623730616569519 |Validation err: 0.3935,
Validation loss: 0.6616983562707901
Epoch 23: Train err: 0.383875, Train loss: 0.6601499290466308 |Validation err: 0.397
5, Validation loss: 0.6574018411338329
Epoch 24: Train err: 0.3825, Train loss: 0.6584016590118408 |Validation err: 0.386,
Validation loss: 0.6561368461698294
Epoch 25: Train err: 0.37875, Train loss: 0.6555012550354004 |Validation err: 0.388,
Validation loss: 0.6552800387144089
Epoch 26: Train err: 0.376875, Train loss: 0.653127950668335 |Validation err: 0.387,
Validation loss: 0.6531846430152655
Epoch 27: Train err: 0.375125, Train loss: 0.6503854460716247 |Validation err: 0.387
5, Validation loss: 0.6519918907433748
Epoch 28: Train err: 0.371625, Train loss: 0.6476585249900818 |Validation err: 0.388
5, Validation loss: 0.6483596488833427
Epoch 29: Train err: 0.367875, Train loss: 0.6451436839103699 |Validation err: 0.38
2, Validation loss: 0.645940650254488
Epoch 30: Train err: 0.362875, Train loss: 0.6423715600967407 |Validation err: 0.379
5, Validation loss: 0.6439278181642294
Finished Training
Total time elapsed: 127.10 seconds
```

In [34]:
```
# ploting graphs for train/validation error/loss
large_model_path2 = get_model_name("large", batch_size=64, learning_rate=0.001, epoch
=29)
print("LARGE NN GRAPHS with learning_rate=0.001:")
plot_training_curve(large_model_path2)
```

```
LARGE NN GRAPHS with learning_rate=0.001:
```



This second model took almost 11 (=138-127) seconds less that the previous large NN model. From the plotted graphs it is very evident that the training and validation error/loss are closer to each other, which means that there was no overfitting of the the training data within the 30 epoches elapsed. We can conlude that this model is more likely to give accurate results given a completely new (i.e. real world) dataset.

Part (b) - 3pt

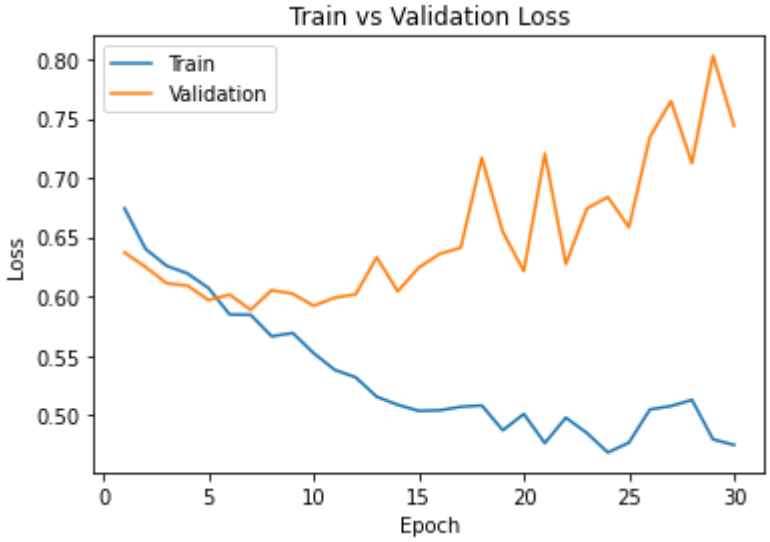
Train `large_net` with all default parameters, except set `learning_rate=0.1` . Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *increasing* the learning rate.

In [35]:
```
large_net = LargeNet()
train_net(large_net, learning_rate=0.1)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.4275, Train loss: 0.6742977557182313 |Validation err: 0.3765,
Validation loss: 0.637031726539135
Epoch 2: Train err: 0.368375, Train loss: 0.6400856471061707 |Validation err: 0.355,
Validation loss: 0.6248273365199566
Epoch 3: Train err: 0.357, Train loss: 0.6258103721141816 |Validation err: 0.3435, V
alidation loss: 0.6111582862213254
Epoch 4: Train err: 0.348625, Train loss: 0.6193035736083984 |Validation err: 0.344
5, Validation loss: 0.6090600341558456
Epoch 5: Train err: 0.330875, Train loss: 0.6069964549541473 |Validation err: 0.320
5, Validation loss: 0.5968910921365023
Epoch 6: Train err: 0.31825, Train loss: 0.5847644207477569 |Validation err: 0.3175,
Validation loss: 0.6014995649456978
Epoch 7: Train err: 0.310125, Train loss: 0.5844504678249359 |Validation err: 0.32,
Validation loss: 0.5886365948244929
Epoch 8: Train err: 0.298875, Train loss: 0.5663281772136688 |Validation err: 0.332,
Validation loss: 0.6052035503089428
Epoch 9: Train err: 0.30175, Train loss: 0.5691309769153595 |Validation err: 0.3425,
Validation loss: 0.6022673770785332
Epoch 10: Train err: 0.29175, Train loss: 0.5521122295856475 |Validation err: 0.319
5, Validation loss: 0.5920912651345134
Epoch 11: Train err: 0.27275, Train loss: 0.5380442805290222 |Validation err: 0.305,
