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/1MtGkaegLqPdIs6R4cVQlUEkV9Mr7sLZH/view?usp=sharing (https://drive.google.com/file/d/1MtGkaegLqPdIs6R4cVQlUEkV9Mr7sLZH/view?usp=sharing)

In [1]:
```
import numpy as np
import time
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
from torch.utils.data.sampler import SubsetRandomSampler
import torchvision.transforms as transforms
```

Part 0. Helper Functions

We will be making use of the following helper functions. You will be asked to look at and possibly modify some of these, but you are not expected to understand all of them.

You should look at the function names and read the docstrings. If you are curious, come back and explore the code *after* making some progress on the lab.

In [2]:
```
##############################################################################
# Data Loading

def get_relevant_indices(dataset, classes, target_classes):
    """ Return the indices for datapoints in the dataset that belongs to the
    desired target classes, a subset of all possible classes.

    Args:
        dataset: Dataset object
        classes: A list of strings denoting the name of each class
        target_classes: A list of strings denoting the name of desired 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[s
plit:]
    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
```

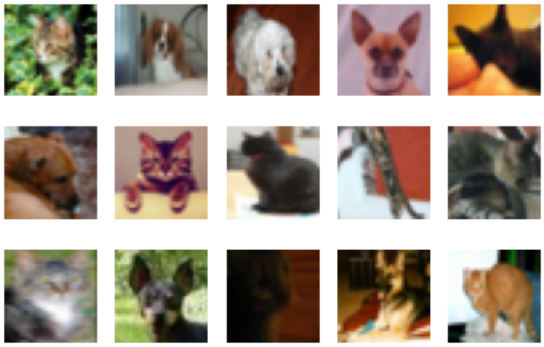
```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-pytho
n.tar.gz


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

```
Total training examples: 8000
Total validation examples: 2000
Total test examples: 2000
```

Part (c) -- 3pt

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

In [6]:
```
# The validation set is used to tune the hyperparameters of the model after
# training the weights (learnable parameters). The performance of the model
# on the training set error is a poor measure, since the model was trained
# using the data. The model's predictions on unobserved data is a much more
# relevant measure; the training set does not qualify as "unobserved" data.
# The performance of the model will always be great against the training 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 [7]:
```
class LargeNet(nn.Module):
    def __init__(self):
        super(LargeNet, self).__init__()
        self.name = "large"
        self.conv1 = nn.Conv2d(3, 5, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(5, 10, 5)
        self.fc1 = nn.Linear(10 * 5 * 5, 32)
        self.fc2 = nn.Linear(32, 1)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 10 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        x = x.squeeze(1) # Flatten to [batch_size]
        return x
```

In [8]:
```
class SmallNet(nn.Module):
    def __init__(self):
        super(SmallNet, self).__init__()
        self.name = "small"
        self.conv = nn.Conv2d(3, 5, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc = nn.Linear(5 * 7 * 7, 1)

    def forward(self, x):
        x = self.pool(F.relu(self.conv(x)))
        x = self.pool(x)
        x = x.view(-1, 5 * 7 * 7)
        x = self.fc(x)
        x = x.squeeze(1) # Flatten to [batch_size]
        return x
```

In [9]:
```
small_net = SmallNet()
large_net = LargeNet()
```

Part (a) -- 2pt

The methods `small_net.parameters()` and `large_net.parameters()` produces an iterator of all the trainable parameters of the network. These parameters are torch tensors containing many scalar values.

We haven't learned how how the parameters in these high-dimensional tensors will be used, but we should be able to count the number of parameters. Measuring the number of parameters in a network is one way of measuring the "size" of a network.

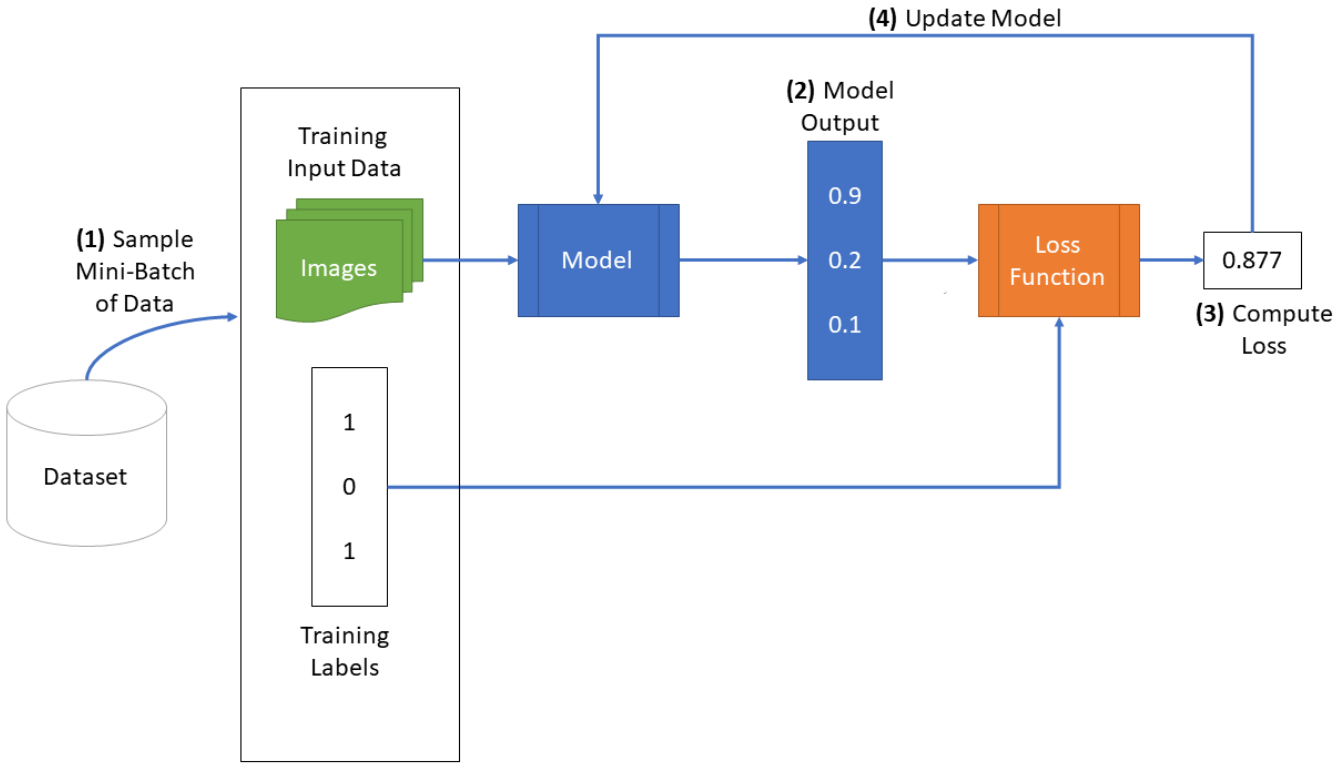
What is the total number of parameters in `small_net` and in `large_net`? (Hint: how many numbers are in each tensor?)

In [10]:
```
print("Total number of parameters in small_net:", sum([
  torch.numel(param)

  for param in small_net.parameters()
]))

print("Total number of parameters in large_net:", sum([
  torch.numel(param)

  for param in large_net.parameters()
]))
```

```
Total number of parameters in small_net: 386
Total number of parameters in large_net: 9705
```

