Lab 2: Cats vs Dogs

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

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

What to submit

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

Do not submit any other files produced by your code.

Include a link to your colab file in your submission.

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

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

Colab Link

Include a link to your colab file here

Colab Link: https://colab.research.google.com/drive/1rpm95fzp-lrar6iybxx1cBaqQ92Gorwm?usp=sharing (https://colab.research.google.com/drive/1rpm95fzp-lrar6iybxx1cBaqQ92Gorwm?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, validati
on
    and testing datasets. Returns data loaders for the three preprocessed data
sets.

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

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

    classes = ('plane', 'car', 'bird', 'cat',
               'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
    ##########################################################################
    # The output of torchvision datasets are PILImage images of range [0, 1].
    # We transform them to Tensors of normalized range [-1, 1].
    transform = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
    # Load CIFAR10 training data
```

```
    trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                            download=True, transform=transfor
m)
    # Get the list of indices to sample from
    relevant_indices = get_relevant_indices(trainset, classes, target_classes)

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

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

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

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

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

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

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

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

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

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

```
    plt.xlabel("Epoch")
    plt.ylabel("Error")
    plt.legend(loc='best')
    plt.show()
    plt.title("Train vs Validation Loss")
    plt.plot(range(1,n+1), train_loss, label="Train")
    plt.plot(range(1,n+1), val_loss, label="Validation")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.legend(loc='best')
    plt.show()
```

Part 1. Visualizing the Data [7 pt]