Validation loss: 0.598898870870471
Epoch 12: Train err: 0.273125, Train loss: 0.531749195098877 |Validation err: 0.313
5, Validation loss: 0.6017276663333178
Epoch 13: Train err: 0.260625, Train loss: 0.515365345954895 |Validation err: 0.317,
Validation loss: 0.6330568259581923
Epoch 14: Train err: 0.259125, Train loss: 0.5085404133796692 |Validation err: 0.309
5, Validation loss: 0.6042338404804468
Epoch 15: Train err: 0.255625, Train loss: 0.5034471051692962 |Validation err: 0.329
5, Validation loss: 0.6242936421185732
Epoch 16: Train err: 0.252125, Train loss: 0.503919352531433 |Validation err: 0.320
5, Validation loss: 0.6358922934159636
Epoch 17: Train err: 0.253, Train loss: 0.5068532364368439 |Validation err: 0.329, V
alidation loss: 0.641121020540595
Epoch 18: Train err: 0.2595, Train loss: 0.5080235059261322 |Validation err: 0.3665,
Validation loss: 0.7169719664379954
Epoch 19: Train err: 0.239, Train loss: 0.48701162552833555 |Validation err: 0.357,
Validation loss: 0.6543149184435606
Epoch 20: Train err: 0.24625, Train loss: 0.5008019053936005 |Validation err: 0.323,
Validation loss: 0.6212702291086316
Epoch 21: Train err: 0.231375, Train loss: 0.47623249506950377 |Validation err: 0.33
2, Validation loss: 0.7205164767801762
Epoch 22: Train err: 0.244, Train loss: 0.4977173686027527 |Validation err: 0.3105,
Validation loss: 0.6273324955254793
Epoch 23: Train err: 0.238375, Train loss: 0.4847790629863739 |Validation err: 0.34
6, Validation loss: 0.6741146314889193
Epoch 24: Train err: 0.228375, Train loss: 0.4683375310897827 |Validation err: 0.309
5, Validation loss: 0.6837004749104381
Epoch 25: Train err: 0.226125, Train loss: 0.47662637662887575 |Validation err: 0.32
25, Validation loss: 0.6582820247858763
Epoch 26: Train err: 0.246125, Train loss: 0.5046286690235138 |Validation err: 0.324
5, Validation loss: 0.7345256488770247
Epoch 27: Train err: 0.249625, Train loss: 0.5075048182010651 |Validation err: 0.346
5, Validation loss: 0.7648427784442902
Epoch 28: Train err: 0.252, Train loss: 0.5126738095283508 |Validation err: 0.339, V
alidation loss: 0.7123841308057308
Epoch 29: Train err: 0.232875, Train loss: 0.4794297907352448 |Validation err: 0.334
5, Validation loss: 0.8030239716172218
Epoch 30: Train err: 0.22025, Train loss: 0.4747068645954132 |Validation err: 0.329,
Validation loss: 0.743980742059648
Finished Training
Total time elapsed: 125.04 seconds
```

In [36]:
```
# ploting graphs for train/validation error/loss
large_model_path3 = get_model_name("large", batch_size=64, learning_rate=0.1, epoch=29)
print("LARGE NN GRAPHS with learning_rate=0.1:")
plot_training_curve(large_model_path3)
```

```
LARGE NN GRAPHS with learning_rate=0.1:
```



This third model took almost 2 (=127-125) seconds less that the previous large NN model. From the plotted graphs we can see that increasing the learning rate does not decrease the error/loss of the model but, on the contrary, it overfits again the training data and the divergence between the training and validation sets is even larger than in the first model where we had learning_rate = 0.01. We can therefore conclude that a higher learning rate does not help the neural network to find a global minimum.

Part (c) - 3pt

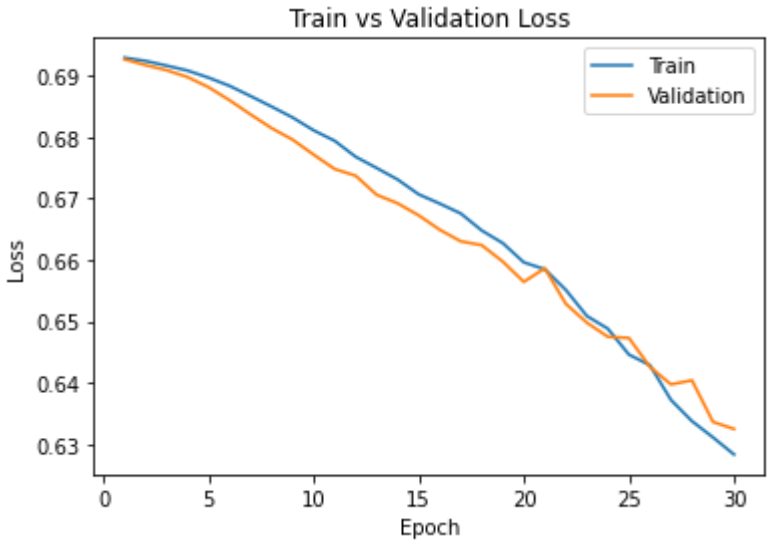
Train `large_net` with all default parameters, including with `learning_rate=0.01` . Now, set `batch_size=512` . Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *increasing* the batch size.

