Lab 2: Cats vs Dogs

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.

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/1MgHLy8mqeaCT31Jl-Tv6X_oQlyoaTOew/view?usp=sharing

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

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

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

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

    Args:
        labels: a 1D tensor containing two possible scalar values
    Returns:
```

```
        A tensor normalize to 0/1 value
    """
    max_val = torch.max(labels)
    min_val = torch.min(labels)
    norm_labels = (labels - min_val)/(max_val - min_val)
    return norm_labels

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

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

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

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

```
    plt.legend(loc='best')
    plt.show()
```

Part 1. Visualizing the Data [7 pt]

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

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

In [ ]:
```
# 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/cifa
r-10-python.tar.gz

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

Part (a) -- 1 pt

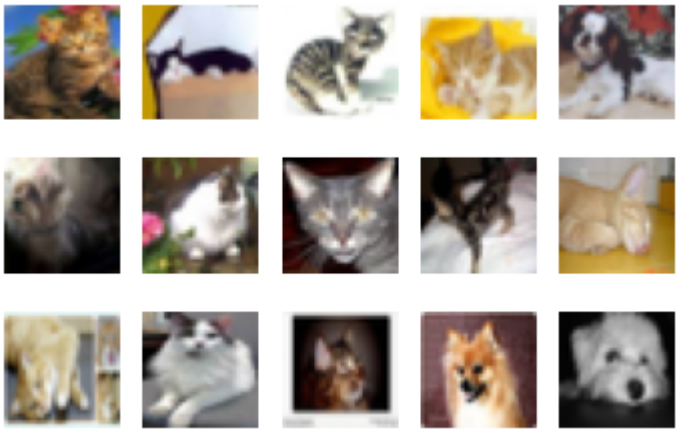
Visualize some of the data by running the code below. Include the visualization in your writeup.

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

In [ ]:
```
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 [ ]:
```
print("The number of training examples for the combined classes is",
      len(train_loader))
print("The number of validation examples for the combined classes is",
      len(val_loader))
print("The number of test examples for the combined classes is",
      len(test_loader))
```

```
The number of training examples for the combined classes is 8000
The number of validation examples for the combined classes is 2000
The number of test examples for the combined classes 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?

Answer:

A validation set is needed when training a model because we require some way to verify the results of the trained model. By tracking the validation set loss/error, we can make more informed decisions when modifying the model architecture and tuning hyperparameters. If we judge the performance of our models using the training set loss/error, we may overfit our models to the training set and not generalize well to a brand-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 [ ]:
```
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 [ ]:
```
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 [ ]:
```
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 [ ]:
```
size = 0
for param in small_net.parameters():
    print(param.shape)
    size += param.numel()
print("The total number of parameters in small_net is", size, "\n")

size = 0
```

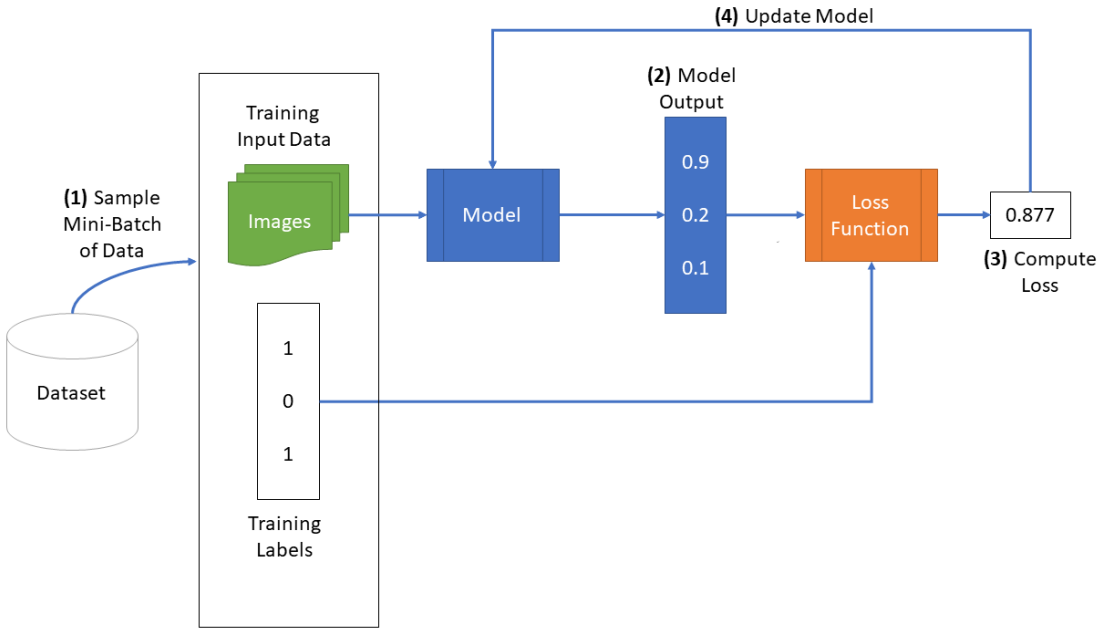
```
for param in large_net.parameters():
    print(param.shape)
    size += param.numel()
print("The total number of parameters in large_net is", size)
```

```
torch.Size([5, 3, 3, 3])
torch.Size([5])
torch.Size([1, 245])
torch.Size([1])
The total number of parameters in small_net is 386

torch.Size([5, 3, 5, 5])
torch.Size([5])
torch.Size([10, 5, 5, 5])
torch.Size([10])
torch.Size([32, 250])
torch.Size([32])
torch.Size([1, 32])
torch.Size([1])
The total number of parameters in 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 [ ]:
```
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`?

Answer:

The default values of the parameters are `batch_size=64`, `learning_rate=0.01`, and `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 [ ]:
```
train_net(small_net, num_epochs=5)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.446375, Train loss: 0.6813716783523559 |Validation err: 0.38
65, Validation loss: 0.6602997500449419
Epoch 2: Train err: 0.37325, Train loss: 0.6497629323005676 |Validation err: 0.384
5, Validation loss: 0.6575995869934559
Epoch 3: Train err: 0.359875, Train loss: 0.6388978385925292 |Validation err: 0.34
95, Validation loss: 0.6291275043040514
Epoch 4: Train err: 0.346375, Train loss: 0.6246587996482849 |Validation err: 0.35
6, Validation loss: 0.6221408396959305
Epoch 5: Train err: 0.334375, Train loss: 0.6153830280303955 |Validation err: 0.32
75, Validation loss: 0.6188967823982239
Finished Training
Total time elapsed: 18.12 seconds
```

Answer:

The files written to disk are:

- model_small_bs64_lr0.01_epoch0 - saved model checkpoint for the first epoch
- model_small_bs64_lr0.01_epoch1 - saved model checkpoint for the second epoch
- model_small_bs64_lr0.01_epoch2 - saved model checkpoint for the third epoch
- model_small_bs64_lr0.01_epoch3 - saved model checkpoint for the fourth epoch
- model_small_bs64_lr0.01_epoch4 - saved model checkpoint for the fifth epoch
- model_small_bs64_lr0.01_epoch4_train_err.csv - training error for each epoch
- model_small_bs64_lr0.01_epoch4_train_loss.csv - training loss for each epoch
- model_small_bs64_lr0.01_epoch4_val_err.csv - validation error for each epoch
- model_small_bs64_lr0.01_epoch4_val_loss.csv - validation loss for 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 [ ]:
```
# 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 [ ]:
```
small_net = SmallNet() # Reconstruct model since it was trained in Part (c)
train_net(small_net)
train_net(large_net)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.446375, Train loss: 0.6813716783523559 |Validation err: 0.38
65, Validation loss: 0.6602997500449419
Epoch 2: Train err: 0.37325, Train loss: 0.6497629323005676 |Validation err: 0.384
5, Validation loss: 0.6575995869934559
Epoch 3: Train err: 0.359875, Train loss: 0.6388978385925292 |Validation err: 0.34
95, Validation loss: 0.6291275043040514
Epoch 4: Train err: 0.346375, Train loss: 0.6246587996482849 |Validation err: 0.35
6, Validation loss: 0.6221408396959305
Epoch 5: Train err: 0.334375, Train loss: 0.6153830280303955 |Validation err: 0.32
75, Validation loss: 0.6188967823982239
Epoch 6: Train err: 0.318, Train loss: 0.6036732516288758 |Validation err: 0.339,
Validation loss: 0.6094125052914023
Epoch 7: Train err: 0.315625, Train loss: 0.5944745948314667 |Validation err: 0.32
9, Validation loss: 0.5974238961935043
Epoch 8: Train err: 0.3085, Train loss: 0.5829453563690186 |Validation err: 0.308
5, Validation loss: 0.5885121468454599
Epoch 9: Train err: 0.302, Train loss: 0.5805657277107239 |Validation err: 0.3115,
Validation loss: 0.5845186104997993
Epoch 10: Train err: 0.29975, Train loss: 0.573062111377716 |Validation err: 0.30
9, Validation loss: 0.5785001656040549
Epoch 11: Train err: 0.287375, Train loss: 0.5632161114215851 |Validation err: 0.3
14, Validation loss: 0.5821095015853643
Epoch 12: Train err: 0.292125, Train loss: 0.5567435595989227 |Validation err: 0.3
115, Validation loss: 0.5860895598307252
Epoch 13: Train err: 0.2885, Train loss: 0.5562505607604981 |Validation err: 0.30
6, Validation loss: 0.5769414035603404
Epoch 14: Train err: 0.280375, Train loss: 0.5473350758552551 |Validation err: 0.3
115, Validation loss: 0.5721263345330954
Epoch 15: Train err: 0.285, Train loss: 0.5481121215820313 |Validation err: 0.305,
Validation loss: 0.5623639700934291
Epoch 16: Train err: 0.2915, Train loss: 0.5539557900428772 |Validation err: 0.313
5, Validation loss: 0.5774335078895092
Epoch 17: Train err: 0.28075, Train loss: 0.5475348830223083 |Validation err: 0.29
95, Validation loss: 0.5680588381364942
Epoch 18: Train err: 0.279625, Train loss: 0.5440063354969025 |Validation err: 0.3
```