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

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

In [ ]:
```
# 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
```

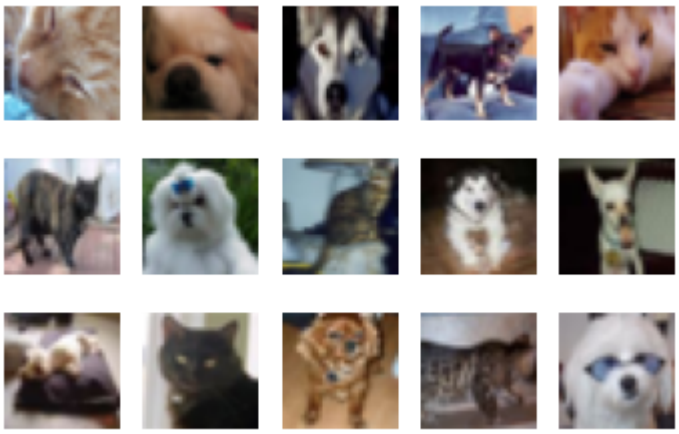
```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./dat
a/cifar-10-python.tar.gz


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

Part (a) -- 1 pt

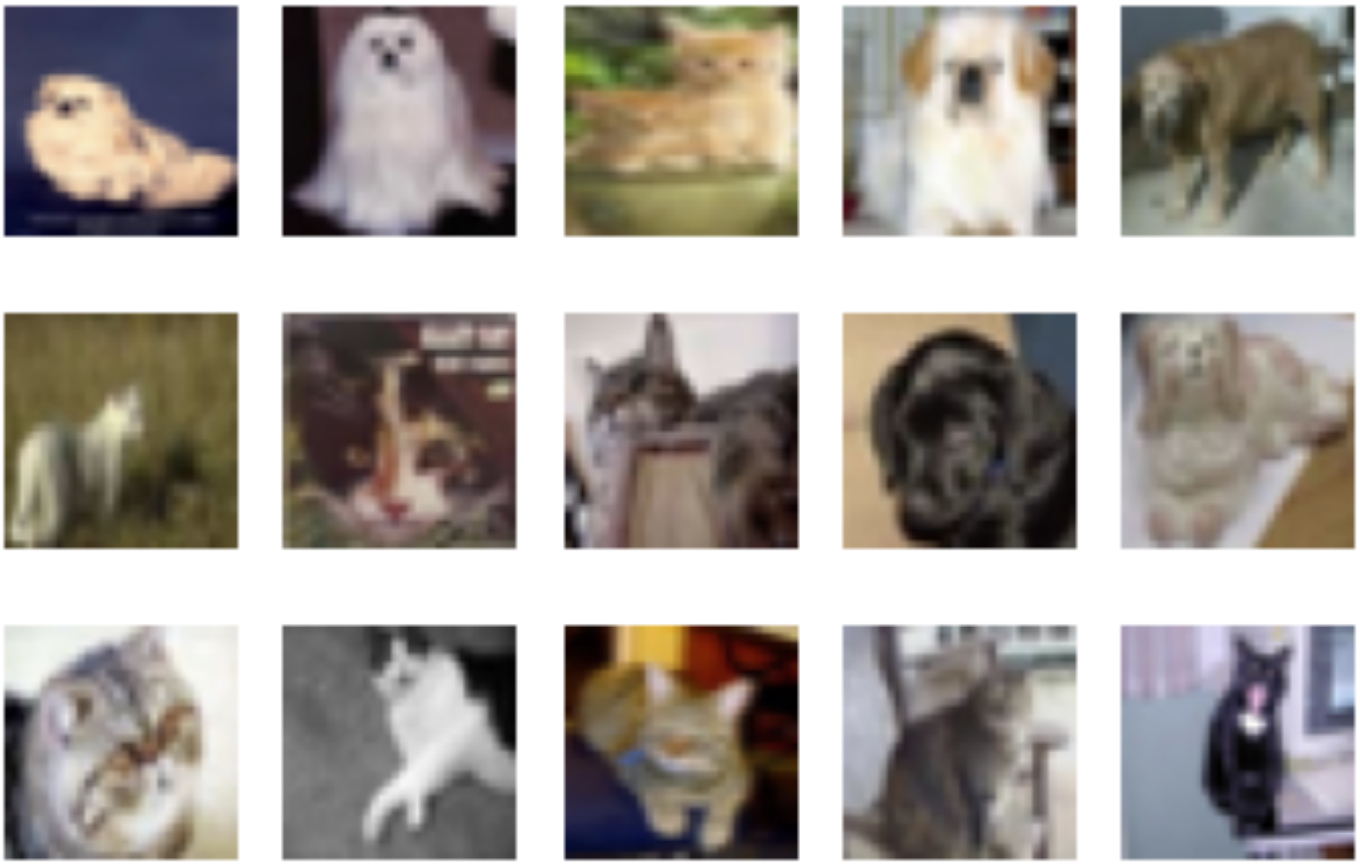
Visualize some of the data by running the code below. Include the visualization in your writeup.

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

In [ ]:
```
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
```



Visualization



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(len(train_loader), len(val_loader), len(test_loader))
# There are 8000 training examples, 2000 validation examples, and 2000 test ex
amples
```

```
8000 2000 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?

A validation set is needed when training the model to tune the hyperparameters of the model. Hyperparameters are not determined by the optimizer during training and must be tuned