The function train_net

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



In [11]:

```
def train_net(net, batch_size=64, learning_rate=0.01, num_epochs=30):
    ################################################################################
    # Train a classifier on cats vs dogs
    target_classes = ["cat", "dog"]
    ################################################################################
    # Fixed PyTorch random seed for reproducible result
    torch.manual_seed(1000)
    ################################################################################
    # Obtain the PyTorch data loader objects to load batches of the datasets
    train_loader, val_loader, test_loader, classes = get_data_loader(
            target_classes, batch_size)
    ################################################################################
    # Define the Loss function and optimizer
    # The loss function will be Binary Cross Entropy (BCE). In this case we
    # will use the BCEWithLogitsLoss which takes unnormalized output from
    # the neural network and scalar label.
    # Optimizer will be SGD with Momentum.
    criterion = nn.BCEWithLogitsLoss()
    optimizer = optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
    ################################################################################
    # Set up some numpy arrays to store the training/test loss/erruracy
    train_err = np.zeros(num_epochs)
    train_loss = np.zeros(num_epochs)
    val_err = np.zeros(num_epochs)
    val_loss = np.zeros(num_epochs)
    ################################################################################
    # Train the network
    # Loop over the data iterator and sample a new batch of training data
    # Get the output from the network, and optimize our loss function.
    start_time = time.time()
    for epoch in range(num_epochs):  # loop over the dataset multiple times
        total_train_loss = 0.0
        total_train_err = 0.0
        total_epoch = 0
        for i, data in enumerate(train_loader, 0):
            # Get the inputs
            inputs, labels = data
            labels = normalize_label(labels) # Convert labels to 0/1
            # Zero the parameter gradients
            optimizer.zero_grad()
            # Forward pass, backward pass, and optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels.float())
            loss.backward()
            optimizer.step()
            # Calculate the statistics
            corr = (outputs > 0.0).squeeze().long() != labels
            total_train_err += int(corr.sum())
            total_train_loss += loss.item()
            total_epoch += len(labels)
        train_err[epoch] = float(total_train_err) / total_epoch
        train_loss[epoch] = float(total_train_loss) / (i+1)
        val_err[epoch], val_loss[epoch] = evaluate(net, val_loader, 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 [12]:
```
# The default parameters 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 [13]:
```
# The four files that are written to the disk are
# - model_small_bs64_lr0.01_epoch4_train_err.csv (training error of each epoch)
# - model_small_bs64_lr0.01_epoch4_train_loss.csv (training loss of each epoch)
# - model_small_bs64_lr0.01_epoch4_val_error.csv (testing error of each epoch)
# - model_small_bs64_lr0.01_epoch4_val_loss.csv (testing loss of 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 [14]:
```
# Since the function writes files to disk, you will need to mount
# your Google Drive. If you are working on the lab locally, you
# can comment out this code.

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

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

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

# The small network took 102.63 seconds while the large network took 111.40
# seconds. This makes sense as the larger network has 9,319 more parameters to
# train, and would take longer to train.
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.423625, Train loss: 0.6725465445518494 |Validation err: 0.386, Validati
on loss: 0.6587053779512644
Epoch 2: Train err: 0.37575, Train loss: 0.650308669090271 |Validation err: 0.387, Validation
loss: 0.6656124647706747
Epoch 3: Train err: 0.360375, Train loss: 0.6403244805335998 |Validation err: 0.3475, Validat
ion loss: 0.6315254252403975
Epoch 4: Train err: 0.34875, Train loss: 0.6274946994781494 |Validation err: 0.345, Validatio
n loss: 0.6260619889944792
Epoch 5: Train err: 0.34525, Train loss: 0.6176664690971374 |Validation err: 0.341, Validatio
n loss: 0.619855560362339
Epoch 6: Train err: 0.328875, Train loss: 0.6052448828220367 |Validation err: 0.331, Validati
on loss: 0.612820478156209
Epoch 7: Train err: 0.321875, Train loss: 0.5986335027217865 |Validation err: 0.3275, Validat
ion loss: 0.6104573626071215
Epoch 8: Train err: 0.3135, Train loss: 0.5914606981277466 |Validation err: 0.334, Validation
loss: 0.6129683535546064
Epoch 9: Train err: 0.3155, Train loss: 0.589647706747055 |Validation err: 0.329, Validation
loss: 0.6122564412653446
Epoch 10: Train err: 0.310875, Train loss: 0.5829922301769257 |Validation err: 0.3195, Valida
tion loss: 0.6066484814509749
Epoch 11: Train err: 0.30775, Train loss: 0.5816051411628723 |Validation err: 0.3175, Validat
ion loss: 0.6091482602059841
Epoch 12: Train err: 0.303125, Train loss: 0.5756118743419647 |Validation err: 0.3345, Valida
tion loss: 0.6117115011438727
Epoch 13: Train err: 0.3065, Train loss: 0.579054584980011 |Validation err: 0.3205, Validatio
n loss: 0.6068468289449811
Epoch 14: Train err: 0.29925, Train loss: 0.5720612363815307 |Validation err: 0.333, Validati
on loss: 0.6151926228776574
Epoch 15: Train err: 0.298, Train loss: 0.5706197772026061 |Validation err: 0.317, Validation
loss: 0.6050211684778333
Epoch 16: Train err: 0.301375, Train loss: 0.5725428817272187 |Validation err: 0.321, Validat
ion loss: 0.6153124030679464
Epoch 17: Train err: 0.29575, Train loss: 0.5679456496238708 |Validation err: 0.3115, Validat
ion loss: 0.5957430517300963
Epoch 18: Train err: 0.294375, Train loss: 0.5640334222316742 |Validation err: 0.3205, Valida
tion loss: 0.6030995976179838
Epoch 19: Train err: 0.287375, Train loss: 0.5593593299388886 |Validation err: 0.326, Validat
ion loss: 0.6158971972763538
Epoch 20: Train err: 0.294875, Train loss: 0.5600222787857055 |Validation err: 0.3165, Valida
tion loss: 0.5926138143986464
Epoch 21: Train err: 0.288375, Train loss: 0.5604016141891479 |Validation err: 0.31, Validati
on loss: 0.5911781489849091
Epoch 22: Train err: 0.293125, Train loss: 0.5587935426235199 |Validation err: 0.326, Validat
ion loss: 0.6049761157482862
Epoch 23: Train err: 0.2895, Train loss: 0.5581324179172515 |Validation err: 0.3145, Validati
on loss: 0.5978777734562755
Epoch 24: Train err: 0.285875, Train loss: 0.5508356144428254 |Validation err: 0.3075, Valida
tion loss: 0.5861171744763851
Epoch 25: Train err: 0.280875, Train loss: 0.5498914589881897 |Validation err: 0.314, Validat
ion loss: 0.582728105597198
Epoch 26: Train err: 0.283125, Train loss: 0.548797999382019 |Validation err: 0.311, Validati
on loss: 0.5865387348458171
Epoch 27: Train err: 0.27875, Train loss: 0.5461882054805756 |Validation err: 0.3025, Validat
ion loss: 0.592717151157558
Epoch 28: Train err: 0.280875, Train loss: 0.5483453097343445 |Validation err: 0.3025, Valida
tion loss: 0.5814519925042987
Epoch 29: Train err: 0.28, Train loss: 0.5502915124893188 |Validation err: 0.3075, Validation
loss: 0.6022119717672467
Epoch 30: Train err: 0.279375, Train loss: 0.5457119107246399 |Validation err: 0.31, Validati
on loss: 0.5836448781192303
Finished Training
Total time elapsed: 115.13 seconds
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.447125, Train loss: 0.6891509671211242 |Validation err: 0.414, Validati
on loss: 0.6776208057999611
Epoch 2: Train err: 0.412875, Train loss: 0.6756665749549866 |Validation err: 0.4315, Validat
ion loss: 0.6815168615430593
Epoch 3: Train err: 0.39275, Train loss: 0.6620882997512817 |Validation err: 0.357, Validatio
```