```
19, Validation loss: 0.576342330314219
Epoch 19: Train err: 0.27575, Train loss: 0.5402116534709931 |Validation err: 0.32
95, Validation loss: 0.606647988781333
Epoch 20: Train err: 0.2715, Train loss: 0.5385935208797454 |Validation err: 0.29
8, Validation loss: 0.5778946885839105
Epoch 21: Train err: 0.27575, Train loss: 0.540246558189392 |Validation err: 0.30
2, Validation loss: 0.5672952607274055
Epoch 22: Train err: 0.279, Train loss: 0.5399930019378663 |Validation err: 0.289
5, Validation loss: 0.5702174408361316
Epoch 23: Train err: 0.27325, Train loss: 0.5354620461463928 |Validation err: 0.30
3, Validation loss: 0.5667499387636781
Epoch 24: Train err: 0.27275, Train loss: 0.5359286315441132 |Validation err: 0.30
1, Validation loss: 0.5878297919407487
Epoch 25: Train err: 0.27325, Train loss: 0.5346703794002533 |Validation err: 0.29
7, Validation loss: 0.563475382514298
Epoch 26: Train err: 0.27025, Train loss: 0.5316284673213959 |Validation err: 0.29
85, Validation loss: 0.5694020707160234
Epoch 27: Train err: 0.270375, Train loss: 0.5298305144309997 |Validation err: 0.3
01, Validation loss: 0.578824263997376
Epoch 28: Train err: 0.269625, Train loss: 0.5351403400897979 |Validation err: 0.3
005, Validation loss: 0.5655373437330127
Epoch 29: Train err: 0.271875, Train loss: 0.5319398436546325 |Validation err: 0.2
955, Validation loss: 0.5849009975790977
Epoch 30: Train err: 0.270875, Train loss: 0.5373601081371308 |Validation err: 0.3
175, Validation loss: 0.5815494349226356
Finished Training
Total time elapsed: 109.83 seconds
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.454375, Train loss: 0.6898944606781006 |Validation err: 0.42
05, Validation loss: 0.6793290264904499
Epoch 2: Train err: 0.418875, Train loss: 0.6788924961090088 |Validation err: 0.42
15, Validation loss: 0.6790428329259157
Epoch 3: Train err: 0.40525, Train loss: 0.6685061388015747 |Validation err: 0.390
5, Validation loss: 0.6528578028082848
Epoch 4: Train err: 0.379125, Train loss: 0.6512472186088561 |Validation err: 0.39
35, Validation loss: 0.6531118471175432
Epoch 5: Train err: 0.35525, Train loss: 0.6316115870475769 |Validation err: 0.346
5, Validation loss: 0.6301526054739952
Epoch 6: Train err: 0.33675, Train loss: 0.6122225694656372 |Validation err: 0.35
2, Validation loss: 0.6261688079684973
Epoch 7: Train err: 0.3215, Train loss: 0.5984608561992645 |Validation err: 0.347
5, Validation loss: 0.6187550257891417
Epoch 8: Train err: 0.31425, Train loss: 0.5851289410591125 |Validation err: 0.320
5, Validation loss: 0.6030112085863948
Epoch 9: Train err: 0.3065, Train loss: 0.5784530780315399 |Validation err: 0.320
5, Validation loss: 0.5930909719318151
Epoch 10: Train err: 0.29425, Train loss: 0.5654694662094116 |Validation err: 0.31
5, Validation loss: 0.5877073928713799
Epoch 11: Train err: 0.28075, Train loss: 0.553060997247696 |Validation err: 0.31
6, Validation loss: 0.595877917483449
Epoch 12: Train err: 0.278625, Train loss: 0.5416067514419556 |Validation err: 0.3
1, Validation loss: 0.5843141302466393
Epoch 13: Train err: 0.273375, Train loss: 0.5358790538311005 |Validation err: 0.2
975, Validation loss: 0.5726522607728839
Epoch 14: Train err: 0.2685, Train loss: 0.5234937779903411 |Validation err: 0.296
5, Validation loss: 0.5771211478859186
Epoch 15: Train err: 0.260125, Train loss: 0.5139133958816529 |Validation err: 0.2
955, Validation loss: 0.5627784207463264
Epoch 16: Train err: 0.259375, Train loss: 0.5125633265972137 |Validation err: 0.3
205, Validation loss: 0.58364431373775
Epoch 17: Train err: 0.24925, Train loss: 0.5033850579261779 |Validation err: 0.31
1, Validation loss: 0.5711408788338304
Epoch 18: Train err: 0.248125, Train loss: 0.4913763873577118 |Validation err: 0.2
96, Validation loss: 0.5609989166259766
```

```
Epoch 19: Train err: 0.23875, Train loss: 0.4821959674358368 |Validation err: 0.30
45, Validation loss: 0.5786248967051506
Epoch 20: Train err: 0.23125, Train loss: 0.4757481541633606 |Validation err: 0.28
7, Validation loss: 0.580998913384974
Epoch 21: Train err: 0.22525, Train loss: 0.46251006150245666 |Validation err: 0.2
87, Validation loss: 0.5646722186356783
Epoch 22: Train err: 0.220375, Train loss: 0.4519791474342346 |Validation err: 0.2
855, Validation loss: 0.5754687804728746
Epoch 23: Train err: 0.21375, Train loss: 0.44430874013900756 |Validation err: 0.2
88, Validation loss: 0.5742224156856537
Epoch 24: Train err: 0.207375, Train loss: 0.43036019349098203 |Validation err: 0.
309, Validation loss: 0.6110728485509753
Epoch 25: Train err: 0.20175, Train loss: 0.4197188427448273 |Validation err: 0.29
75, Validation loss: 0.5966625260189176
Epoch 26: Train err: 0.191125, Train loss: 0.4100140264034271 |Validation err: 0.2
995, Validation loss: 0.6168392198160291
Epoch 27: Train err: 0.18725, Train loss: 0.4011841118335724 |Validation err: 0.30
35, Validation loss: 0.6397394668310881
Epoch 28: Train err: 0.177375, Train loss: 0.38736681079864504 |Validation err: 0.
3035, Validation loss: 0.614994109608233
Epoch 29: Train err: 0.170375, Train loss: 0.3770212644338608 |Validation err: 0.3
23, Validation loss: 0.7061460921540856
Epoch 30: Train err: 0.166125, Train loss: 0.3615760552883148 |Validation err: 0.3
015, Validation loss: 0.6578065417706966
Finished Training
Total time elapsed: 122.63 seconds
```

Answer:

As shown in the code above, the total time elapsed for training `small_net` is `109.83` seconds and the total time elapsed for training `large_net` is `122.63` seconds. `large_net` took longer to train because it has significantly more parameters to be updated compared to `small_net` as shown in Part (a), which also means that it has a higher model complexity or capacity.

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.

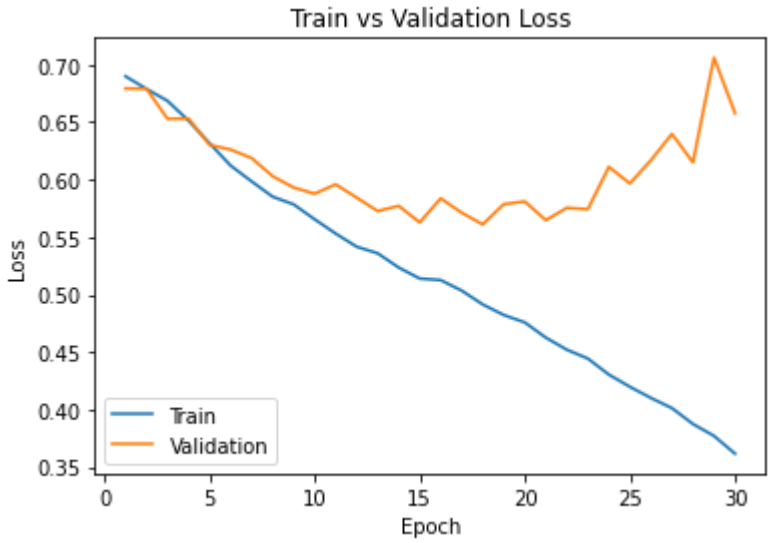
Do this for both the small network and the large network. Include both plots in your writeup.

In [ ]:
```
print("Small Network")
model_path = get_model_name("small", batch_size=64, learning_rate=0.01, epoch=29)
plot_training_curve(model_path)
```