by experimentally/iteratively training a model multiple times and evaluating the impact on the validation accuracy to check how well the model generalizes (cannot do this with test data since it would result in overfitting to the test data, skewing the perceived accuracy of the model).

If we judge the performance of our models using the training set loss/error instead of validation set loss/error, we would end up choosing hyperparameters that cause overfitting to the training data, preventing the model from functioning well when it is actually deployed. For example, increasing the capcity of the model by increasing the number of hidden layers and neurons per layer may greatly increase training accuracy since it can more easily "memorize" the training data, but if this were to happen, the model would not generalize very well to new scenarios.

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.

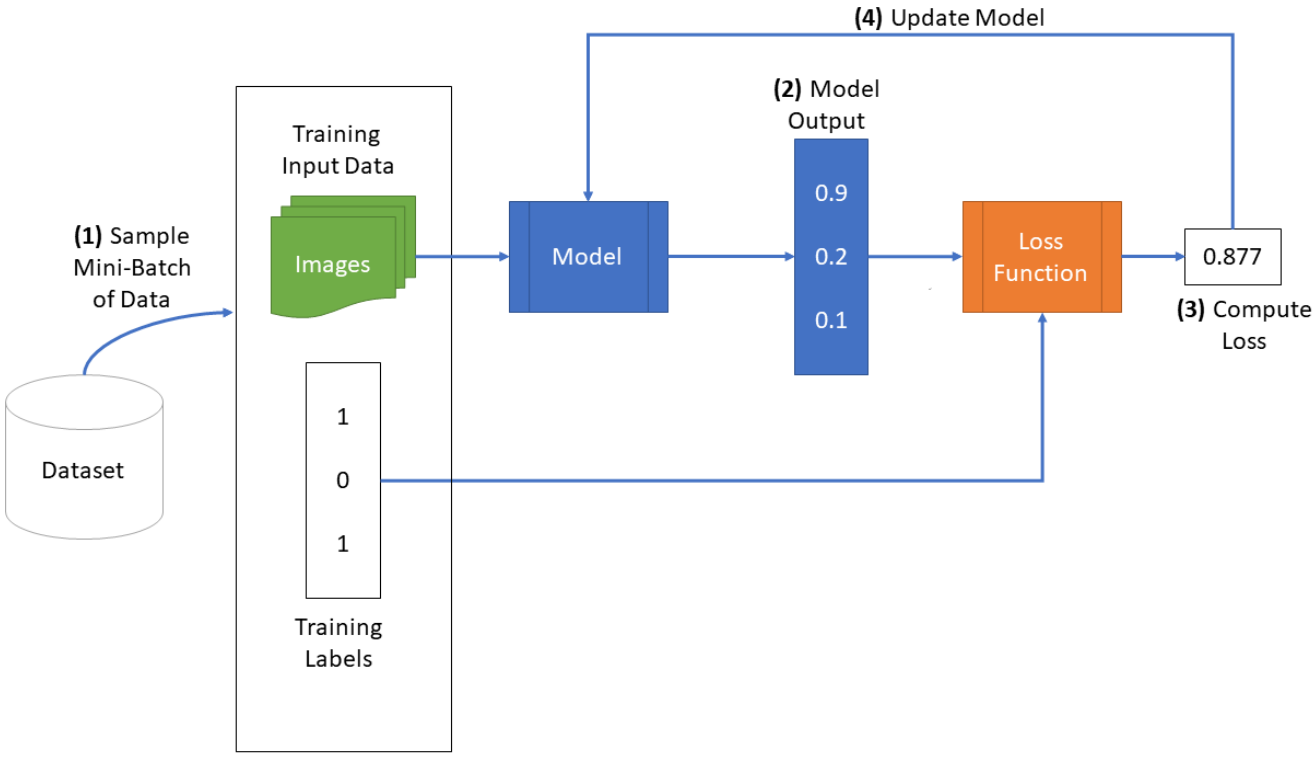
What is the total number of parameters in `small_net` and in `large_net` ? (Hint: how many numbers are in each tensor?)

In [ ]:
```
small_params, large_params = 0, 0