```
n loss: 0.644333915784955
Epoch 4: Train err: 0.36425, Train loss: 0.6416445260047913 |Validation err: 0.367, Validatio
n loss: 0.6396895572543144
Epoch 5: Train err: 0.35125, Train loss: 0.6296070952415467 |Validation err: 0.346, Validatio
n loss: 0.6262928657233715
Epoch 6: Train err: 0.32925, Train loss: 0.613728575706482 |Validation err: 0.345, Validation
loss: 0.6226661149412394
Epoch 7: Train err: 0.32125, Train loss: 0.6006641488075256 |Validation err: 0.3325, Validati
on loss: 0.6035385970026255
Epoch 8: Train err: 0.31375, Train loss: 0.5827394366264343 |Validation err: 0.3205, Validati
on loss: 0.5939359944313765
Epoch 9: Train err: 0.2975, Train loss: 0.5704047927856445 |Validation err: 0.317, Validation
loss: 0.5887306202203035
Epoch 10: Train err: 0.29275, Train loss: 0.5566689889431 |Validation err: 0.3025, Validation
loss: 0.5805406859144568
Epoch 11: Train err: 0.278375, Train loss: 0.5465063607692718 |Validation err: 0.3225, Valida
tion loss: 0.5925672575831413
Epoch 12: Train err: 0.274, Train loss: 0.536328331708908 |Validation err: 0.318, Validation
loss: 0.5940371369943023
Epoch 13: Train err: 0.26225, Train loss: 0.5231085739135742 |Validation err: 0.308, Validati
on loss: 0.5891228504478931
Epoch 14: Train err: 0.258125, Train loss: 0.515414620399475 |Validation err: 0.293, Validati
on loss: 0.5828925613313913
Epoch 15: Train err: 0.250875, Train loss: 0.5061796193122864 |Validation err: 0.303, Validat
ion loss: 0.5898134810850024
Epoch 16: Train err: 0.2455, Train loss: 0.5001272361278534 |Validation err: 0.3055, Validati
on loss: 0.5826594298705459
Epoch 17: Train err: 0.237875, Train loss: 0.4859476044178009 |Validation err: 0.298, Validat
ion loss: 0.583400116302073
Epoch 18: Train err: 0.2325, Train loss: 0.4763805820941925 |Validation err: 0.3045, Validati
on loss: 0.5940498914569616
Epoch 19: Train err: 0.22975, Train loss: 0.47174351930618286 |Validation err: 0.3025, Valida
tion loss: 0.5812169453129172
Epoch 20: Train err: 0.213625, Train loss: 0.45457123017311096 |Validation err: 0.3185, Valid
ation loss: 0.6692689387127757
Epoch 21: Train err: 0.213625, Train loss: 0.45125345325469973 |Validation err: 0.3115, Valid
ation loss: 0.6032408941537142
Epoch 22: Train err: 0.2065, Train loss: 0.43983844661712646 |Validation err: 0.3065, Validat
ion loss: 0.6180091435089707
Epoch 23: Train err: 0.1975, Train loss: 0.4229888718128204 |Validation err: 0.2955, Validati
on loss: 0.6098165139555931
Epoch 24: Train err: 0.186375, Train loss: 0.4044830322265625 |Validation err: 0.3115, Valida
tion loss: 0.6406115358695388
Epoch 25: Train err: 0.184, Train loss: 0.395404550075531 |Validation err: 0.3055, Validation
loss: 0.6312467521056533
Epoch 26: Train err: 0.17075, Train loss: 0.3846285363435745 |Validation err: 0.3035, Validat
ion loss: 0.6940741082653403
Epoch 27: Train err: 0.1635, Train loss: 0.36605922937393187 |Validation err: 0.311, Validati
on loss: 0.6734102126210928
Epoch 28: Train err: 0.15925, Train loss: 0.3596725949048996 |Validation err: 0.3175, Validat
ion loss: 0.6605491498485208
Epoch 29: Train err: 0.15125, Train loss: 0.3407430748939514 |Validation err: 0.312, Validati
on loss: 0.7553919861093163
Epoch 30: Train err: 0.141, Train loss: 0.3213317505121231 |Validation err: 0.303, Validation
loss: 0.7728157490491867
Finished Training
Total time elapsed: 132.43 seconds
```

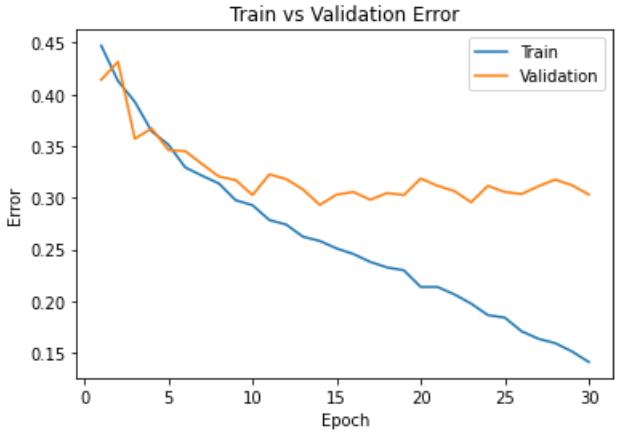
Part (e) - 2pt

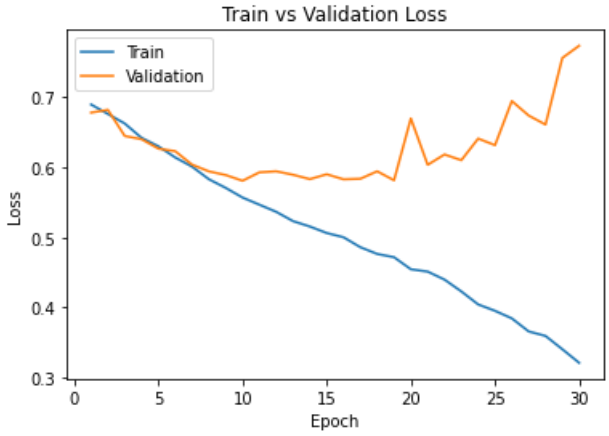
Use the function `plot_training_curve` to display the trajectory of the training/validation error and the training/validation loss. You will need to use the function `get_model_name` to generate the argument to the `plot_training_curve` function.

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

In [16]:
```
small_model_path = get_model_name("small", batch_size=64, learning_rate=0.01, epoch=29)
plot_training_curve(small_model_path)
```

In [17]:
```
large_model_path = get_model_name("large", batch_size=64, learning_rate=0.01, epoch=29)
plot_training_curve(large_model_path)
```





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.

In [18]:
```
# The training curve for small_net fluctuates a lot more than the training curve
# for large_net. The large_net network begins with a far greater training error
# and loss than the small_net, due to the amount of initial parameters, but
# steadily converges to a better performance than the small_net. The large_net
# training curve doesn't fluctuate much. There is also not a lot of overall
# decrease in the small_net training curve.

# The small_net training curve is an example of underfitting. The training and
# validation loss does not change by much as the training continues. The
# difference initial beteween the beginning and end loss/error of both curves
# are not significant. The large_net training curve is an example of
# overfitting, since the validation error and loss curves back upwards around
# the 25th epoch, even though the training error and loss decreases. This
# indicates that the model is not properly generalizing to unobserved data.
```

Part 3. Optimization Parameters [12 pt]

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

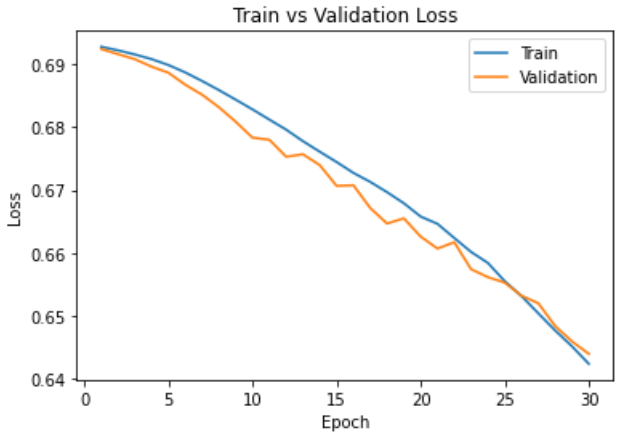
Part (a) - 3pt

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 [19]:

```
# 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)
large_model_path = get_model_name("large", batch_size=64, learning_rate=0.001, epoch=29)
plot_training_curve(large_model_path)

# The training took a shorter amount of time, but not enough to note a
# significant change. Learning rate should, theoretically, not effect the speed
# of training, but rather the amount in which the weights are effected by the
# loss at each epoch.