In [37]:
```
large_net = LargeNet()
train_net(large_net, batch_size=512, learning_rate=0.01)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.48175, Train loss: 0.6929379589855671 |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: 0.6916500627994537 |Validation err: 0.4265, V
alidation loss: 0.6909130364656448
Epoch 4: Train err: 0.433625, Train loss: 0.6908450126647949 |Validation err: 0.424,
Validation loss: 0.6897871196269989
Epoch 5: Train err: 0.434, Train loss: 0.6896936185657978 |Validation err: 0.424, Va
lidation loss: 0.6881358623504639
Epoch 6: Train err: 0.43825, Train loss: 0.6883535124361515 |Validation err: 0.4285,
Validation loss: 0.6860134303569794
Epoch 7: Train err: 0.43925, Train loss: 0.6866881102323532 |Validation err: 0.426,
Validation loss: 0.6836976855993271
Epoch 8: Train err: 0.43525, Train loss: 0.6849788911640644 |Validation err: 0.412,
Validation loss: 0.681468278169632
Epoch 9: Train err: 0.42375, Train loss: 0.6832026764750481 |Validation err: 0.414,
Validation loss: 0.6795943975448608
Epoch 10: Train err: 0.421, Train loss: 0.6811111941933632 |Validation err: 0.416, V
alidation loss: 0.6771571338176727
Epoch 11: Train err: 0.42075, Train loss: 0.6794053874909878 |Validation err: 0.409
5, Validation loss: 0.6748155355453491
Epoch 12: Train err: 0.414875, Train loss: 0.6768094897270203 |Validation err: 0.41
2, Validation loss: 0.6737167537212372
Epoch 13: Train err: 0.410375, Train loss: 0.6749758012592793 |Validation err: 0.41
2, Validation loss: 0.6706155687570572
Epoch 14: Train err: 0.407125, Train loss: 0.6730947196483612 |Validation err: 0.412
5, Validation loss: 0.6692224740982056
Epoch 15: Train err: 0.4005, Train loss: 0.6706876419484615 |Validation err: 0.4105,
Validation loss: 0.6672652214765549
Epoch 16: Train err: 0.397625, Train loss: 0.6691837050020695 |Validation err: 0.40
5, Validation loss: 0.6649233102798462
Epoch 17: Train err: 0.393875, Train loss: 0.6675742603838444 |Validation err: 0.401
5, Validation loss: 0.6630482077598572
Epoch 18: Train err: 0.392875, Train loss: 0.6648024022579193 |Validation err: 0.39
4, Validation loss: 0.6624124348163605
Epoch 19: Train err: 0.386375, Train loss: 0.6627459488809109 |Validation err: 0.38
8, Validation loss: 0.6597411781549454
Epoch 20: Train err: 0.381625, Train loss: 0.6596225798130035 |Validation err: 0.399
5, Validation loss: 0.6564429700374603
Epoch 21: Train err: 0.386, Train loss: 0.6584866605699062 |Validation err: 0.388, V
alidation loss: 0.6586524397134781
Epoch 22: Train err: 0.378375, Train loss: 0.6551279500126839 |Validation err: 0.38
6, Validation loss: 0.6528601199388504
Epoch 23: Train err: 0.372, Train loss: 0.6508878879249096 |Validation err: 0.3835,
Validation loss: 0.649801716208458
Epoch 24: Train err: 0.376625, Train loss: 0.6488120630383492 |Validation err: 0.38
4, Validation loss: 0.6474965512752533
Epoch 25: Train err: 0.36875, Train loss: 0.6446062549948692 |Validation err: 0.382,
Validation loss: 0.6473138779401779
Epoch 26: Train err: 0.372875, Train loss: 0.6428665332496166 |Validation err: 0.374
5, Validation loss: 0.642597571015358
Epoch 27: Train err: 0.3595, Train loss: 0.6372309774160385 |Validation err: 0.3785,
Validation loss: 0.639750525355339
Epoch 28: Train err: 0.354, Train loss: 0.6337686441838741 |Validation err: 0.37, Va
lidation loss: 0.6404179036617279
Epoch 29: Train err: 0.3535, Train loss: 0.6311231106519699 |Validation err: 0.3675,
Validation loss: 0.6336427330970764
Epoch 30: Train err: 0.353, Train loss: 0.6283389702439308 |Validation err: 0.3675,
Validation loss: 0.6324894577264786
Finished Training
Total time elapsed: 109.06 seconds
```

In [40]:
```
# ploting graphs for train/validation error/loss
large_model_path4 = get_model_name("large", batch_size=512, learning_rate=0.01, epoch
=29)
print("LARGE NN GRAPHS with batch_size=512:")
plot_training_curve(large_model_path4)
```

```
LARGE NN GRAPHS with batch_size=512:
```



This model took 109 seconds to finish; it has been the quickest model so far and this makes sense because by increasing the batch size we have reduced the number of computations that the neural network needs to perform. Similarly to when we reduced the learning rate, we can see from these graphs that incresing the batch size results in no overfit in the training model and consistent error/loss results between training and validation sets.

Part (d) - 3pt

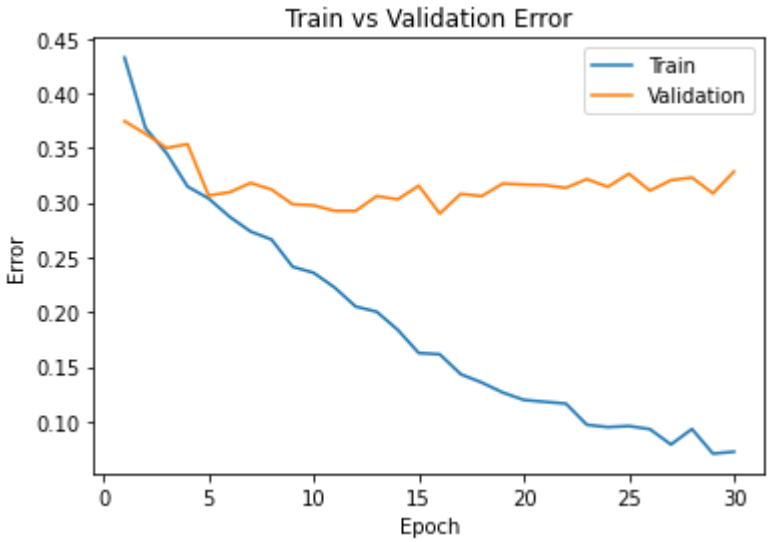
Train `large_net` with all default parameters, including with `learning_rate=0.01` . Now, set `batch_size=16` . Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *decreasing* the batch size.