```
Small Network
```

In [ ]:
```
print("Large Network")
model_path = get_model_name("large", batch_size=64, learning_rate=0.01, epoch=29)
plot_training_curve(model_path)
```

```
Large Network
```



Part (f) - 5pt

Describe what you notice about the training curve. How do the curves differ for `small_net` and `large_net` ? Identify any occurences of underfitting and overfitting.

Answer:

The main differences between the training curves of `small_net` and `large_net` are that:

1. The `small_net` training error decreases much more rapidly at lower epochs compared to `large_net` .
2. The `large_net` training curves are much less noisy compared to `small_net` .

For `small_net` , underfitting occurs at lower epochs (0-17) since both training error/loss and validation error/loss decrease as the number of epochs increases. Underfitting becomes less prominent for `small_net` as the number of epochs approaches 30 since the error and loss curves start to flatten out.

For `large_net` , underfitting again occurs at lower epochs (0-17) since both training error/loss and validation error/loss decrease as the number of epochs increases. However, the model starts to overfit for larger numbers of epochs (18-29) since validation error/loss flattens out and increases as training error/loss continues to decrease.

Part 3. Optimization Parameters [12 pt]

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

Part (a) - 3pt

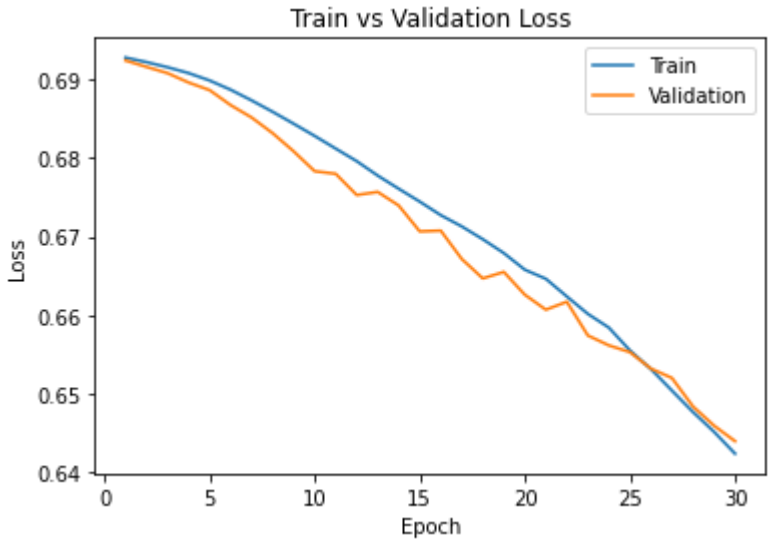
Train `large_net` with all default parameters, except set `learning_rate=0.001` . Does the model take longer/shorter to train? Plot the training curve. Describe the effect of *lowering* the learning rate.

In [ ]:
```
# 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)
model_path = get_model_name("large", batch_size=64, learning_rate=0.001, epoch=29)
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.47625, Train loss: 0.6928360013961792 |Validation err: 0.46
7, Validation loss: 0.6924686580896378
Epoch 2: Train err: 0.448625, Train loss: 0.6922589712142945 |Validation err: 0.43
05, Validation loss: 0.691649341955781
Epoch 3: Train err: 0.43575, Train loss: 0.6916067280769348 |Validation err: 0.428
5, Validation loss: 0.690854424610734
Epoch 4: Train err: 0.43, Train loss: 0.690861343383789 |Validation err: 0.424, Va
lidation loss: 0.6896595880389214
Epoch 5: Train err: 0.434125, Train loss: 0.6899195008277893 |Validation err: 0.41
95, Validation loss: 0.6886935643851757
Epoch 6: Train err: 0.43575, Train loss: 0.6887411961555481 |Validation err: 0.419
5, Validation loss: 0.6867824867367744
Epoch 7: Train err: 0.437125, Train loss: 0.6873774147033691 |Validation err: 0.41
85, Validation loss: 0.6851982977241278
Epoch 8: Train err: 0.4375, Train loss: 0.6859278454780579 |Validation err: 0.412,
Validation loss: 0.683199780061841
Epoch 9: Train err: 0.424375, Train loss: 0.6844058036804199 |Validation err: 0.41
1, Validation loss: 0.6808880660682917
Epoch 10: Train err: 0.424, Train loss: 0.6828502931594849 |Validation err: 0.408,
Validation loss: 0.6783502567559481
Epoch 11: Train err: 0.425375, Train loss: 0.6812348766326904 |Validation err: 0.4
125, Validation loss: 0.6780214440077543
Epoch 12: Train err: 0.42, Train loss: 0.6796319708824158 |Validation err: 0.4125,
Validation loss: 0.6753159202635288
Epoch 13: Train err: 0.414875, Train loss: 0.6777918744087219 |Validation err: 0.4
15, Validation loss: 0.6757059413939714
Epoch 14: Train err: 0.412375, Train loss: 0.6761112003326416 |Validation err: 0.4
12, Validation loss: 0.6739734839648008
Epoch 15: Train err: 0.40925, Train loss: 0.674472680568695 |Validation err: 0.41
5, Validation loss: 0.6706762500107288
Epoch 16: Train err: 0.406375, Train loss: 0.6727448840141297 |Validation err: 0.4
105, Validation loss: 0.6707733049988747
Epoch 17: Train err: 0.4015, Train loss: 0.6713076601028443 |Validation err: 0.404
5, Validation loss: 0.6671545393764973
Epoch 18: Train err: 0.3995, Train loss: 0.6696742882728577 |Validation err: 0.405
5, Validation loss: 0.6646782550960779
Epoch 19: Train err: 0.40075, Train loss: 0.6679086356163025 |Validation err: 0.39
6, Validation loss: 0.6655019577592611
Epoch 20: Train err: 0.392375, Train loss: 0.665787980556488 |Validation err: 0.40
5, Validation loss: 0.6626011095941067
Epoch 21: Train err: 0.38975, Train loss: 0.6646300601959229 |Validation err: 0.39
4, Validation loss: 0.660687854513526
Epoch 22: Train err: 0.388875, Train loss: 0.662373058795929 |Validation err: 0.39
3, Validation loss: 0.6616998575627804
Epoch 23: Train err: 0.38425, Train loss: 0.6601516346931458 |Validation err: 0.39
75, Validation loss: 0.6573981791734695
Epoch 24: Train err: 0.382375, Train loss: 0.6584009389877319 |Validation err: 0.3
86, Validation loss: 0.6561364810913801
Epoch 25: Train err: 0.37875, Train loss: 0.6554971766471863 |Validation err: 0.38
8, Validation loss: 0.6552744228392839
Epoch 26: Train err: 0.376625, Train loss: 0.6531173253059387 |Validation err: 0.3
875, Validation loss: 0.6531743723899126
Epoch 27: Train err: 0.375, Train loss: 0.6503696331977844 |Validation err: 0.387,
Validation loss: 0.6519789285957813
Epoch 28: Train err: 0.371375, Train loss: 0.6476435809135437 |Validation err: 0.3
875, Validation loss: 0.6483502741903067
Epoch 29: Train err: 0.368375, Train loss: 0.6451257643699646 |Validation err: 0.3
```