# The lowered learning rate slowed the speed of the convergence so much that the
# final training and validation loss/error that the model achieves is far from
# the optimum achieved with the learning rate of 0.01. The curve, however, is
# steady and consistent. Although the nature of the curve is an example of good
# training i.e. the training and validation loss/error decreasing consistently,
# the changes are too slow and small each epoch for the final model to have good
# predictions.
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.47625, Train loss: 0.6928360013961792 |Validation err: 0.467, Validatio
n loss: 0.6924686580896378
Epoch 2: Train err: 0.448625, Train loss: 0.6922589712142945 |Validation err: 0.4305, Validat
ion loss: 0.691649341955781
Epoch 3: Train err: 0.43575, Train loss: 0.6916067280769348 |Validation err: 0.4285, Validati
on loss: 0.690854424610734
Epoch 4: Train err: 0.43, Train loss: 0.690861343383789 |Validation err: 0.424, Validation lo
ss: 0.6896595880389214
Epoch 5: Train err: 0.434125, Train loss: 0.6899195008277893 |Validation err: 0.4195, Validat
ion loss: 0.6886935643851757
Epoch 6: Train err: 0.43575, Train loss: 0.6887411961555481 |Validation err: 0.4195, Validati
on loss: 0.6867824867367744
Epoch 7: Train err: 0.437125, Train loss: 0.6873774147033691 |Validation err: 0.4185, Validat
ion 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, Validati
on loss: 0.6808880660682917
Epoch 10: Train err: 0.424, Train loss: 0.6828502931594849 |Validation err: 0.408, Validation
loss: 0.6783502567559481
Epoch 11: Train err: 0.425375, Train loss: 0.6812348766326904 |Validation err: 0.4125, Valida
tion loss: 0.6780214440077543
Epoch 12: Train err: 0.42, Train loss: 0.6796319708824158 |Validation err: 0.4125, Validation
loss: 0.6753159202635288
Epoch 13: Train err: 0.414875, Train loss: 0.6777918744087219 |Validation err: 0.415, Validat
ion loss: 0.6757059413939714
Epoch 14: Train err: 0.412375, Train loss: 0.6761112003326416 |Validation err: 0.412, Validat
ion loss: 0.6739734839648008
Epoch 15: Train err: 0.40925, Train loss: 0.674472680568695 |Validation err: 0.415, Validatio
n loss: 0.6706762500107288
Epoch 16: Train err: 0.406375, Train loss: 0.6727448840141297 |Validation err: 0.4105, Valida
tion loss: 0.6707733049988747
Epoch 17: Train err: 0.4015, Train loss: 0.6713076601028443 |Validation err: 0.4045, Validati
on loss: 0.6671545393764973
Epoch 18: Train err: 0.3995, Train loss: 0.6696742882728577 |Validation err: 0.4055, Validati
on loss: 0.6646782550960779
Epoch 19: Train err: 0.40075, Train loss: 0.6679086356163025 |Validation err: 0.396, Validati
on loss: 0.6655019577592611
Epoch 20: Train err: 0.392375, Train loss: 0.665787980556488 |Validation err: 0.405, Validati
on loss: 0.6626011095941067
Epoch 21: Train err: 0.38975, Train loss: 0.6646300601959229 |Validation err: 0.394, Validati
on loss: 0.660687854513526
Epoch 22: Train err: 0.388875, Train loss: 0.662373058795929 |Validation err: 0.393, Validati
on loss: 0.6616998575627804
Epoch 23: Train err: 0.38425, Train loss: 0.6601516346931458 |Validation err: 0.3975, Validat
ion loss: 0.6573981791734695
Epoch 24: Train err: 0.382375, Train loss: 0.6584009389877319 |Validation err: 0.386, Validat
ion loss: 0.6561364810913801
Epoch 25: Train err: 0.37875, Train loss: 0.6554971766471863 |Validation err: 0.388, Validati
on loss: 0.6552744228392839
Epoch 26: Train err: 0.376625, Train loss: 0.6531173253059387 |Validation err: 0.3875, Valida
tion loss: 0.6531743723899126
Epoch 27: Train err: 0.375, Train loss: 0.6503696331977844 |Validation err: 0.387, Validation
loss: 0.6519789285957813
Epoch 28: Train err: 0.371375, Train loss: 0.6476435809135437 |Validation err: 0.3875, Valida
tion loss: 0.6483502741903067
Epoch 29: Train err: 0.368375, Train loss: 0.6451257643699646 |Validation err: 0.3825, Valida
tion loss: 0.6459067314863205
Epoch 30: Train err: 0.362625, Train loss: 0.6423329524993896 |Validation err: 0.3785, Valida
tion loss: 0.6439237017184496
Finished Training
Total time elapsed: 126.57 seconds
```



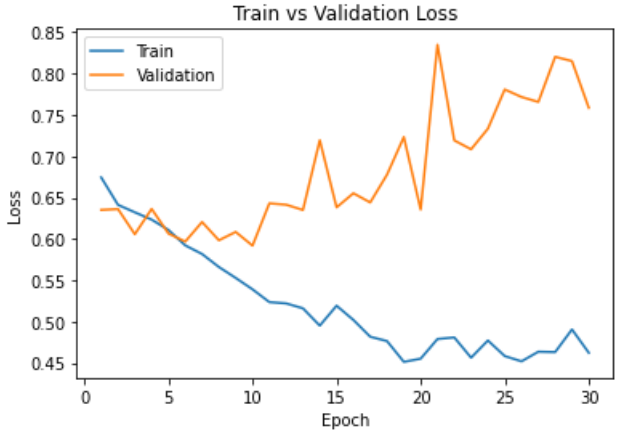
Part (b) - 3pt

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

In [20]:

```
large_net = LargeNet()

train_net(large_net, learning_rate=0.1)
large_model_path = get_model_name("large", batch_size=64, learning_rate=0.1, epoch=29)
plot_training_curve(large_model_path)

# Just like the previous training, raising the learning rate should not effect
# the training time by a significant amount, since it'll only raise the amount
# of change to the weights at each epoch. The training time did decrease in
# comparison to the training time of learning rate 0.01, but not by a
# significant amount.

# The increased learning rate caused the training curves to be a lot more
# erratic. The training finishes with a lower training loss/error than the
# previous model, but it sacrificed model performance against the validation
# set. Neither the validation loss nor error indicates any significant decreases
# or improvements on performance.
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.4295, Train loss: 0.67437779712677 |Validation err: 0.3595, Validation
loss: 0.6350857093930244
Epoch 2: Train err: 0.36075, Train loss: 0.6411805458068848 |Validation err: 0.3535, Validati
on loss: 0.6361209936439991
Epoch 3: Train err: 0.365125, Train loss: 0.6321813461780548 |Validation err: 0.3385, Validat
ion loss: 0.6056603882461786
Epoch 4: Train err: 0.352625, Train loss: 0.6233456182479858 |Validation err: 0.3575, Validat
ion loss: 0.6362800188362598
Epoch 5: Train err: 0.34075, Train loss: 0.6108013873100281 |Validation err: 0.3305, Validati
on loss: 0.6064918786287308
Epoch 6: Train err: 0.323375, Train loss: 0.5921835997104645 |Validation err: 0.317, Validati
on loss: 0.5967769594863057
Epoch 7: Train err: 0.3145, Train loss: 0.5817317583560944 |Validation err: 0.3365, Validatio
n loss: 0.6204487886279821
Epoch 8: Train err: 0.29825, Train loss: 0.5660300073623658 |Validation err: 0.3285, Validati
on loss: 0.5983372200280428
Epoch 9: Train err: 0.290875, Train loss: 0.552809501171112 |Validation err: 0.3315, Validati
on loss: 0.6084455158561468
Epoch 10: Train err: 0.278625, Train loss: 0.539032607793808 |Validation err: 0.306, Validati
on loss: 0.5918631898239255
Epoch 11: Train err: 0.272375, Train loss: 0.5236025826931 |Validation err: 0.33, Validation
loss: 0.6430060230195522
Epoch 12: Train err: 0.267375, Train loss: 0.5220149435997009 |Validation err: 0.2925, Valida
tion loss: 0.6413561534136534
Epoch 13: Train err: 0.266, Train loss: 0.5160510110855102 |Validation err: 0.3125, Validatio
n loss: 0.6349832843989134
Epoch 14: Train err: 0.24875, Train loss: 0.4951590054035187 |Validation err: 0.3145, Validat
ion loss: 0.7193072671070695
Epoch 15: Train err: 0.264625, Train loss: 0.519231944322586 |Validation err: 0.314, Validati
on loss: 0.6381420725956559
Epoch 16: Train err: 0.252625, Train loss: 0.5020012385845184 |Validation err: 0.3225, Valida
tion loss: 0.6551959458738565
Epoch 17: Train err: 0.23875, Train loss: 0.481714787364006 |Validation err: 0.357, Validatio
n loss: 0.6440742611885071
Epoch 18: Train err: 0.23375, Train loss: 0.47645506453514097 |Validation err: 0.3375, Valida
tion loss: 0.6777342790737748
Epoch 19: Train err: 0.218125, Train loss: 0.45134368968009947 |Validation err: 0.3445, Valid
ation loss: 0.7232250478118658
Epoch 20: Train err: 0.217875, Train loss: 0.45516350817680357 |Validation err: 0.3245, Valid
ation loss: 0.6354950983077288
Epoch 21: Train err: 0.23275, Train loss: 0.47897080445289614 |Validation err: 0.3255, Valida
tion loss: 0.8348110988736153
Epoch 22: Train err: 0.234875, Train loss: 0.4808810565471649 |Validation err: 0.334, Validat
ion loss: 0.7191346418112516
Epoch 23: Train err: 0.21575, Train loss: 0.4563647754192352 |Validation err: 0.316, Validati
on loss: 0.7083508176729083
Epoch 24: Train err: 0.2355, Train loss: 0.47718250966072084 |Validation err: 0.327, Validati
on loss: 0.7333047650754452
Epoch 25: Train err: 0.22025, Train loss: 0.4583414270877838 |Validation err: 0.3315, Validat
ion loss: 0.7806987538933754
Epoch 26: Train err: 0.209625, Train loss: 0.4519626965522766 |Validation err: 0.3435, Valida
tion loss: 0.7715998776257038
Epoch 27: Train err: 0.22175, Train loss: 0.4636160457134247 |Validation err: 0.3215, Validat
ion loss: 0.7656293725594878
Epoch 28: Train err: 0.219375, Train loss: 0.46314777398109436 |Validation err: 0.348, Valida
tion loss: 0.8202023077756166
Epoch 29: Train err: 0.235875, Train loss: 0.49053542733192446 |Validation err: 0.326, Valida
tion 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: 126.72 seconds
```



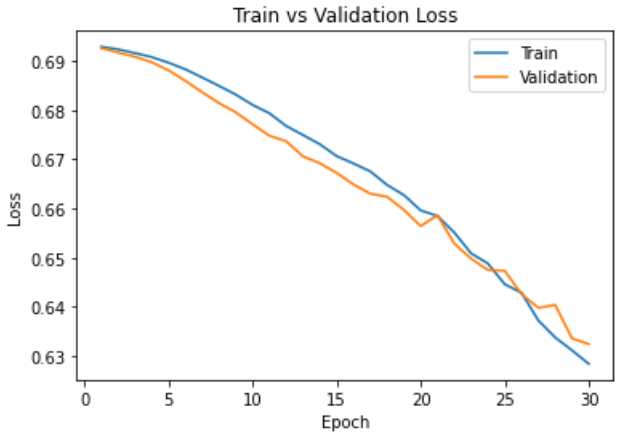
Part (c) - 3pt

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 [21]:

```
large_net = LargeNet()

train_net(large_net, batch_size=512)
large_model_path = get_model_name("large", batch_size=512, learning_rate=0.01, epoch=29)
plot_training_curve(large_model_path)

# The model takes a lot less time to train, since each batch size is larger and
# requires less iterations to complete 1 epoch.