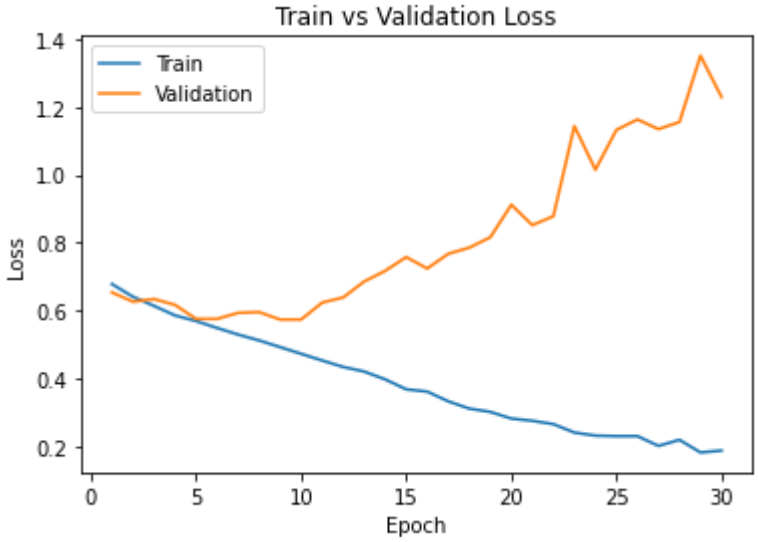
In [41]:
```
large_net = LargeNet()
train_net(large_net, batch_size=16)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.432625, Train loss: 0.6776098045110702 |Validation err: 0.374
5, Validation loss: 0.6529334635734558
Epoch 2: Train err: 0.368125, Train loss: 0.6402349994182587 |Validation err: 0.363,
Validation loss: 0.6257380561828614
Epoch 3: Train err: 0.34525, Train loss: 0.6132398887872695 |Validation err: 0.35, V
alidation loss: 0.633517231464386
Epoch 4: Train err: 0.31475, Train loss: 0.5851500154137611 |Validation err: 0.3535,
Validation loss: 0.6160770399570465
Epoch 5: Train err: 0.303875, Train loss: 0.5685942940711975 |Validation err: 0.306
5, Validation loss: 0.5749539234638215
Epoch 6: Train err: 0.287125, Train loss: 0.5481366667151452 |Validation err: 0.309
5, Validation loss: 0.5749097964763641
Epoch 7: Train err: 0.273625, Train loss: 0.5287300577163696 |Validation err: 0.318,
Validation loss: 0.5926791634559632
Epoch 8: Train err: 0.26625, Train loss: 0.5114683332741261 |Validation err: 0.312,
Validation loss: 0.5949784636497497
Epoch 9: Train err: 0.241375, Train loss: 0.4916866025328636 |Validation err: 0.298
5, Validation loss: 0.5729981939792633
Epoch 10: Train err: 0.235875, Train loss: 0.4719826597571373 |Validation err: 0.297
5, Validation loss: 0.5729825073480606
Epoch 11: Train err: 0.222375, Train loss: 0.45227504616975783 |Validation err: 0.29
25, Validation loss: 0.6230180209875107
Epoch 12: Train err: 0.205, Train loss: 0.4331896264255047 |Validation err: 0.2925,
Validation loss: 0.6381484514474869
Epoch 13: Train err: 0.20025, Train loss: 0.4197095323652029 |Validation err: 0.306,
Validation loss: 0.6852599532604218
Epoch 14: Train err: 0.18375, Train loss: 0.3964645936340094 |Validation err: 0.303,
Validation loss: 0.7167223781347275
Epoch 15: Train err: 0.1625, Train loss: 0.36721558406949045 |Validation err: 0.315
5, Validation loss: 0.7572290523052215
Epoch 16: Train err: 0.161375, Train loss: 0.36049251943826677 |Validation err: 0.2
9, Validation loss: 0.7235817478895188
Epoch 17: Train err: 0.143125, Train loss: 0.3317276249304414 |Validation err: 0.30
8, Validation loss: 0.7668795034885406
Epoch 18: Train err: 0.1355, Train loss: 0.3103339282795787 |Validation err: 0.306,
Validation loss: 0.7849839997291564
Epoch 19: Train err: 0.126375, Train loss: 0.30059052174538375 |Validation err: 0.31
75, Validation loss: 0.8156078016757965
Epoch 20: Train err: 0.119625, Train loss: 0.28093607498705386 |Validation err: 0.31
65, Validation loss: 0.9115923576354981
Epoch 21: Train err: 0.117875, Train loss: 0.2743322209268808 |Validation err: 0.31
6, Validation loss: 0.8515847996473312
Epoch 22: Train err: 0.11625, Train loss: 0.2645689582154155 |Validation err: 0.313
5, Validation loss: 0.8775560821294784
Epoch 23: Train err: 0.096875, Train loss: 0.23964216740056873 |Validation err: 0.32
15, Validation loss: 1.1437096375226974
Epoch 24: Train err: 0.09475, Train loss: 0.23061982102319598 |Validation err: 0.314
5, Validation loss: 1.0149425619840622
Epoch 25: Train err: 0.09575, Train loss: 0.2290520599540323 |Validation err: 0.326
5, Validation loss: 1.132411583542824
Epoch 26: Train err: 0.092875, Train loss: 0.22914073724485934 |Validation err: 0.31
1, Validation loss: 1.1632333843708038
Epoch 27: Train err: 0.078875, Train loss: 0.20027509773382918 |Validation err: 0.32
05, Validation loss: 1.1347101644277573
Epoch 28: Train err: 0.093, Train loss: 0.2181558216291014 |Validation err: 0.323, V
alidation loss: 1.1555380868911742
Epoch 29: Train err: 0.0705, Train loss: 0.1808979991725646 |Validation err: 0.3085,
Validation loss: 1.3511998779922725
Epoch 30: Train err: 0.07225, Train loss: 0.1865640614181757 |Validation err: 0.328
5, Validation loss: 1.2295081096887588
Finished Training
Total time elapsed: 172.31 seconds
```

In [42]:
```
# ploting graphs for train/validation error/loss
large_model_path5 = get_model_name("large", batch_size=16, learning_rate=0.01, epoch=
29)
print("LARGE NN GRAPHS with batch_size=16:")
plot_training_curve(large_model_path5)
```

```
LARGE NN GRAPHS with batch_size=16:
```





As we might have been expecting, decreasing the batch size results in a required longer time to train the model (172 > 124 > 109 secs) since we have incresed the number of iterations the neural network needs to go through. Also, from the graphs we can clearly see that the training model highly overfits once again and the inconcistency between training and validation error/loss is huge!