```
825, Validation loss: 0.6459067314863205
Epoch 30: Train err: 0.362625, Train loss: 0.6423329524993896 |Validation err: 0.3
785, Validation loss: 0.6439237017184496
Finished Training
Total time elapsed: 118.86 seconds
```



Answer:

The model takes roughly the same amount of time to train compared to the default settings.

By lowering the learning rate from `0.01` to `0.001`, the size of each gradient descent step is smaller, which makes the error/loss to decrease slower compared to the default settings. As a result, the model no longer overfits for larger numbers of epochs.

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 [ ]:
```
large_net = LargeNet()
train_net(large_net, learning_rate=0.1)
model_path = get_model_name("large", batch_size=64, learning_rate=0.1, epoch=29)
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.4295, Train loss: 0.67437779712677 |Validation err: 0.3595,
Validation loss: 0.6350857093930244
Epoch 2: Train err: 0.36075, Train loss: 0.6411805458068848 |Validation err: 0.353
5, Validation loss: 0.6361209936439991
Epoch 3: Train err: 0.365125, Train loss: 0.6321813461780548 |Validation err: 0.33
85, Validation loss: 0.6056603882461786
Epoch 4: Train err: 0.352625, Train loss: 0.6233456182479858 |Validation err: 0.35
75, Validation loss: 0.6362800188362598
Epoch 5: Train err: 0.34075, Train loss: 0.6108013873100281 |Validation err: 0.330
5, Validation loss: 0.6064918786287308
Epoch 6: Train err: 0.323375, Train loss: 0.5921835997104645 |Validation err: 0.31
7, Validation loss: 0.5967769594863057
Epoch 7: Train err: 0.3145, Train loss: 0.5817317583560944 |Validation err: 0.336
5, Validation loss: 0.6204487886279821
Epoch 8: Train err: 0.29825, Train loss: 0.5660300073623658 |Validation err: 0.328
5, Validation loss: 0.5983372200280428
Epoch 9: Train err: 0.290875, Train loss: 0.552809501171112 |Validation err: 0.331
5, Validation loss: 0.6084455158561468
Epoch 10: Train err: 0.278625, Train loss: 0.539032607793808 |Validation err: 0.30
6, Validation loss: 0.5918631898239255
Epoch 11: Train err: 0.272375, Train loss: 0.5236025826931 |Validation err: 0.33,
Validation loss: 0.6430060230195522
Epoch 12: Train err: 0.267375, Train loss: 0.5220149435997009 |Validation err: 0.2
925, Validation loss: 0.6413561534136534
Epoch 13: Train err: 0.266, Train loss: 0.5160510110855102 |Validation err: 0.312
5, Validation loss: 0.6349832843989134
Epoch 14: Train err: 0.24875, Train loss: 0.4951590054035187 |Validation err: 0.31
45, Validation loss: 0.7193072671070695
Epoch 15: Train err: 0.264625, Train loss: 0.519231944322586 |Validation err: 0.31
4, Validation loss: 0.6381420725956559
Epoch 16: Train err: 0.252625, Train loss: 0.5020012385845184 |Validation err: 0.3
225, Validation loss: 0.6551959458738565
Epoch 17: Train err: 0.23875, Train loss: 0.481714787364006 |Validation err: 0.35
7, Validation loss: 0.6440742611885071
Epoch 18: Train err: 0.23375, Train loss: 0.47645506453514097 |Validation err: 0.3
375, Validation loss: 0.6777342790737748
Epoch 19: Train err: 0.218125, Train loss: 0.45134368968009947 |Validation err: 0.
3445, Validation loss: 0.7232250478118658
Epoch 20: Train err: 0.217875, Train loss: 0.45516350817680357 |Validation err: 0.
3245, Validation loss: 0.6354950983077288
Epoch 21: Train err: 0.23275, Train loss: 0.47897080445289614 |Validation err: 0.3
255, Validation loss: 0.8348110988736153
Epoch 22: Train err: 0.234875, Train loss: 0.4808810565471649 |Validation err: 0.3
34, Validation loss: 0.7191346418112516
Epoch 23: Train err: 0.21575, Train loss: 0.4563647754192352 |Validation err: 0.31
6, Validation loss: 0.7083508176729083
Epoch 24: Train err: 0.2355, Train loss: 0.47718250966072084 |Validation err: 0.32
7, Validation loss: 0.7333047650754452
Epoch 25: Train err: 0.22025, Train loss: 0.4583414270877838 |Validation err: 0.33
15, Validation loss: 0.7806987538933754
Epoch 26: Train err: 0.209625, Train loss: 0.4519626965522766 |Validation err: 0.3
435, Validation loss: 0.7715998776257038
Epoch 27: Train err: 0.22175, Train loss: 0.4636160457134247 |Validation err: 0.32
15, Validation loss: 0.7656293725594878
Epoch 28: Train err: 0.219375, Train loss: 0.46314777398109436 |Validation err: 0.
348, Validation loss: 0.8202023077756166
Epoch 29: Train err: 0.235875, Train loss: 0.49053542733192446 |Validation err: 0.
326, Validation loss: 0.8150460105389357
Epoch 30: Train err: 0.22, Train loss: 0.4623157248497009 |Validation err: 0.3165,
Validation loss: 0.7585078496485949
Finished Training
Total time elapsed: 120.09 seconds
```

Answer:

The model takes roughly the same amount of time to train compared to the default settings.

By increasing the learning rate from `0.01` to `0.1`, the size of each gradient descent step is larger, which makes the error/loss to decrease faster compared to the default settings. As a result, the model starts to overfit much earlier compared to the default model.

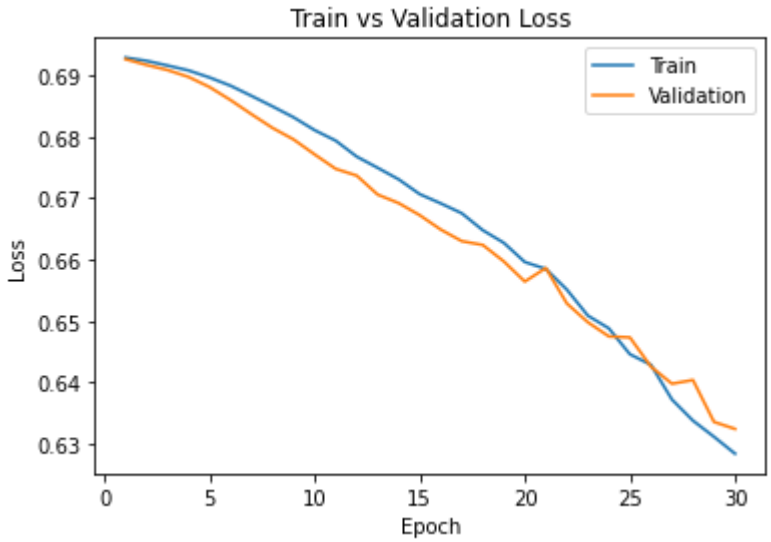
Part (c) - 3pt

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

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.48175, Train loss: 0.6929379552602768 |Validation err: 0.47
8, Validation loss: 0.6926824003458023
```

```
Epoch 2: Train err: 0.457625, Train loss: 0.6924104019999504 |Validation err: 0.43
4, Validation loss: 0.6917425245046616
Epoch 3: Train err: 0.437, Train loss: 0.6916500590741634 |Validation err: 0.4265,
Validation loss: 0.6909129917621613
Epoch 4: Train err: 0.433625, Train loss: 0.6908449940383434 |Validation err: 0.42
4, Validation loss: 0.6897870451211929
Epoch 5: Train err: 0.434, Train loss: 0.6896935552358627 |Validation err: 0.424,
Validation loss: 0.6881355047225952
Epoch 6: Train err: 0.438, Train loss: 0.688353206962347 |Validation err: 0.4285,
Validation loss: 0.686011865735054
Epoch 7: Train err: 0.439375, Train loss: 0.6866871677339077 |Validation err: 0.42
6, Validation loss: 0.6836968809366226
Epoch 8: Train err: 0.43525, Train loss: 0.6849770769476891 |Validation err: 0.411
5, Validation loss: 0.6814671903848648
Epoch 9: Train err: 0.42375, Train loss: 0.6832009293138981 |Validation err: 0.41
4, Validation loss: 0.679591491818428
Epoch 10: Train err: 0.421, Train loss: 0.6811089366674423 |Validation err: 0.416,
Validation loss: 0.6771548539400101
Epoch 11: Train err: 0.420875, Train loss: 0.6794026419520378 |Validation err: 0.4
095, Validation loss: 0.6748111099004745
Epoch 12: Train err: 0.41475, Train loss: 0.6768048219382763 |Validation err: 0.41
2, Validation loss: 0.6737060546875
Epoch 13: Train err: 0.4105, Train loss: 0.6749702803790569 |Validation err: 0.41
2, Validation loss: 0.6706101596355438
Epoch 14: Train err: 0.407125, Train loss: 0.6730880849063396 |Validation err: 0.4
125, Validation loss: 0.6692148000001907
Epoch 15: Train err: 0.4005, Train loss: 0.6706806942820549 |Validation err: 0.410
5, Validation loss: 0.667252704501152
Epoch 16: Train err: 0.397625, Train loss: 0.6691771410405636 |Validation err: 0.4
05, Validation loss: 0.6649097055196762
Epoch 17: Train err: 0.393875, Train loss: 0.6675694733858109 |Validation err: 0.4
01, Validation loss: 0.6630224883556366
Epoch 18: Train err: 0.393, Train loss: 0.6648042872548103 |Validation err: 0.394
5, Validation loss: 0.6624014377593994
Epoch 19: Train err: 0.38625, Train loss: 0.662746611982584 |Validation err: 0.38
8, Validation loss: 0.6597220152616501
Epoch 20: Train err: 0.38175, Train loss: 0.6596181839704514 |Validation err: 0.40
05, Validation loss: 0.6564337313175201
Epoch 21: Train err: 0.38575, Train loss: 0.6584899798035622 |Validation err: 0.38
85, Validation loss: 0.6586423963308334
Epoch 22: Train err: 0.378125, Train loss: 0.655123382806778 |Validation err: 0.38
55, Validation loss: 0.6528600305318832
Epoch 23: Train err: 0.372125, Train loss: 0.6508794128894806 |Validation err: 0.3
835, Validation loss: 0.6497963815927505
Epoch 24: Train err: 0.37675, Train loss: 0.6488028429448605 |Validation err: 0.38
5, Validation loss: 0.6474899500608444
Epoch 25: Train err: 0.368625, Train loss: 0.6445869170129299 |Validation err: 0.3
82, Validation loss: 0.6473268568515778
Epoch 26: Train err: 0.372625, Train loss: 0.6428566053509712 |Validation err: 0.3
745, Validation loss: 0.6425703465938568
Epoch 27: Train err: 0.359375, Train loss: 0.6372117549180984 |Validation err: 0.3
79, Validation loss: 0.6397799849510193
Epoch 28: Train err: 0.35425, Train loss: 0.6337667480111122 |Validation err: 0.36
95, Validation loss: 0.6403783112764359
Epoch 29: Train err: 0.3535, Train loss: 0.6311353109776974 |Validation err: 0.36
6, Validation loss: 0.6335585117340088
Epoch 30: Train err: 0.353, Train loss: 0.6283832415938377 |Validation err: 0.367
5, Validation loss: 0.6324127316474915
Finished Training
Total time elapsed: 104.53 seconds
```