for i in small_net.parameters():
  small_params += i.numel()
for i in large_net.parameters():
  large_params += i.numel()

print(small_params, large_params)
# There are 386 parameters in small_net and 9705 parameters in large_net
```

```
386 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, epoc
h)
        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 [ ]:
```
# batch_size is 64, learning_rate is 0.01, and num_epochs is 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.

- The model - a file containing the state of the model (ex. parameters, buffers, etc.) so that the model can be loaded from the file elsewhere
- Training error - a csv file containing the progressive training error of the model
- Training loss - a csv file containing the progressive training loss of the model
- Validation error - a csv file containing the progressive training error of the model
- Validation loss - a csv file containing the progressive training loss of the model

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 [ ]:
```
train_net(small_net) # It took 141.4 seconds to train
train_net(large_net) # It took 151.7 seconds to train

# The large_net took longer to train because there are more computations for
# both the forward and backward passes.
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.269375, Train loss: 0.5366925568580627 |Validation err:
0.3085, Validation loss: 0.5878767343237996
Epoch 2: Train err: 0.268625, Train loss: 0.5359468927383423 |Validation err:
0.3065, Validation loss: 0.5873097376897931
Epoch 3: Train err: 0.2675, Train loss: 0.5370846486091614 |Validation err:
0.3105, Validation loss: 0.5859405416995287
Epoch 4: Train err: 0.269125, Train loss: 0.536867474079132 |Validation err:
0.3195, Validation loss: 0.5890214843675494
Epoch 5: Train err: 0.267875, Train loss: 0.5347643525600433 |Validation err:
0.3165, Validation loss: 0.587564435787499
Epoch 6: Train err: 0.269, Train loss: 0.5319040122032166 |Validation err: 0.
3145, Validation loss: 0.6016236059367657
Epoch 7: Train err: 0.275625, Train loss: 0.5373723311424256 |Validation err:
0.309, Validation loss: 0.5851427866145968
Epoch 8: Train err: 0.27, Train loss: 0.5325418524742126 |Validation err: 0.3
05, Validation loss: 0.5804283954203129
Epoch 9: Train err: 0.265875, Train loss: 0.5365991735458374 |Validation err:
0.31, Validation loss: 0.5889394041150808
Epoch 10: Train err: 0.267875, Train loss: 0.5322402195930481 |Validation er
r: 0.3045, Validation loss: 0.5761402752250433
Epoch 11: Train err: 0.26875, Train loss: 0.5337754747867585 |Validation err:
0.3165, Validation loss: 0.5889995563775301
Epoch 12: Train err: 0.270125, Train loss: 0.5297126052379608 |Validation er
r: 0.318, Validation loss: 0.6004948783665895
Epoch 13: Train err: 0.27075, Train loss: 0.5353372263908386 |Validation err:
0.3045, Validation loss: 0.586690541356802
Epoch 14: Train err: 0.269625, Train loss: 0.5316413698196412 |Validation er
r: 0.3125, Validation loss: 0.5919125936925411
Epoch 15: Train err: 0.26825, Train loss: 0.5301118726730347 |Validation err:
0.3115, Validation loss: 0.583173917606473
Epoch 16: Train err: 0.275375, Train loss: 0.5398368308544159 |Validation er
r: 0.319, Validation loss: 0.5978946518152952
Epoch 17: Train err: 0.27375, Train loss: 0.5349525728225708 |Validation err:
0.308, Validation loss: 0.5816372307017446
Epoch 18: Train err: 0.267375, Train loss: 0.5313468625545502 |Validation er
r: 0.3165, Validation loss: 0.5925112459808588
Epoch 19: Train err: 0.26625, Train loss: 0.5293733129501342 |Validation err:
0.3035, Validation loss: 0.5863998448476195
Epoch 20: Train err: 0.264875, Train loss: 0.5295597815513611 |Validation er
r: 0.3035, Validation loss: 0.5908113997429609
Epoch 21: Train err: 0.263875, Train loss: 0.5319955835342407 |Validation er
r: 0.3025, Validation loss: 0.5795717434957623
Epoch 22: Train err: 0.27, Train loss: 0.5331517338752747 |Validation err: 0.
3155, Validation loss: 0.5971522005274892
Epoch 23: Train err: 0.2685, Train loss: 0.5323400418758393 |Validation err:
0.2955, Validation loss: 0.5806593066081405
Epoch 24: Train err: 0.258625, Train loss: 0.5289957547187805 |Validation er
r: 0.308, Validation loss: 0.5817407714203
Epoch 25: Train err: 0.264125, Train loss: 0.5271837210655212 |Validation er
r: 0.297, Validation loss: 0.5781899020075798
Epoch 26: Train err: 0.26525, Train loss: 0.5270459642410278 |Validation err:
0.2945, Validation loss: 0.5827515032142401
Epoch 27: Train err: 0.26525, Train loss: 0.5261651043891906 |Validation err:
0.305, Validation loss: 0.5901663610711694
Epoch 28: Train err: 0.268875, Train loss: 0.5292545228004456 |Validation er
```