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 previous experiments it seems that the best set of hyperparamenters includes a large neural network - having more parameters allows indeed for higher accuracy - a large batch size and a low learning rate - since they both result in no overfitting of the training model. My first suggestion is to try (large_net, batch_size=512, learning rate=0.001)

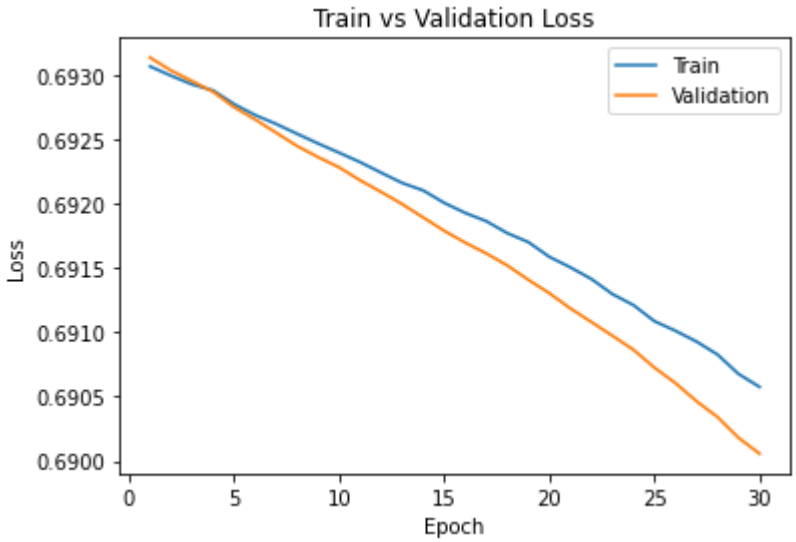
Part (b) - 1pt

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

In [45]:

```
large_net = LargeNet()
train_net(large_net, batch_size=512, learning_rate=0.001, num_epochs=30)
# ploting graphs for train/validation error/loss
model_search1 = get_model_name("large", batch_size=512, learning_rate=0.001, epoch=29
)
print("LARGE NN TEST with batch_size=512 and learing_rate=0.001:")
plot_training_curve(model_search1)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.48825, Train loss: 0.6930677443742752 |Validation err: 0.4955,
Validation loss: 0.6931362152099609
Epoch 2: Train err: 0.483125, Train loss: 0.6929955147206783 |Validation err: 0.494
5, Validation loss: 0.6930360496044159
Epoch 3: Train err: 0.480375, Train loss: 0.6929280422627926 |Validation err: 0.493,
Validation loss: 0.6929539740085602
Epoch 4: Train err: 0.477, Train loss: 0.6928808465600014 |Validation err: 0.4885, V
alidation loss: 0.6928706914186478
Epoch 5: Train err: 0.473375, Train loss: 0.692774411290884 |Validation err: 0.4835,
Validation loss: 0.692750483751297
Epoch 6: Train err: 0.469, Train loss: 0.692689623683691 |Validation err: 0.472, Val
idation loss: 0.6926551908254623
Epoch 7: Train err: 0.46325, Train loss: 0.692620363086462 |Validation err: 0.47, Va
lidation loss: 0.6925524473190308
Epoch 8: Train err: 0.46225, Train loss: 0.6925435587763786 |Validation err: 0.463,
Validation loss: 0.6924485266208649
Epoch 9: Train err: 0.459625, Train loss: 0.6924680471420288 |Validation err: 0.457,
Validation loss: 0.6923621445894241
Epoch 10: Train err: 0.458, Train loss: 0.6923965588212013 |Validation err: 0.4555,
Validation loss: 0.6922826170921326
Epoch 11: Train err: 0.454875, Train loss: 0.692323099821806 |Validation err: 0.450
5, Validation loss: 0.692181870341301
Epoch 12: Train err: 0.4535, Train loss: 0.6922412402927876 |Validation err: 0.441,
Validation loss: 0.6920914500951767
Epoch 13: Train err: 0.450375, Train loss: 0.6921614743769169 |Validation err: 0.43
7, Validation loss: 0.6919969022274017
Epoch 14: Train err: 0.44725, Train loss: 0.6921032629907131 |Validation err: 0.433,
Validation loss: 0.6918932199478149
Epoch 15: Train err: 0.449375, Train loss: 0.6920064613223076 |Validation err: 0.43
2, Validation loss: 0.6917892098426819
Epoch 16: Train err: 0.44425, Train loss: 0.6919283792376518 |Validation err: 0.432,
Validation loss: 0.6916972696781158
Epoch 17: Train err: 0.441375, Train loss: 0.6918644830584526 |Validation err: 0.43
1, Validation loss: 0.6916135549545288
Epoch 18: Train err: 0.438125, Train loss: 0.6917712613940239 |Validation err: 0.429
5, Validation loss: 0.6915201842784882
Epoch 19: Train err: 0.436375, Train loss: 0.6917018331587315 |Validation err: 0.42
8, Validation loss: 0.6914086788892746
Epoch 20: Train err: 0.436375, Train loss: 0.6915871202945709 |Validation err: 0.427
5, Validation loss: 0.6913044303655624
Epoch 21: Train err: 0.437, Train loss: 0.6915052533149719 |Validation err: 0.4285,
Validation loss: 0.6911861002445221
Epoch 22: Train err: 0.438625, Train loss: 0.6914149858057499 |Validation err: 0.42
8, Validation loss: 0.6910804063081741
Epoch 23: Train err: 0.436875, Train loss: 0.691297460347414 |Validation err: 0.428,
Validation loss: 0.6909734606742859
Epoch 24: Train err: 0.436875, Train loss: 0.6912120655179024 |Validation err: 0.42
5, Validation loss: 0.6908645182847977
Epoch 25: Train err: 0.435125, Train loss: 0.6910865493118763 |Validation err: 0.425
5, Validation loss: 0.6907256990671158
Epoch 26: Train err: 0.434625, Train loss: 0.6910119391977787 |Validation err: 0.424
5, Validation loss: 0.6906052231788635
Epoch 27: Train err: 0.43675, Train loss: 0.6909283809363842 |Validation err: 0.426
5, Validation loss: 0.6904649585485458