Answer:

The model takes less time to train compared to the default settings.

By increasing the batch size from `64` to `512`, the model no longer overfits for larger numbers of epochs. However, the new model results in slightly higher training error/loss and validation error compared to the default settings.

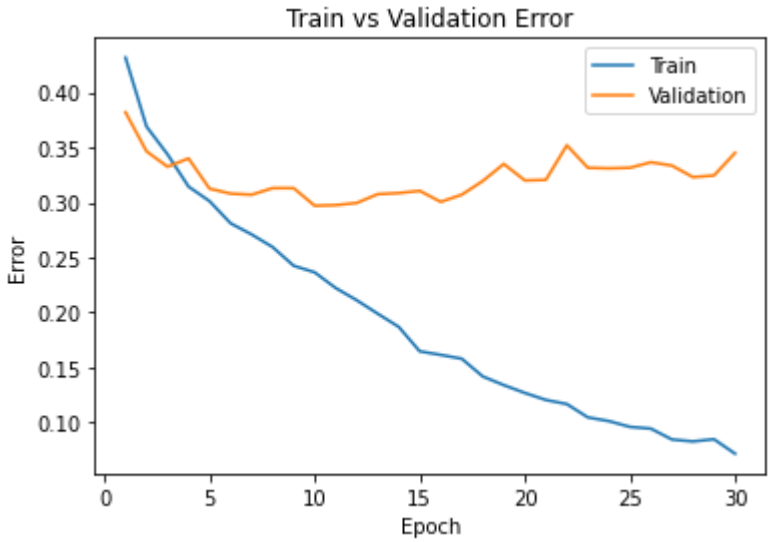
Part (d) - 3pt

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 [ ]:
```
large_net = LargeNet()
train_net(large_net, batch_size=16)
model_path = get_model_name("large", batch_size=16, learning_rate=0.01, epoch=29)
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.43175, Train loss: 0.6774994022846222 |Validation err: 0.38
2, Validation loss: 0.6513170118331909
```

```
Epoch 2: Train err: 0.369, Train loss: 0.639639899969101 |Validation err: 0.3465,
Validation loss: 0.6161113576889038
Epoch 3: Train err: 0.34375, Train loss: 0.6098222947120666 |Validation err: 0.332
5, Validation loss: 0.6260210764408112
Epoch 4: Train err: 0.314375, Train loss: 0.5849691489338875 |Validation err: 0.3
4, Validation loss: 0.6044013917446136
Epoch 5: Train err: 0.301125, Train loss: 0.5689119303822517 |Validation err: 0.31
25, Validation loss: 0.576918310880661
Epoch 6: Train err: 0.281, Train loss: 0.5452213581204415 |Validation err: 0.308,
Validation loss: 0.5708447456359863
Epoch 7: Train err: 0.270875, Train loss: 0.5272981298565864 |Validation err: 0.30
7, Validation loss: 0.5854293291568756
Epoch 8: Train err: 0.259375, Train loss: 0.5070905526578426 |Validation err: 0.31
3, Validation loss: 0.5877130818367005
Epoch 9: Train err: 0.242375, Train loss: 0.4968344421982765 |Validation err: 0.31
3, Validation loss: 0.5922425072193146
Epoch 10: Train err: 0.236375, Train loss: 0.4756101597249508 |Validation err: 0.2
97, Validation loss: 0.5718690166473389
Epoch 11: Train err: 0.222125, Train loss: 0.4599769461452961 |Validation err: 0.2
975, Validation loss: 0.6376970833539963
Epoch 12: Train err: 0.211, Train loss: 0.4454492371380329 |Validation err: 0.299
5, Validation loss: 0.609202565908432
Epoch 13: Train err: 0.19875, Train loss: 0.4245421719551086 |Validation err: 0.30
75, Validation loss: 0.6494987765550614
Epoch 14: Train err: 0.18675, Train loss: 0.4007472907453775 |Validation err: 0.30
85, Validation loss: 0.6610016552209854
Epoch 15: Train err: 0.1645, Train loss: 0.3759974058121443 |Validation err: 0.310
5, Validation loss: 0.7106090537309646
Epoch 16: Train err: 0.16125, Train loss: 0.3591455406397581 |Validation err: 0.30
05, Validation loss: 0.7310364942550659
Epoch 17: Train err: 0.15775, Train loss: 0.3463234790861607 |Validation err: 0.30
7, Validation loss: 0.7263009325265884
Epoch 18: Train err: 0.141625, Train loss: 0.32175366275012496 |Validation err: 0.
3195, Validation loss: 0.7913952842950821
Epoch 19: Train err: 0.13375, Train loss: 0.30618105667084455 |Validation err: 0.3
35, Validation loss: 0.8032052783966065
Epoch 20: Train err: 0.126625, Train loss: 0.3029071792438626 |Validation err: 0.3
2, Validation loss: 0.8106685240268707
Epoch 21: Train err: 0.12025, Train loss: 0.28682796490937473 |Validation err: 0.3
205, Validation loss: 0.8259474284648896
Epoch 22: Train err: 0.1165, Train loss: 0.27489088076353074 |Validation err: 0.35
2, Validation loss: 0.8937610774040222
Epoch 23: Train err: 0.104375, Train loss: 0.2467898527495563 |Validation err: 0.3
315, Validation loss: 1.0021928198337555
Epoch 24: Train err: 0.101, Train loss: 0.23970085787773132 |Validation err: 0.33
1, Validation loss: 1.1290796399116516
Epoch 25: Train err: 0.09575, Train loss: 0.23643119425699116 |Validation err: 0.3
315, Validation loss: 1.1338514368534087
Epoch 26: Train err: 0.094125, Train loss: 0.2325953512713313 |Validation err: 0.3
365, Validation loss: 1.1414263204336166
Epoch 27: Train err: 0.08425, Train loss: 0.21040759468451142 |Validation err: 0.3
335, Validation loss: 1.1823678107261657
Epoch 28: Train err: 0.0825, Train loss: 0.20643112615589052 |Validation err: 0.32
3, Validation loss: 1.266836181640625
Epoch 29: Train err: 0.0845, Train loss: 0.21273409337876364 |Validation err: 0.32
45, Validation loss: 1.406717705130577
Epoch 30: Train err: 0.071375, Train loss: 0.18387044295761734 |Validation err: 0.
345, Validation loss: 1.4871552000045776
Finished Training
Total time elapsed: 181.00 seconds
```



Answer:

The model takes more time to train compared to the default settings.