# Increasing the batch size allows the descent to be a lot smoother, as the
# gradient being calculated at each iteration is more generalized. Due to the
# larger batch size, however, there are fewer iterations and fewer updates to
# the weight parameters, causing the overall training to be "slower". With a
# larger batch size, the training requires more epochs to achieve an optimum
# that might've been achieved with a smaller batch size like 64.
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.48175, Train loss: 0.6929379552602768 |Validation err: 0.478, Validatio
n loss: 0.6926824003458023
Epoch 2: Train err: 0.457625, Train loss: 0.6924104019999504 |Validation err: 0.434, Validati
on 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, Validati
on 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: 0.4285, Validation
loss: 0.686011865735054
Epoch 7: Train err: 0.439375, Train loss: 0.6866871677339077 |Validation err: 0.426, Validati
on loss: 0.6836968809366226
Epoch 8: Train err: 0.43525, Train loss: 0.6849770769476891 |Validation err: 0.4115, Validati
on loss: 0.6814671903848648
Epoch 9: Train err: 0.42375, Train loss: 0.6832009293138981 |Validation err: 0.414, Validatio
n 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 err: 0.4095, Valida
tion loss: 0.6748111099004745
Epoch 12: Train err: 0.41475, Train loss: 0.6768048219382763 |Validation err: 0.412, Validati
on loss: 0.6737060546875
Epoch 13: Train err: 0.4105, Train loss: 0.6749702803790569 |Validation err: 0.412, Validatio
n loss: 0.6706101596355438
Epoch 14: Train err: 0.407125, Train loss: 0.6730880849063396 |Validation err: 0.4125, Valida
tion loss: 0.6692148000001907
Epoch 15: Train err: 0.4005, Train loss: 0.6706806942820549 |Validation err: 0.4105, Validati
on loss: 0.667252704501152
Epoch 16: Train err: 0.397625, Train loss: 0.6691771410405636 |Validation err: 0.405, Validat
ion loss: 0.6649097055196762
Epoch 17: Train err: 0.393875, Train loss: 0.6675694733858109 |Validation err: 0.401, Validat
ion loss: 0.6630224883556366
Epoch 18: Train err: 0.393, Train loss: 0.6648042872548103 |Validation err: 0.3945, Validatio
n loss: 0.6624014377593994
Epoch 19: Train err: 0.38625, Train loss: 0.662746611982584 |Validation err: 0.388, Validatio
n loss: 0.6597220152616501
Epoch 20: Train err: 0.38175, Train loss: 0.6596181839704514 |Validation err: 0.4005, Validat
ion loss: 0.6564337313175201
Epoch 21: Train err: 0.38575, Train loss: 0.6584899798035622 |Validation err: 0.3885, Validat
ion loss: 0.6586423963308334
Epoch 22: Train err: 0.378125, Train loss: 0.655123382806778 |Validation err: 0.3855, Validat
ion loss: 0.6528600305318832
Epoch 23: Train err: 0.372125, Train loss: 0.6508794128894806 |Validation err: 0.3835, Valida
tion loss: 0.6497963815927505
Epoch 24: Train err: 0.37675, Train loss: 0.6488028429448605 |Validation err: 0.385, Validati
on loss: 0.6474899500608444
Epoch 25: Train err: 0.368625, Train loss: 0.6445869170129299 |Validation err: 0.382, Validat
ion loss: 0.6473268568515778
Epoch 26: Train err: 0.372625, Train loss: 0.6428566053509712 |Validation err: 0.3745, Valida
tion loss: 0.6425703465938568
Epoch 27: Train err: 0.359375, Train loss: 0.6372117549180984 |Validation err: 0.379, Validat
ion loss: 0.6397799849510193
Epoch 28: Train err: 0.35425, Train loss: 0.6337667480111122 |Validation err: 0.3695, Validat
ion loss: 0.6403783112764359
Epoch 29: Train err: 0.3535, Train loss: 0.6311353109776974 |Validation err: 0.366, Validatio
n loss: 0.6335585117340088
Epoch 30: Train err: 0.353, Train loss: 0.6283832415938377 |Validation err: 0.3675, Validatio
n loss: 0.6324127316474915
Finished Training
Total time elapsed: 111.71 seconds
```



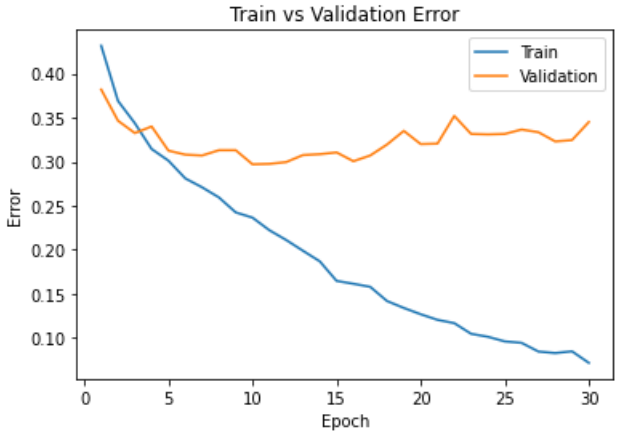
Part (d) - 3pt

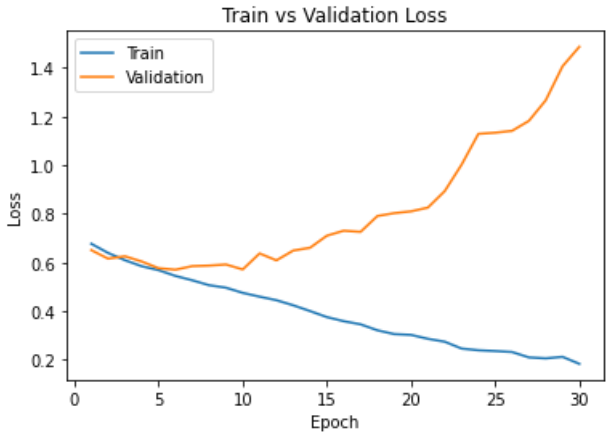
Train `large_net` with all default parameters, including with `learning_rate=0.01`. Now, set `batch_size=16`. Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *decreasing* the batch size.

In [22]:

```
large_net = LargeNet()

train_net(large_net, batch_size=16)
large_model_path = get_model_name("large", batch_size=16, learning_rate=0.01, epoch=29)
plot_training_curve(large_model_path)

# The model takes a lot more time to train, since each batch size is smaller and
# requires more iterations to complete 1 epoch.