Epoch 28: Train err: 0.43575, Train loss: 0.6908275820314884 |Validation err: 0.426
5, Validation loss: 0.6903414130210876
Epoch 29: Train err: 0.436375, Train loss: 0.6906765811145306 |Validation err: 0.42
3, Validation loss: 0.6901804357767105
Epoch 30: Train err: 0.435625, Train loss: 0.690575547516346 |Validation err: 0.423
5, Validation loss: 0.6900565922260284
Finished Training
Total time elapsed: 107.10 seconds
LARGE NN TEST with batch_size=512 and learing_rate=0.001:
```



Part (c) - 2pt

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

From the first suggested set of hyperparameters there was no overfitting so based on part (a) and (b) another set of parameters we could test in order to try to reduce the train/validation error (which was quite high in the previous set compared to previus cases, ~42%) is batch_size=600, learning rate=0.004. By slightly increasing the batch size and the learning rate we hope to reduce the error percentage without overfitting the training model. This time I will use a small_net to make sure that the higher learning rate does not cause overfit.

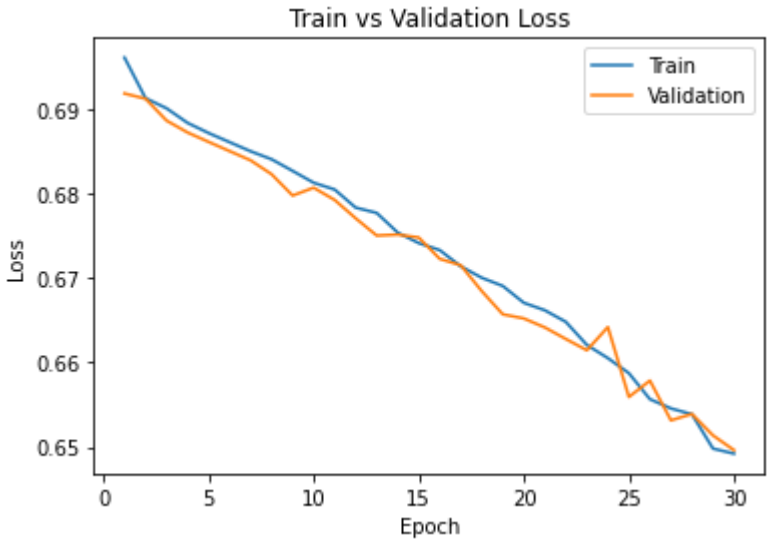
Part (d) - 1pt

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

In [22]:

```
small_net = SmallNet()
train_net(small_net, batch_size=600, learning_rate=0.004, num_epochs=30)
# ploting graphs for train/validation error/loss
model_search2 = get_model_name("small", batch_size=600, learning_rate=0.004, epoch=29
)
print("LARGE NN TEST with batch_size=600 and learing_rate=0.004:")
plot_training_curve(model_search2)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.49725, Train loss: 0.6962058544158936 |Validation err: 0.4725,
Validation loss: 0.691946342587471
Epoch 2: Train err: 0.483375, Train loss: 0.6913888198988778 |Validation err: 0.477
5, Validation loss: 0.6913255006074905
Epoch 3: Train err: 0.471375, Train loss: 0.6901889187949044 |Validation err: 0.459,
Validation loss: 0.688731923699379
Epoch 4: Train err: 0.453, Train loss: 0.688457693372454 |Validation err: 0.454, Val
idation loss: 0.6873193681240082
Epoch 5: Train err: 0.44475, Train loss: 0.6872541180678776 |Validation err: 0.436,
Validation loss: 0.6862045675516129
Epoch 6: Train err: 0.441375, Train loss: 0.6861703779016223 |Validation err: 0.436
5, Validation loss: 0.6851088404655457
Epoch 7: Train err: 0.437875, Train loss: 0.6850897712366921 |Validation err: 0.427
5, Validation loss: 0.6840131729841232
Epoch 8: Train err: 0.436125, Train loss: 0.6841362714767456 |Validation err: 0.424
5, Validation loss: 0.6823611855506897
Epoch 9: Train err: 0.4345, Train loss: 0.6827569007873535 |Validation err: 0.4185,
Validation loss: 0.6798324733972549
Epoch 10: Train err: 0.426, Train loss: 0.6813540586403438 |Validation err: 0.4225,
Validation loss: 0.6807624101638794
Epoch 11: Train err: 0.422125, Train loss: 0.6805474119527 |Validation err: 0.422, V
alidation loss: 0.6793078184127808
Epoch 12: Train err: 0.422625, Train loss: 0.6784005079950605 |Validation err: 0.408
5, Validation loss: 0.6771319210529327
Epoch 13: Train err: 0.417625, Train loss: 0.6777703038283757 |Validation err: 0.415
5, Validation loss: 0.6750840097665787
Epoch 14: Train err: 0.411875, Train loss: 0.6754355302878788 |Validation err: 0.405
5, Validation loss: 0.6751988679170609
Epoch 15: Train err: 0.40575, Train loss: 0.6741908660956791 |Validation err: 0.4, V
alidation loss: 0.6748519986867905
Epoch 16: Train err: 0.40625, Train loss: 0.6733545107500893 |Validation err: 0.387
5, Validation loss: 0.6723185330629349
Epoch 17: Train err: 0.4035, Train loss: 0.6714280290263039 |Validation err: 0.387,
Validation loss: 0.6715567708015442
Epoch 18: Train err: 0.396125, Train loss: 0.6700717381068638 |Validation err: 0.383
5, Validation loss: 0.6684847325086594
Epoch 19: Train err: 0.39625, Train loss: 0.6690980579171862 |Validation err: 0.383,
Validation loss: 0.6657331883907318
Epoch 20: Train err: 0.39375, Train loss: 0.6671014981610435 |Validation err: 0.376,
Validation loss: 0.6652442812919617
Epoch 21: Train err: 0.39225, Train loss: 0.6662125289440155 |Validation err: 0.378
5, Validation loss: 0.664176270365715