By decreasing the batch size from `64` to `16`, the model starts to overfit much earlier. However, the new model results in a lower training error/loss but a much higher validation loss compared to the default settings.

Part 4. Hyperparameter Search [6 pt]

Part (a) - 2pt

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

Answer:

The set of hyperparameter values I have chosen is `(large_net, batch_size=512, learning_rate=0.001)`.

I chose `large_net` because based on Part 2(f), it is observed that `large_net` is less susceptible to noise compared to `small_net`. However, it will overfit at larger numbers of epochs so the remaining hyperparameters will need to be adjusted to compensate for this.

I chose `batch_size=512` and `learning_rate=0.001` because based on Part 3(a) and Part 3(c), increasing the batch size and decreasing the learning rate can help reduce overfitting at larger numbers of epochs.

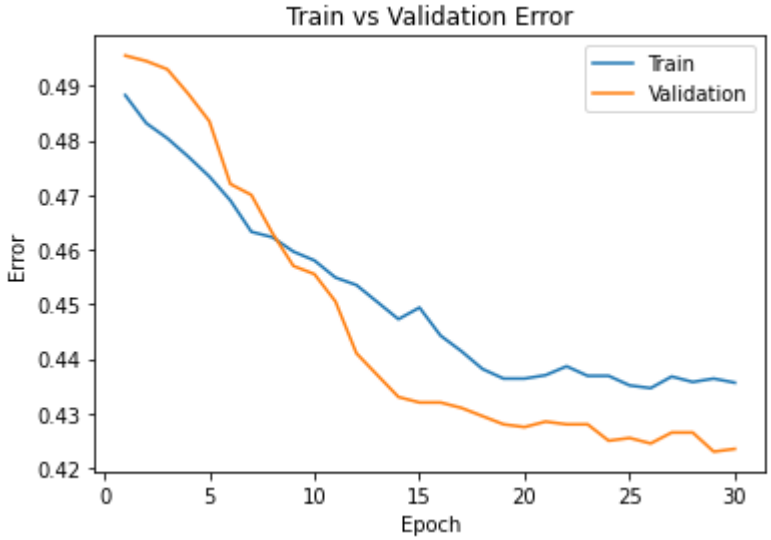
Part (b) - 1pt

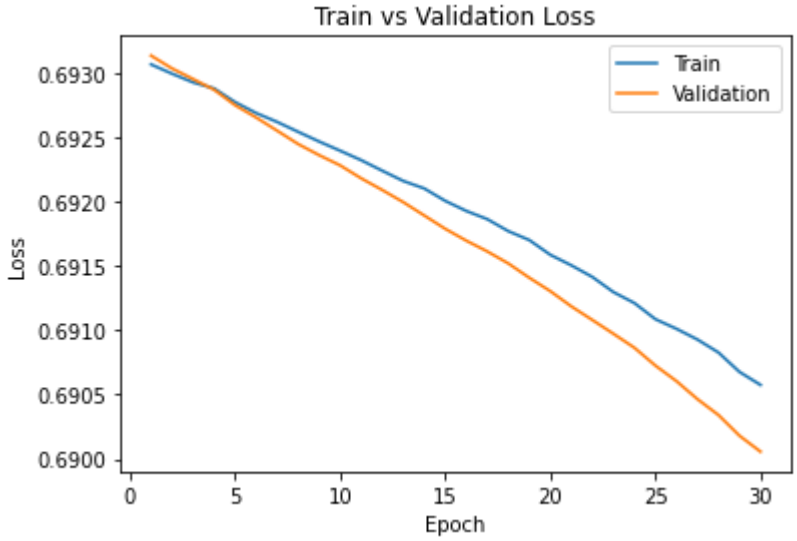
Train the model with the hyperparameters you chose in part(a), and include the training curve.

In [ ]:
```
large_net = LargeNet()
train_net(large_net, batch_size=512, learning_rate=0.001)
model_path = get_model_name("large", batch_size=512, learning_rate=0.001, epoch=29
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.48825, Train loss: 0.6930677480995655 |Validation err: 0.495
5, Validation loss: 0.6931362152099609
Epoch 2: Train err: 0.483125, Train loss: 0.692995510995388 |Validation err: 0.494
5, Validation loss: 0.6930360496044159
Epoch 3: Train err: 0.480375, Train loss: 0.6929280497133732 |Validation err: 0.49
3, Validation loss: 0.6929539889097214
Epoch 4: Train err: 0.477, Train loss: 0.6928808391094208 |Validation err: 0.4885,
Validation loss: 0.692870706319809
Epoch 5: Train err: 0.473375, Train loss: 0.692774411290884 |Validation err: 0.483
5, Validation loss: 0.6927504986524582
Epoch 6: Train err: 0.469, Train loss: 0.6926896274089813 |Validation err: 0.472,
Validation loss: 0.6926551759243011
Epoch 7: Train err: 0.46325, Train loss: 0.692620363086462 |Validation err: 0.47,
Validation loss: 0.6925524920225143
Epoch 8: Train err: 0.46225, Train loss: 0.6925435550510883 |Validation err: 0.46
3, Validation loss: 0.6924485266208649
Epoch 9: Train err: 0.459625, Train loss: 0.6924680322408676 |Validation err: 0.45
7, Validation loss: 0.6923621594905853
Epoch 10: Train err: 0.458, Train loss: 0.6923965662717819 |Validation err: 0.455
5, Validation loss: 0.6922826021909714
Epoch 11: Train err: 0.454875, Train loss: 0.6923230737447739 |Validation err: 0.4
505, Validation loss: 0.6921818852424622
Epoch 12: Train err: 0.4535, Train loss: 0.6922412514686584 |Validation err: 0.44
1, Validation loss: 0.6920914500951767
Epoch 13: Train err: 0.450375, Train loss: 0.6921614557504654 |Validation err: 0.4
37, Validation loss: 0.691996842622757
Epoch 14: Train err: 0.44725, Train loss: 0.6921032443642616 |Validation err: 0.43
3, Validation loss: 0.6918932348489761
Epoch 15: Train err: 0.449375, Train loss: 0.6920064650475979 |Validation err: 0.4
32, Validation loss: 0.6917892098426819
Epoch 16: Train err: 0.44425, Train loss: 0.6919283680617809 |Validation err: 0.43
2, Validation loss: 0.6916972398757935
Epoch 17: Train err: 0.441375, Train loss: 0.6918644718825817 |Validation err: 0.4
31, Validation loss: 0.6916135102510452
Epoch 18: Train err: 0.438125, Train loss: 0.6917712315917015 |Validation err: 0.4
295, Validation loss: 0.6915201395750046
Epoch 19: Train err: 0.436375, Train loss: 0.6917018257081509 |Validation err: 0.4
28, Validation loss: 0.6914086490869522
Epoch 20: Train err: 0.436375, Train loss: 0.6915871091187 |Validation err: 0.427
5, Validation loss: 0.6913044154644012
Epoch 21: Train err: 0.437, Train loss: 0.6915052235126495 |Validation err: 0.428
5, Validation loss: 0.6911860555410385
```

```
Epoch 22: Train err: 0.438625, Train loss: 0.6914149634540081 |Validation err: 0.4
28, Validation loss: 0.6910803616046906
Epoch 23: Train err: 0.436875, Train loss: 0.6912974379956722 |Validation err: 0.4
28, Validation loss: 0.6909734457731247
Epoch 24: Train err: 0.436875, Train loss: 0.6912120543420315 |Validation err: 0.4
25, Validation loss: 0.6908644735813141
Epoch 25: Train err: 0.435125, Train loss: 0.6910865269601345 |Validation err: 0.4
255, Validation loss: 0.6907256692647934
Epoch 26: Train err: 0.434625, Train loss: 0.6910119205713272 |Validation err: 0.4
245, Validation loss: 0.6906051337718964
Epoch 27: Train err: 0.43675, Train loss: 0.6909283325076103 |Validation err: 0.42
65, Validation loss: 0.6904648989439011
Epoch 28: Train err: 0.43575, Train loss: 0.6908275187015533 |Validation err: 0.42
65, Validation loss: 0.6903413087129593
Epoch 29: Train err: 0.436375, Train loss: 0.6906765103340149 |Validation err: 0.4
23, Validation loss: 0.6901802867650986
Epoch 30: Train err: 0.435625, Train loss: 0.6905755028128624 |Validation err: 0.4
235, Validation loss: 0.6900565475225449
Finished Training
Total time elapsed: 109.07 seconds
```