# Due to the smaller batch size, there are far more iterations and updates to
# the weight parameters, causing the overall training to be "faster". With a
# smaller batch size, the model becomes overfit to the training data. This is
# reflected in the validation curve, as the validation loss increases as
# training loss decreases.
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.43175, Train loss: 0.6774994022846222 |Validation err: 0.382, Validatio
n loss: 0.6513170118331909
Epoch 2: Train err: 0.369, Train loss: 0.639639899969101 |Validation err: 0.3465, Validation
loss: 0.6161113576889038
Epoch 3: Train err: 0.34375, Train loss: 0.6098222947120666 |Validation err: 0.3325, Validati
on loss: 0.6260210764408112
Epoch 4: Train err: 0.314375, Train loss: 0.5849691489338875 |Validation err: 0.34, Validatio
n loss: 0.6044013917446136
Epoch 5: Train err: 0.301125, Train loss: 0.5689119303822517 |Validation err: 0.3125, Validat
ion 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, Validati
on loss: 0.5854293291568756
Epoch 8: Train err: 0.259375, Train loss: 0.5070905526578426 |Validation err: 0.313, Validati
on loss: 0.5877130818367005
Epoch 9: Train err: 0.242375, Train loss: 0.4968344421982765 |Validation err: 0.313, Validati
on loss: 0.5922425072193146
Epoch 10: Train err: 0.236375, Train loss: 0.4756101597249508 |Validation err: 0.297, Validat
ion loss: 0.5718690166473389
Epoch 11: Train err: 0.222125, Train loss: 0.4599769461452961 |Validation err: 0.2975, Valida
tion loss: 0.6376970833539963
Epoch 12: Train err: 0.211, Train loss: 0.4454492371380329 |Validation err: 0.2995, Validatio
n loss: 0.609202565908432
Epoch 13: Train err: 0.19875, Train loss: 0.4245421719551086 |Validation err: 0.3075, Validat
ion loss: 0.6494987765550614
Epoch 14: Train err: 0.18675, Train loss: 0.4007472907453775 |Validation err: 0.3085, Validat
ion loss: 0.6610016552209854
Epoch 15: Train err: 0.1645, Train loss: 0.3759974058121443 |Validation err: 0.3105, Validati
on loss: 0.7106090537309646
Epoch 16: Train err: 0.16125, Train loss: 0.3591455406397581 |Validation err: 0.3005, Validat
ion loss: 0.7310364942550659
Epoch 17: Train err: 0.15775, Train loss: 0.3463234790861607 |Validation err: 0.307, Validati
on loss: 0.7263009325265884
Epoch 18: Train err: 0.141625, Train loss: 0.32175366275012496 |Validation err: 0.3195, Valid
ation loss: 0.7913952842950821
Epoch 19: Train err: 0.13375, Train loss: 0.30618105667084455 |Validation err: 0.335, Validat
ion loss: 0.8032052783966065
Epoch 20: Train err: 0.126625, Train loss: 0.3029071792438626 |Validation err: 0.32, Validati
on loss: 0.8106685240268707
Epoch 21: Train err: 0.12025, Train loss: 0.28682796490937473 |Validation err: 0.3205, Valida
tion loss: 0.8259474284648896
Epoch 22: Train err: 0.1165, Train loss: 0.27489088076353074 |Validation err: 0.352, Validati
on loss: 0.8937610774040222
Epoch 23: Train err: 0.104375, Train loss: 0.2467898527495563 |Validation err: 0.3315, Valida
tion loss: 1.0021928198337555
Epoch 24: Train err: 0.101, Train loss: 0.23970085787773132 |Validation err: 0.331, Validatio
n loss: 1.1290796399116516
Epoch 25: Train err: 0.09575, Train loss: 0.23643119425699116 |Validation err: 0.3315, Valida
tion loss: 1.1338514368534087
Epoch 26: Train err: 0.094125, Train loss: 0.2325953512713313 |Validation err: 0.3365, Valida
tion loss: 1.1414263204336166
Epoch 27: Train err: 0.08425, Train loss: 0.21040759468451142 |Validation err: 0.3335, Valida
tion loss: 1.1823678107261657
Epoch 28: Train err: 0.0825, Train loss: 0.20643112615589052 |Validation err: 0.323, Validati
on loss: 1.266836181640625
Epoch 29: Train err: 0.0845, Train loss: 0.21273409337876364 |Validation err: 0.3245, Validat
ion loss: 1.406717705130577
Epoch 30: Train err: 0.071375, Train loss: 0.18387044295761734 |Validation err: 0.345, Valida
tion loss: 1.4871552000045776
Finished Training
Total time elapsed: 182.33 seconds
```





Part 4. Hyperparameter Search [6 pt]

Part (a) - 2pt

Based on the plots from above, choose another set of values for the hyperparameters (network, batch_size, learning_rate) that you think would help you improve the validation accuracy. Justify your choice.

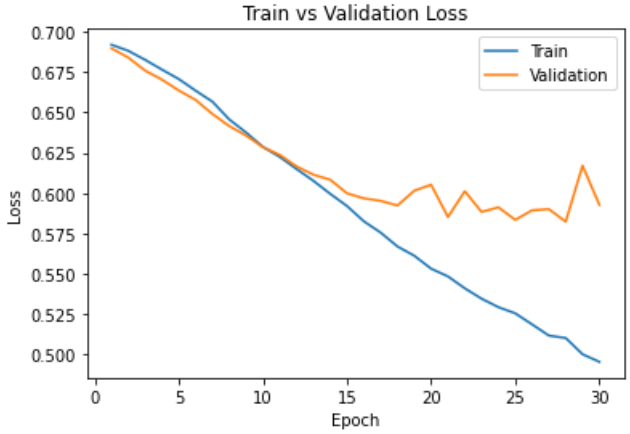
In [23]:
```
# I think using the large network, smaller batch size, and slower learning rate
# would help improve the validation accuracy. The smaller batch size would
# introduce noise and help converge on a global optimum, instead of local
# minima. The slower learning rate would help converge more carefully, as
# the noise from the smaller batch size might cause the training to be too
# too erratic. Working with the larger network also allows more diverse sets of
# parameters to be trained.
```

Part (b) - 1pt

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

In [24]:
```
large_net = LargeNet()

train_net(large_net, batch_size=16, learning_rate=0.001)
large_model_path = get_model_name("large", batch_size=16, learning_rate=0.001, epoch=29)
plot_training_curve(large_model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.454875, Train loss: 0.6919687836170196 |Validation err: 0.4335, Validat
ion loss: 0.6896940703392029
Epoch 2: Train err: 0.441875, Train loss: 0.6881972500085831 |Validation err: 0.4155, Validat
ion loss: 0.6840243172645569
Epoch 3: Train err: 0.425, Train loss: 0.6826031126976013 |Validation err: 0.4065, Validation
loss: 0.6759178109169006
Epoch 4: Train err: 0.413375, Train loss: 0.676551971077919 |Validation err: 0.411, Validatio
n loss: 0.6703446621894836
Epoch 5: Train err: 0.403, Train loss: 0.6706955729722976 |Validation err: 0.405, Validation
loss: 0.6636172285079956
Epoch 6: Train err: 0.388375, Train loss: 0.6634951171875 |Validation err: 0.388, Validation
loss: 0.6577684187889099
Epoch 7: Train err: 0.38325, Train loss: 0.6566604214906693 |Validation err: 0.387, Validatio
n loss: 0.6490317993164062
Epoch 8: Train err: 0.372625, Train loss: 0.6456744936108589 |Validation err: 0.3795, Validat
ion loss: 0.6415270042419433
Epoch 9: Train err: 0.36425, Train loss: 0.637563487291336 |Validation err: 0.362, Validation
loss: 0.6356466286182404
Epoch 10: Train err: 0.353375, Train loss: 0.6286305035948754 |Validation err: 0.3565, Valida
tion loss: 0.6284913148880005
Epoch 11: Train err: 0.348875, Train loss: 0.6224352505207061 |Validation err: 0.343, Validat
ion loss: 0.6236779315471649
Epoch 12: Train err: 0.3395, Train loss: 0.6149048793315888 |Validation err: 0.3335, Validati
on loss: 0.6165166666507721
Epoch 13: Train err: 0.337625, Train loss: 0.6076011454463005 |Validation err: 0.3345, Valida
tion loss: 0.6113973023891449
Epoch 14: Train err: 0.326, Train loss: 0.5995380475521087 |Validation err: 0.332, Validation
loss: 0.6082773017883301
Epoch 15: Train err: 0.318625, Train loss: 0.5919455865621567 |Validation err: 0.3175, Valida
tion loss: 0.5998473017215729
Epoch 16: Train err: 0.31475, Train loss: 0.5824866974949837 |Validation err: 0.3165, Validat
ion loss: 0.596711287021637
Epoch 17: Train err: 0.304625, Train loss: 0.5754690542817116 |Validation err: 0.305, Validat
ion loss: 0.5951672253608704
Epoch 18: Train err: 0.301, Train loss: 0.5668730340003967 |Validation err: 0.318, Validation
loss: 0.5922847588062287
Epoch 19: Train err: 0.295125, Train loss: 0.5611101251840591 |Validation err: 0.332, Validat
ion loss: 0.6014690661430359
Epoch 20: Train err: 0.2875, Train loss: 0.5531035642623902 |Validation err: 0.312, Validatio
n loss: 0.605187665939331
Epoch 21: Train err: 0.289375, Train loss: 0.5483110781908035 |Validation err: 0.304, Validat
ion loss: 0.5851338193416595
Epoch 22: Train err: 0.280125, Train loss: 0.5409212954640389 |Validation err: 0.3235, Valida
tion loss: 0.6011423192024231
Epoch 23: Train err: 0.276875, Train loss: 0.534481340944767 |Validation err: 0.303, Validati
on loss: 0.5883027715682984
Epoch 24: Train err: 0.27025, Train loss: 0.5292306969761849 |Validation err: 0.297, Validati
on loss: 0.5911655931472778
Epoch 25: Train err: 0.263875, Train loss: 0.5254164955615998 |Validation err: 0.294, Validat
ion loss: 0.5833725001811981
Epoch 26: Train err: 0.2605, Train loss: 0.5185559968054294 |Validation err: 0.3065, Validati
on loss: 0.5893082580566407
Epoch 27: Train err: 0.2565, Train loss: 0.5116140705645085 |Validation err: 0.288, Validatio
n loss: 0.5900122969150543
Epoch 28: Train err: 0.259125, Train loss: 0.5101234393119812 |Validation err: 0.3015, Valida
tion loss: 0.582257523059845
Epoch 29: Train err: 0.247875, Train loss: 0.5000545628666878 |Validation err: 0.3205, Valida
tion loss: 0.616970046043396
Epoch 30: Train err: 0.248875, Train loss: 0.49533707863092424 |Validation err: 0.2995, Valid
ation loss: 0.5927074880599975
Finished Training
Total time elapsed: 181.47 seconds
```



Part (c) - 2pt

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

In [25]:
```
# I think using the large network, larger batch size, and faster learning rate
# could be another way to improve validation accuracy. This is the other extreme
# from the previous set of hyperparameters, by introducing noise through the
# learning rate, and trying to counterbalance with the larger batch size.
```

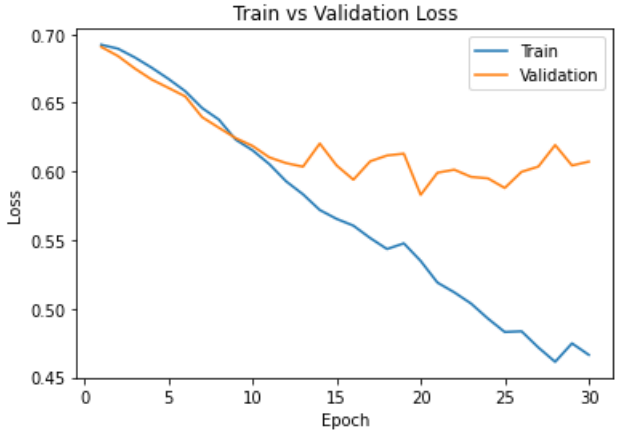
Part (d) - 1pt

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

In [26]:

```
large_net = LargeNet()

train_net(large_net, batch_size=512, learning_rate=0.05)
large_model_path = get_model_name("large", batch_size=512, learning_rate=0.05, epoch=29)
plot_training_curve(large_model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.460375, Train loss: 0.692389577627182 |Validation err: 0.428, Validatio
n loss: 0.6908226907253265
Epoch 2: Train err: 0.450875, Train loss: 0.6895152404904366 |Validation err: 0.4225, Validat
ion loss: 0.6841378808021545
Epoch 3: Train err: 0.430125, Train loss: 0.6831201873719692 |Validation err: 0.4165, Validat
ion loss: 0.6750585436820984
Epoch 4: Train err: 0.41775, Train loss: 0.6756713278591633 |Validation err: 0.4075, Validati
on loss: 0.6670383214950562
Epoch 5: Train err: 0.401625, Train loss: 0.667489554733038 |Validation err: 0.393, Validatio
n loss: 0.6609293520450592
Epoch 6: Train err: 0.385125, Train loss: 0.6585139334201813 |Validation err: 0.3795, Validat
ion loss: 0.6547118723392487
Epoch 7: Train err: 0.372125, Train loss: 0.6462636440992355 |Validation err: 0.377, Validati
on loss: 0.6396276503801346
Epoch 8: Train err: 0.36675, Train loss: 0.6378163807094097 |Validation err: 0.368, Validatio
n loss: 0.631873294711113
Epoch 9: Train err: 0.34675, Train loss: 0.6231794133782387 |Validation err: 0.35, Validation
loss: 0.6242963373661041
Epoch 10: Train err: 0.341625, Train loss: 0.6154387779533863 |Validation err: 0.341, Validat
ion loss: 0.6186475604772568
Epoch 11: Train err: 0.33375, Train loss: 0.6053492873907089 |Validation err: 0.3345, Validat
ion loss: 0.61027030646801
Epoch 12: Train err: 0.31975, Train loss: 0.5927943289279938 |Validation err: 0.3305, Validat
ion loss: 0.6060836315155029
Epoch 13: Train err: 0.310375, Train loss: 0.5833878181874752 |Validation err: 0.322, Validat
ion loss: 0.6034611314535141
Epoch 14: Train err: 0.30425, Train loss: 0.5719117373228073 |Validation err: 0.332, Validati
on loss: 0.620382159948349
Epoch 15: Train err: 0.292375, Train loss: 0.5654458925127983 |Validation err: 0.33, Validati
on loss: 0.6043886244297028
Epoch 16: Train err: 0.29225, Train loss: 0.56050194054842 |Validation err: 0.313, Validation
loss: 0.5939978957176208
Epoch 17: Train err: 0.287125, Train loss: 0.5514604449272156 |Validation err: 0.331, Validat
ion loss: 0.6074156165122986
Epoch 18: Train err: 0.28075, Train loss: 0.543450653553009 |Validation err: 0.3095, Validati
on loss: 0.6116497665643692
Epoch 19: Train err: 0.280875, Train loss: 0.5476020090281963 |Validation err: 0.3395, Valida
tion loss: 0.6130003929138184
Epoch 20: Train err: 0.27675, Train loss: 0.5347804129123688 |Validation err: 0.304, Validati
on loss: 0.582925096154213
Epoch 21: Train err: 0.2605, Train loss: 0.5189887993037701 |Validation err: 0.3185, Validati
on loss: 0.5990970432758331
Epoch 22: Train err: 0.25925, Train loss: 0.5118873845785856 |Validation err: 0.3175, Validat
ion loss: 0.601287305355072
Epoch 23: Train err: 0.24525, Train loss: 0.5036966446787119 |Validation err: 0.305, Validati
on loss: 0.5960592776536942
Epoch 24: Train err: 0.242625, Train loss: 0.4927808493375778 |Validation err: 0.3115, Valida
tion loss: 0.5949872732162476
Epoch 25: Train err: 0.235125, Train loss: 0.4829829428344965 |Validation err: 0.3055, Valida
tion loss: 0.5879651457071304
Epoch 26: Train err: 0.235125, Train loss: 0.48349691927433014 |Validation err: 0.3175, Valid
ation loss: 0.5997236669063568
Epoch 27: Train err: 0.227875, Train loss: 0.4716887269169092 |Validation err: 0.3045, Valida
tion loss: 0.603562667965889
Epoch 28: Train err: 0.224875, Train loss: 0.4612164553254843 |Validation err: 0.311, Validat
ion loss: 0.6192651689052582
Epoch 29: Train err: 0.23075, Train loss: 0.47470046021044254 |Validation err: 0.3075, Valida
tion loss: 0.604360818862915
Epoch 30: Train err: 0.227125, Train loss: 0.4663128312677145 |Validation err: 0.303, Validat
ion loss: 0.607151985168457
Finished Training
Total time elapsed: 112.14 seconds
```



Part 5. 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 [27]:
```
net = LargeNet()
model_path = get_model_name(net.name, batch_size=512, learning_rate=0.05, epoch=29)
state = torch.load(model_path)
net.load_state_dict(state)
```

Out[27]:
```
<All keys matched successfully>
```

Part (b) - 2pt

Justify your choice of model from part (a).

In [28]:
```
# This particular model had the best training and validation curve. The two
# curves decrease consistently as there are more epochs i.e. not too much
# erratic descent, no upwards increase (not overfitting), and closeness of the
# two curves (not underfitting).
```

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 [29]:
```
# 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=512)

criterion = nn.BCEWithLogitsLoss()
err, loss = evaluate(net, test_loader, criterion)

print("Test Classification Error:", err)
print("Test Classification Loss:", loss)
```

```
Files already downloaded and verified
Files already downloaded and verified
Test Classification Error: 0.299
Test Classification Loss: 0.584050253033638
```

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 [30]:
```
# The test classification error is in fact lower than the validation error, but
# for the sake of the question I will assume otherwise.