Epoch 22: Train err: 0.387125, Train loss: 0.6648390676294055 |Validation err: 0.373
5, Validation loss: 0.6628039926290512
Epoch 23: Train err: 0.382375, Train loss: 0.6620988505227225 |Validation err: 0.371
5, Validation loss: 0.661468580365181
Epoch 24: Train err: 0.380375, Train loss: 0.6605312951973507 |Validation err: 0.365
5, Validation loss: 0.6642224192619324
Epoch 25: Train err: 0.3765, Train loss: 0.6587460041046143 |Validation err: 0.3645,
Validation loss: 0.6559229791164398
Epoch 26: Train err: 0.374375, Train loss: 0.6556600332260132 |Validation err: 0.364
5, Validation loss: 0.6578655689954758
Epoch 27: Train err: 0.37475, Train loss: 0.6545626733984266 |Validation err: 0.364,
Validation loss: 0.6531492918729782
Epoch 28: Train err: 0.371125, Train loss: 0.6538683814661843 |Validation err: 0.364
5, Validation loss: 0.6539032906293869
Epoch 29: Train err: 0.3685, Train loss: 0.649798469884055 |Validation err: 0.358, V
alidation loss: 0.6513632386922836
Epoch 30: Train err: 0.36525, Train loss: 0.6491962841578892 |Validation err: 0.360
5, Validation loss: 0.6496092677116394
Finished Training
Total time elapsed: 102.77 seconds
LARGE NN TEST with batch_size=600 and learing_rate=0.004:
```



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 [24]:
```
net = SmallNet()
model_path = get_model_name(net.name, batch_size=600, learning_rate=0.004, epoch=29)
state = torch.load(model_path)
net.load_state_dict(state)
```

Out[24]:
```
<All keys matched successfully>
```

Part (b) - 2pt

Justify your choice of model from part (a).

My choice of what is the best model is based on two chriteria:

1. The model must not overfit the training data
2. Both training and validation sets need to have the lower possible error/loss values since this shows that the model can correctly perform with higher accuracy when using a "real world" dataset.

In general, from previous tests I saw that the models where there is overfitting have bad accuracy on the validation data, and the models with no overfitting have low accuracy all around. Therefore I used the model from Part 4(c): a small_net model with the higher batch size and slightly higher learning rate. Indeed this model has a better accuracy than the preceeding models without overfitting.

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 [25]:
```
# 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)
```

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

In [34]:
```
errC, lossC = evaluate(net, test_loader, nn.BCEWithLogitsLoss())
errorC = "Test classification error is {}".format(errC)
print(errorC)
lossssC = "Test classification loss is {}".format(lossC)
print(lossssC)
```

```
Test classification error is 0.3695
Test classification loss is 0.6474927384406328
```

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]:
```
errV, lossV = evaluate(net, val_loader, nn.BCEWithLogitsLoss())
errorV = "Validation error is {}".format(errV)
print(errorV)
lossssV = "Validation loss is {}".format(lossV)
print(lossssV)
```

```
Validation error is 0.3605
Validation loss is 0.6510091107338667
```

The model I have choosen gives slightly lower error (0.3605) on the validation data than on the test data (0.3695). Since the model is continuously tweaked and optimized to perform well against a valiation set, the training set in a sort of sense gets "biased" by the validation set. Therefore, i suppose that when a testing set (which is different than the validation set) enters the model, there is a chance that the model performs with less accuracy on the testing set or it could perform better on the testing set if the testing set shares characteristics that align more with the training set than the validation data.

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 dataset is needed for the very last final check. If we had had used it before, we would have gained insights on model performance and we would have fallen into the temptation of optimizing the model based on that dataset. It is important to use the test data as little as possible because otherwise there would be no true final test for the model to see if it performs well against real world data (i.e. unknown/unpredictable dataset).