Part (c) - 2pt

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

Answer:

Based on the results of Part (a), the improved set of hyperparameter values I have chosen is `(large_net, batch_size=512, learning_rate=0.05, num_epochs=19)` .

Despite the fact that increasing the batch size and decreasing the learning rate helped reduce overfitting, the combined effect also greatly reduced the rate at which error/loss decreases. As a result, I decided to increase the learning rate to `0.05` and let the error/loss decrease more rapidly.

Increasing the learning rate introduced some minor overfitting at the larger epoch range, specifically epochs 20-30. So, I chose to reduce the number of epochs to `19` as well.

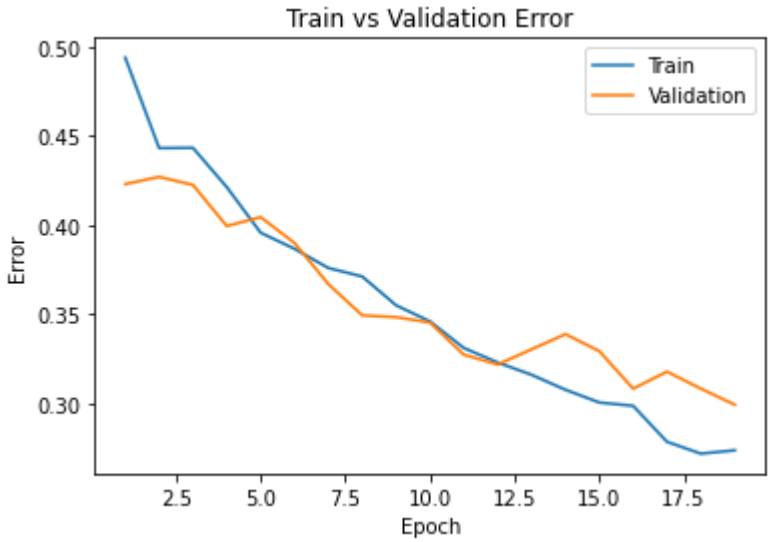
Part (d) - 1pt

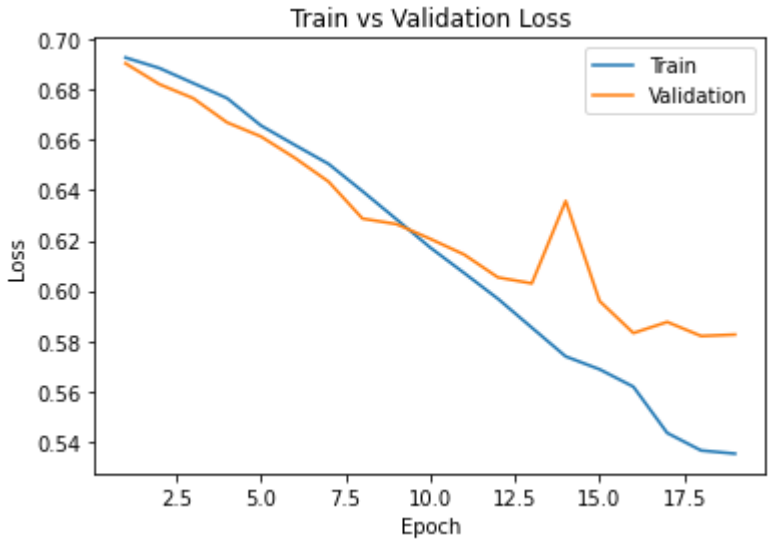
Train the model with the hyperparameters you chose in part(c), and include the training curve.

In [ ]:
```
large_net = LargeNet()
train_net(large_net, batch_size=512, learning_rate=0.05, num_epochs=19)
model_path = get_model_name("large", batch_size=512, learning_rate=0.05, epoch=18)
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.49375, Train loss: 0.6928116045892239 |Validation err: 0.42
3, Validation loss: 0.6905251741409302
Epoch 2: Train err: 0.443125, Train loss: 0.688632857054472 |Validation err: 0.42
7, Validation loss: 0.6822147369384766
Epoch 3: Train err: 0.44325, Train loss: 0.6826433055102825 |Validation err: 0.422
5, Validation loss: 0.6766358017921448
Epoch 4: Train err: 0.42125, Train loss: 0.6766663640737534 |Validation err: 0.399
5, Validation loss: 0.6669993251562119
Epoch 5: Train err: 0.39575, Train loss: 0.665775328874588 |Validation err: 0.404
5, Validation loss: 0.6614061146974564
Epoch 6: Train err: 0.38675, Train loss: 0.6580385267734528 |Validation err: 0.39,
Validation loss: 0.6528950929641724
Epoch 7: Train err: 0.376, Train loss: 0.6504689082503319 |Validation err: 0.367,
Validation loss: 0.6434173136949539
Epoch 8: Train err: 0.37125, Train loss: 0.63962047919631 |Validation err: 0.3495,
Validation loss: 0.6287297606468201
Epoch 9: Train err: 0.355125, Train loss: 0.6284837387502193 |Validation err: 0.34
85, Validation loss: 0.6265468597412109
Epoch 10: Train err: 0.346125, Train loss: 0.6172070913016796 |Validation err: 0.3
455, Validation loss: 0.6206691563129425
Epoch 11: Train err: 0.33125, Train loss: 0.6070893071591854 |Validation err: 0.32
75, Validation loss: 0.6144364476203918
Epoch 12: Train err: 0.323125, Train loss: 0.5968068800866604 |Validation err: 0.3
22, Validation loss: 0.6053186058998108
Epoch 13: Train err: 0.31625, Train loss: 0.5853234939277172 |Validation err: 0.33
05, Validation loss: 0.6029408276081085
Epoch 14: Train err: 0.307875, Train loss: 0.5738637149333954 |Validation err: 0.3
39, Validation loss: 0.6357403099536896
Epoch 15: Train err: 0.30075, Train loss: 0.5687415562570095 |Validation err: 0.32
95, Validation loss: 0.595936581492424
Epoch 16: Train err: 0.298875, Train loss: 0.5618034973740578 |Validation err: 0.3
085, Validation loss: 0.5831653028726578
Epoch 17: Train err: 0.27875, Train loss: 0.5433838963508606 |Validation err: 0.31
8, Validation loss: 0.5875828564167023
Epoch 18: Train err: 0.272125, Train loss: 0.5364223718643188 |Validation err: 0.3
085, Validation loss: 0.5819875150918961
```

```
Epoch 19: Train err: 0.274, Train loss: 0.5351713746786118 |Validation err: 0.299
5, Validation loss: 0.5825539380311966
Finished Training
Total time elapsed: 63.01 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 [ ]:
```
net = LargeNet()
model_path = get_model_name(net.name, batch_size=512, learning_rate=0.05, epoch=18
state = torch.load(model_path)
net.load_state_dict(state)
```

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

Part (b) - 2pt

Justify your choice of model from part (a).

Answer:

I chose `large_net` because based on Part 2(f), it is observed that `large_net` is less susceptible to noise compared to `small_net`. However, it will overfit at larger numbers of epochs so the remaining hyperparameters will need to be adjusted to compensate for this.

I chose `batch_size=512` because based on Part 3(a), increasing the batch size can help reduce overfitting at larger numbers of epochs and reduce model training time.

I chose `learning_rate=0.05` so the model can learn faster by letting the error/loss decrease more rapidly.

I chose `num_epochs=19` because the model overfits at greater numbers of epochs.

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 [ ]:
```
# 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("The test classification error is", err, "and the test loss is", loss)
```