# The validation set was used to tune the hyperparameters, and thus, in some
# sort of way, the model was fit to the validation set. As a result, the
# validation set is not considered "unobserved" data anymore, and the validation
# error was reduced. The testing or holdout set is now the true "unobserved"
# data, as no part of the model has been tuned with respect to the data. The
# model's performance against unobserved data is should theoretically be poorer
# than its performance against observed data that it has been tuned for.
```

Part (e) - 2pt

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

In [31]:
```
# The test data should be used only after the model has been completely trained
# and tuned. This data is considered "unobserved", and should be presented to
# the model as little as possible, to avoid the model being biased towards the
# data to any degree.
```

Part (f) - 5pt

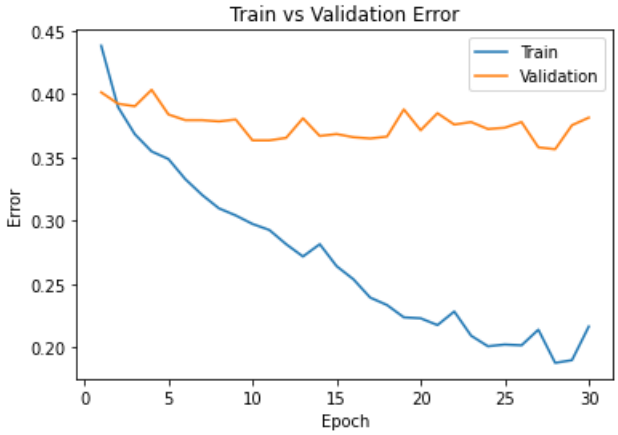
How does your best CNN model compare with an 2-layer ANN model (no convolutional layers) on classifying cat and dog images? You can use a 2-layer ANN architecture similar to what you used in Lab 1. You should explore different hyperparameter settings to determine how well you can do on the validation dataset. Once satisified with the performance, you may test it out on the test data.

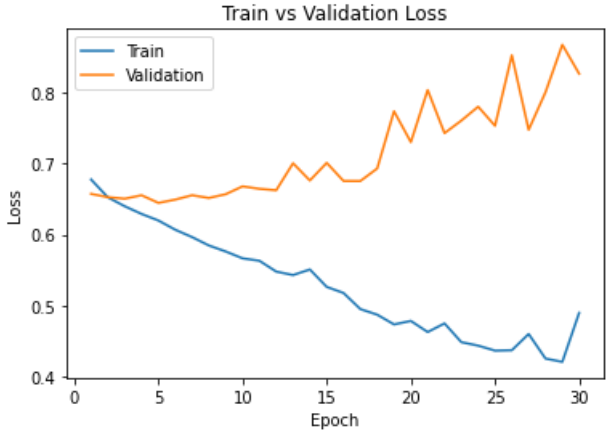
Hint: The ANN in lab 1 was applied on greyscale images. The cat and dog images are colour (RGB) and so you will need to flatted and concatinate all three colour layers before feeding them into an ANN.

In [32]:

```
torch.manual_seed(1) # set the random seed

class Pigeon(nn.Module):
    def __init__(self):
        super(Pigeon, self).__init__()
        self.name = "pigeon"
        self.layer1 = nn.Linear(3 * 32 * 32, 20)
        self.layer2 = nn.Linear(20, 1)

    def forward(self, x):
        flattened = x.view(-1, 3 * 32 * 32)
        activation1 = self.layer1(flattened)
        activation1 = F.relu(activation1)
        activation2 = self.layer2(activation1)
        return activation2.squeeze(1)

pigeon = Pigeon()

train_net(pigeon, batch_size=512, learning_rate=0.05, num_epochs=30)
ann_model_path = get_model_name("pigeon", batch_size=512, learning_rate=0.05, epoch=29)
plot_training_curve(ann_model_path)

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

criterion = nn.BCEWithLogitsLoss()
err, loss = evaluate(pigeon, test_loader, criterion)

print("Test Classification Error:", err)
print("Test Classification Loss:", loss)

# The 2-layer ANN unfortunately could not predict as well as the CNN model. The
# best test accuracy the ANN model could achieve was lower compared to the CNN
# model. This is due to the fact that CNN models use convolution, an image
# filtering technique to help recognize important features in images that
# normal ANN model's can't pick out. Thus, for specific image prediction
# problems such as this, a CNN model would work better than an ANN model.
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.438375, Train loss: 0.6775508187711239 |Validation err: 0.4015, Validat
ion loss: 0.6573085337877274
Epoch 2: Train err: 0.389875, Train loss: 0.6524141989648342 |Validation err: 0.3925, Validat
ion loss: 0.652656763792038
Epoch 3: Train err: 0.368375, Train loss: 0.639910165220499 |Validation err: 0.3905, Validati
on loss: 0.6505168825387955
Epoch 4: Train err: 0.35475, Train loss: 0.6290779784321785 |Validation err: 0.4035, Validati
on loss: 0.6553575843572617
Epoch 5: Train err: 0.34875, Train loss: 0.619865033775568 |Validation err: 0.384, Validation
loss: 0.6446329951286316
Epoch 6: Train err: 0.332875, Train loss: 0.6069212555885315 |Validation err: 0.3795, Validat
ion loss: 0.6492351591587067
Epoch 7: Train err: 0.320375, Train loss: 0.5965600870549679 |Validation err: 0.3795, Validat
ion loss: 0.6553872674703598
Epoch 8: Train err: 0.30975, Train loss: 0.5847571343183517 |Validation err: 0.3785, Validati
on loss: 0.6515602618455887
Epoch 9: Train err: 0.304125, Train loss: 0.576296504586935 |Validation err: 0.38, Validation
loss: 0.6568337678909302
Epoch 10: Train err: 0.297375, Train loss: 0.5666744336485863 |Validation err: 0.3635, Valida
tion loss: 0.66785728931427
Epoch 11: Train err: 0.292625, Train loss: 0.5632903054356575 |Validation err: 0.3635, Valida
tion loss: 0.6644353419542313
Epoch 12: Train err: 0.281375, Train loss: 0.548039223998785 |Validation err: 0.3655, Validat
ion loss: 0.6624082028865814
Epoch 13: Train err: 0.271625, Train loss: 0.5431803166866302 |Validation err: 0.381, Validat
ion loss: 0.700372040271759
Epoch 14: Train err: 0.281375, Train loss: 0.5510073266923428 |Validation err: 0.367, Validat
ion loss: 0.6761218011379242
Epoch 15: Train err: 0.264, Train loss: 0.5265394784510136 |Validation err: 0.3685, Validatio
n loss: 0.7009106278419495
Epoch 16: Train err: 0.253625, Train loss: 0.5179759189486504 |Validation err: 0.366, Validat
ion loss: 0.6754751652479172
Epoch 17: Train err: 0.239125, Train loss: 0.4953144993633032 |Validation err: 0.365, Validat
ion loss: 0.675371527671814
Epoch 18: Train err: 0.23325, Train loss: 0.48747524805366993 |Validation err: 0.3665, Valida
tion loss: 0.6929812729358673
Epoch 19: Train err: 0.2235, Train loss: 0.47376042045652866 |Validation err: 0.388, Validati
on loss: 0.7734626978635788
Epoch 20: Train err: 0.22275, Train loss: 0.47867608442902565 |Validation err: 0.3715, Valida
tion loss: 0.7300821393728256
Epoch 21: Train err: 0.217375, Train loss: 0.46305382065474987 |Validation err: 0.385, Valida
tion loss: 0.8033304512500763
Epoch 22: Train err: 0.228125, Train loss: 0.47517951764166355 |Validation err: 0.376, Valida
tion loss: 0.7426969259977341
Epoch 23: Train err: 0.209, Train loss: 0.4488253854215145 |Validation err: 0.378, Validation
loss: 0.7603601962327957
Epoch 24: Train err: 0.200625, Train loss: 0.4439147859811783 |Validation err: 0.3725, Valida
tion loss: 0.7799203395843506
Epoch 25: Train err: 0.202, Train loss: 0.4369277376681566 |Validation err: 0.3735, Validatio
n loss: 0.7530268579721451
Epoch 26: Train err: 0.201375, Train loss: 0.4374817702919245 |Validation err: 0.378, Validat
ion loss: 0.8521502315998077
Epoch 27: Train err: 0.21375, Train loss: 0.46030522882938385 |Validation err: 0.358, Validat
ion loss: 0.7474046349525452
Epoch 28: Train err: 0.1875, Train loss: 0.4257699828594923 |Validation err: 0.3565, Validati
on loss: 0.8002592176198959
Epoch 29: Train err: 0.189625, Train loss: 0.42113729007542133 |Validation err: 0.3755, Valid
ation loss: 0.8670114427804947
Epoch 30: Train err: 0.21625, Train loss: 0.4900075886398554 |Validation err: 0.3815, Validat
ion loss: 0.8261867612600327
Finished Training
Total time elapsed: 79.49 seconds
```






```
Files already downloaded and verified
Files already downloaded and verified
Test Classification Error: 0.38
Test Classification Loss: 0.8307395875453949
```