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 [37]:
```
# Code taken from lab 1
torch.manual_seed(1) # set the random seed
class CatsVsDogs(nn.Module):# define a 2-layer artificial neural network
    def __init__(self):
        self.name = "CatsVsDogs"
        super(CatsVsDogs, self).__init__()
        self.layer1 = nn.Linear(3 * 32*32, 60)
        self.layer2 = nn.Linear(60, 1)
    def forward(self, img):
        flattened = img.view(-1, 3 * 32*32)
        activation1 = self.layer1(flattened)
        activation1 = F.relu(activation1)
        activation2 = self.layer2(activation1)
        activation2 = activation2.squeeze(1)
        return activation2

train_CatsVsDogs = CatsVsDogs()
#values from training are taken from my best model acheved in Part4(c)
train_net(train_CatsVsDogs, batch_size = 600, learning_rate=0.004, num_epochs = 30)
best_model = get_model_name("CatsVsDogs", batch_size=600, learning_rate=0.004, epoch=
29)
plot_training_curve(best_model)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.466625, Train loss: 0.6891649578298841 |Validation err: 0.437,
Validation loss: 0.6842824518680573
Epoch 2: Train err: 0.415875, Train loss: 0.6765400469303131 |Validation err: 0.419,
Validation loss: 0.6719160228967667
Epoch 3: Train err: 0.3995, Train loss: 0.6687765462057931 |Validation err: 0.4125,
Validation loss: 0.668404221534729
Epoch 4: Train err: 0.39675, Train loss: 0.6624033323356083 |Validation err: 0.406,
Validation loss: 0.6630838811397552
Epoch 5: Train err: 0.39025, Train loss: 0.6580306206430707 |Validation err: 0.4035,
Validation loss: 0.6593413203954697
Epoch 6: Train err: 0.388, Train loss: 0.6561821954590934 |Validation err: 0.4, Vali
dation loss: 0.6616483181715012
Epoch 7: Train err: 0.382375, Train loss: 0.6517495853560311 |Validation err: 0.400
5, Validation loss: 0.6592665016651154
Epoch 8: Train err: 0.376875, Train loss: 0.6492433718272618 |Validation err: 0.393
5, Validation loss: 0.6568602025508881
Epoch 9: Train err: 0.376125, Train loss: 0.6459116297108787 |Validation err: 0.393,
Validation loss: 0.6505311280488968
Epoch 10: Train err: 0.37425, Train loss: 0.6444470712116787 |Validation err: 0.394
5, Validation loss: 0.655517503619194
Epoch 11: Train err: 0.370125, Train loss: 0.6417097960199628 |Validation err: 0.392
5, Validation loss: 0.6514455378055573
Epoch 12: Train err: 0.3645, Train loss: 0.6377735265663692 |Validation err: 0.389,
Validation loss: 0.6498356908559799
Epoch 13: Train err: 0.36525, Train loss: 0.6372105606964656 |Validation err: 0.388
5, Validation loss: 0.6490793079137802
Epoch 14: Train err: 0.361625, Train loss: 0.6334422273295266 |Validation err: 0.39,
Validation loss: 0.6534063071012497
Epoch 15: Train err: 0.359, Train loss: 0.6324563792773655 |Validation err: 0.389, V
alidation loss: 0.6491223573684692
Epoch 16: Train err: 0.3575, Train loss: 0.6322726820196424 |Validation err: 0.3855,
Validation loss: 0.6492438763380051
Epoch 17: Train err: 0.35525, Train loss: 0.6306419883455548 |Validation err: 0.382
5, Validation loss: 0.6462553441524506
Epoch 18: Train err: 0.35225, Train loss: 0.6272191490445819 |Validation err: 0.382,
Validation loss: 0.6486814022064209
Epoch 19: Train err: 0.351125, Train loss: 0.627028056553432 |Validation err: 0.378
5, Validation loss: 0.6428937762975693
Epoch 20: Train err: 0.350125, Train loss: 0.6240121977669852 |Validation err: 0.38,
Validation loss: 0.6482072174549103
Epoch 21: Train err: 0.350375, Train loss: 0.6233591479914529 |Validation err: 0.378
5, Validation loss: 0.6470166742801666
Epoch 22: Train err: 0.34575, Train loss: 0.6237070986202785 |Validation err: 0.381,
Validation loss: 0.6456041187047958
Epoch 23: Train err: 0.342625, Train loss: 0.6204481167452676 |Validation err: 0.38
2, Validation loss: 0.6456600576639175
Epoch 24: Train err: 0.34025, Train loss: 0.6167318097182682 |Validation err: 0.383,
Validation loss: 0.6523781418800354
Epoch 25: Train err: 0.339125, Train loss: 0.6174277705805642 |Validation err: 0.381
5, Validation loss: 0.6427127569913864
Epoch 26: Train err: 0.3375, Train loss: 0.6145449536187308 |Validation err: 0.382,
Validation loss: 0.6487342417240143
Epoch 27: Train err: 0.33525, Train loss: 0.6125987938472203 |Validation err: 0.386,
Validation loss: 0.6475462317466736
Epoch 28: Train err: 0.33375, Train loss: 0.610906434910638 |Validation err: 0.3845,
Validation loss: 0.6446759551763535
Epoch 29: Train err: 0.3315, Train loss: 0.608482654605593 |Validation err: 0.3835,
Validation loss: 0.6427947878837585
Epoch 30: Train err: 0.33225, Train loss: 0.6081497371196747 |Validation err: 0.381,
Validation loss: 0.6442983597517014
Finished Training
Total time elapsed: 82.69 seconds
```

In [39]:
```
train_loader, val_loader, test_loader, classes = get_data_loader(
    target_classes=["cat", "dog"],
    batch_size=600)

errC, lossC = evaluate(train_CatsVsDogs, test_loader, nn.BCEWithLogitsLoss())
errorC = "Test classification error is {}".format(errC)
print(errorC)
lossssC = "Test classification loss is {}".format(lossC)
print(lossssC)

errV, lossV = evaluate(train_CatsVsDogs, val_loader, nn.BCEWithLogitsLoss())
errorV = "Validation error is {}".format(errV)
print(errorV)
lossssV = "Validation loss is {}".format(lossV)
print(lossssV)
```

```
Files already downloaded and verified
Files already downloaded and verified
Test classification error is 0.363
Test classification loss is 0.6383683830499649
Validation error is 0.381
Validation loss is 0.6446462720632553
```

From the collected results it is clear that the CNN architecture performs way better than my 2 layer ANN architecture. \
E.g. Val_err_CNN = 0.3605 vs Val_err_ANN = 0.381