```
Files already downloaded and verified
Files already downloaded and verified
The test classification error is 0.304 and the test loss is 0.5739964246749878
```

Part (d) - 3pt

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

Answer:

The test classification error is `0.304` and the validation error is `0.2995`. While the errors are quite similar, the test classification error is indeed slightly higher compared to the validation error. This is expected because the test error is an indication of how the model will perform on a new data set. The test set has only been shown to the model for the first time while the validation set has been used quite extensively when searching for the best hyperparameters.

Part (e) - 2pt

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

Answer:

The test data set is used at the very end because it is utilized to provide a realistic estimate of how the model will perform on a brand-new data set. It is important that the test data is used as little as possible because it is crucial to not make any neural network architecture decisions based on test data or test accuracy. Otherwise, bias towards the test data set will be introduced and the model will be overfitted to the test data.

Part (f) - 5pt

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

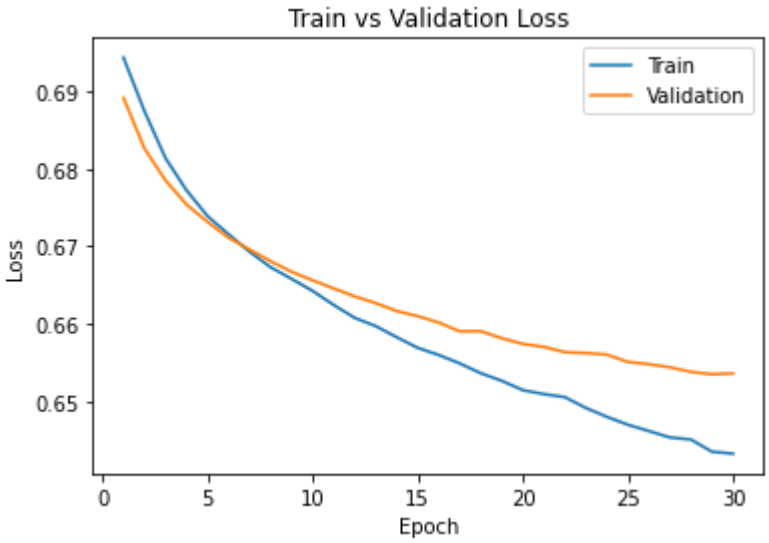
Hint: The ANN in lab 1 was applied on greyscale images. The cat and dog images are colour (RGB) and so you will need to flatted and concatinate all three colour layers before feeding them into an ANN.

In [ ]:
```
class Pigeon(nn.Module):
    def __init__(self):
        super(Pigeon, self).__init__()
        self.name = "pigeon"
        self.layer1 = nn.Linear(32 * 32 * 3, 30)
        self.layer2 = nn.Linear(30, 1)
    def forward(self, img):
        flattened = img.view(-1, 32 * 32 * 3)
        activation1 = self.layer1(flattened)
        activation1 = F.relu(activation1)
        activation2 = self.layer2(activation1)
        activation2 = activation2.squeeze(1)
        return activation2
```

In [ ]:
```
# Training
pigeon = Pigeon()
train_net(pigeon, batch_size=512, learning_rate=0.001)
model_path = get_model_name("pigeon", batch_size=512, learning_rate=0.001, epoch=2
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.487125, Train loss: 0.6943724788725376 |Validation err: 0.46
05, Validation loss: 0.689177542924881
Epoch 2: Train err: 0.456625, Train loss: 0.6874385997653008 |Validation err: 0.43
05, Validation loss: 0.6826355457305908
Epoch 3: Train err: 0.4335, Train loss: 0.6813802681863308 |Validation err: 0.42,
Validation loss: 0.6785184442996979
Epoch 4: Train err: 0.422, Train loss: 0.6772349141538143 |Validation err: 0.421,
Validation loss: 0.6753996312618256
Epoch 5: Train err: 0.4145, Train loss: 0.6739057414233685 |Validation err: 0.420
5, Validation loss: 0.673160195350647
Epoch 6: Train err: 0.4105, Train loss: 0.6715681962668896 |Validation err: 0.413
5, Validation loss: 0.6711207330226898
Epoch 7: Train err: 0.408625, Train loss: 0.6692988276481628 |Validation err: 0.4
1, Validation loss: 0.6695920825004578
Epoch 8: Train err: 0.405375, Train loss: 0.6673070974647999 |Validation err: 0.40
7, Validation loss: 0.6680852025747299
Epoch 9: Train err: 0.402, Train loss: 0.6657969579100609 |Validation err: 0.41, V
```

```
alidation loss: 0.6667183190584183
Epoch 10: Train err: 0.403125, Train loss: 0.6642512530088425 |Validation err: 0.4
035, Validation loss: 0.665614053606987
Epoch 11: Train err: 0.3985, Train loss: 0.6624419838190079 |Validation err: 0.397
5, Validation loss: 0.6645730435848236
Epoch 12: Train err: 0.3955, Train loss: 0.6607705317437649 |Validation err: 0.398
5, Validation loss: 0.6635369509458542
Epoch 13: Train err: 0.393625, Train loss: 0.6597073785960674 |Validation err: 0.3
965, Validation loss: 0.662675067782402
Epoch 14: Train err: 0.392125, Train loss: 0.6582720205187798 |Validation err: 0.3
935, Validation loss: 0.6616563946008682
Epoch 15: Train err: 0.391, Train loss: 0.6568984352052212 |Validation err: 0.395
5, Validation loss: 0.6609941273927689
Epoch 16: Train err: 0.389625, Train loss: 0.655973594635725 |Validation err: 0.39
55, Validation loss: 0.6601681262254715
Epoch 17: Train err: 0.386625, Train loss: 0.6548884995281696 |Validation err: 0.3
95, Validation loss: 0.6590256094932556
Epoch 18: Train err: 0.385875, Train loss: 0.6536205783486366 |Validation err: 0.3
935, Validation loss: 0.659050315618515
Epoch 19: Train err: 0.38525, Train loss: 0.6526415199041367 |Validation err: 0.39
25, Validation loss: 0.6581411957740784
Epoch 20: Train err: 0.38375, Train loss: 0.6514371633529663 |Validation err: 0.39
25, Validation loss: 0.6574095785617828
Epoch 21: Train err: 0.382875, Train loss: 0.6509279534220695 |Validation err: 0.3
93, Validation loss: 0.6570215225219727
Epoch 22: Train err: 0.383625, Train loss: 0.6505362801253796 |Validation err: 0.3
91, Validation loss: 0.6563445180654526
Epoch 23: Train err: 0.383375, Train loss: 0.6491378732025623 |Validation err: 0.3
93, Validation loss: 0.6562289595603943
Epoch 24: Train err: 0.38075, Train loss: 0.6479924693703651 |Validation err: 0.39
05, Validation loss: 0.6560273319482803
Epoch 25: Train err: 0.38175, Train loss: 0.6469727531075478 |Validation err: 0.39
25, Validation loss: 0.6550809890031815
Epoch 26: Train err: 0.379375, Train loss: 0.6461551114916801 |Validation err: 0.3
91, Validation loss: 0.654788002371788
Epoch 27: Train err: 0.379875, Train loss: 0.6453301012516022 |Validation err: 0.3
895, Validation loss: 0.6543925106525421
Epoch 28: Train err: 0.378, Train loss: 0.6450622528791428 |Validation err: 0.385
5, Validation loss: 0.6537979692220688
Epoch 29: Train err: 0.378, Train loss: 0.6434877142310143 |Validation err: 0.385,
Validation loss: 0.6534916311502457
Epoch 30: Train err: 0.37725, Train loss: 0.6432384327054024 |Validation err: 0.38
3, Validation loss: 0.6536015421152115
Finished Training
Total time elapsed: 62.51 seconds
```



In [ ]:
```
# Testing
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("The test classification error is", err, "and the test loss is", loss)
```

```
Files already downloaded and verified
Files already downloaded and verified
The test classification error is 0.381 and the test loss is 0.6476677060127258
```

Answer:

As shown in the code above, the best test classification error achieved using a 2-layer ANN architecture is `0.381`. This is much higher compared to the `0.304` test classification error produced using the CNN model. Therefore, a CNN architecture is more suited for this problem of cats vs dogs classification.