```
r: 0.301, Validation loss: 0.5832408182322979
Epoch 29: Train err: 0.26625, Train loss: 0.5303199172019959 |Validation err:
0.3105, Validation loss: 0.5887585310265422
Epoch 30: Train err: 0.267875, Train loss: 0.5321251130104065 |Validation er
r: 0.309, Validation loss: 0.5839564418420196
Finished Training
Total time elapsed: 144.87 seconds
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.148875, Train loss: 0.33484546983242036 |Validation er
r: 0.303, Validation loss: 0.6656000576913357
Epoch 2: Train err: 0.151625, Train loss: 0.3308900545835495 |Validation err:
0.303, Validation loss: 0.7200132189318538
Epoch 3: Train err: 0.136625, Train loss: 0.31600552713871 |Validation err:
0.307, Validation loss: 0.7543083345517516
Epoch 4: Train err: 0.129625, Train loss: 0.2980842932462692 |Validation err:
0.315, Validation loss: 0.7644116301089525
Epoch 5: Train err: 0.12425, Train loss: 0.2856417044401169 |Validation err:
0.307, Validation loss: 0.7456822963431478
Epoch 6: Train err: 0.109, Train loss: 0.25771701288223264 |Validation err:
0.3105, Validation loss: 0.8201885251328349
Epoch 7: Train err: 0.106375, Train loss: 0.252949555516243 |Validation err:
0.297, Validation loss: 0.8359709568321705
Epoch 8: Train err: 0.100625, Train loss: 0.23721237051486968 |Validation er
r: 0.31, Validation loss: 0.926182471215725
Epoch 9: Train err: 0.095125, Train loss: 0.22631127643585205 |Validation er
r: 0.3085, Validation loss: 0.9022348113358021
Epoch 10: Train err: 0.0885, Train loss: 0.21385653281211853 |Validation err:
0.314, Validation loss: 0.9866158310323954
Epoch 11: Train err: 0.08675, Train loss: 0.20962235468626023 |Validation er
r: 0.303, Validation loss: 0.9819177147001028
Epoch 12: Train err: 0.083, Train loss: 0.19654230278730392 |Validation err:
0.3, Validation loss: 1.00873170979321
Epoch 13: Train err: 0.0825, Train loss: 0.1966316127181053 |Validation err:
0.3175, Validation loss: 1.0660358183085918
Epoch 14: Train err: 0.070625, Train loss: 0.1759000490307808 |Validation er
r: 0.3285, Validation loss: 1.150572868064046
Epoch 15: Train err: 0.0785, Train loss: 0.18722469091415406 |Validation err:
0.318, Validation loss: 1.169174149632454
Epoch 16: Train err: 0.054625, Train loss: 0.14003385561704634 |Validation er
r: 0.31, Validation loss: 1.2371026631444693
Epoch 17: Train err: 0.0565, Train loss: 0.1425884413421154 |Validation err:
0.326, Validation loss: 1.3010713290423155
Epoch 18: Train err: 0.0555, Train loss: 0.14093617245554924 |Validation err:
0.325, Validation loss: 1.4169797133654356
Epoch 19: Train err: 0.05125, Train loss: 0.1319162645190954 |Validation err:
0.319, Validation loss: 1.4072729051113129
Epoch 20: Train err: 0.06125, Train loss: 0.15764917901158332 |Validation er
r: 0.324, Validation loss: 1.5256977677345276
Epoch 21: Train err: 0.063, Train loss: 0.1522057032585144 |Validation err:
0.3155, Validation loss: 1.393835541792214
Epoch 22: Train err: 0.044125, Train loss: 0.11111763721704483 |Validation er
r: 0.3205, Validation loss: 1.454516600817442
Epoch 23: Train err: 0.036875, Train loss: 0.10023742139339448 |Validation er
r: 0.3215, Validation loss: 1.5914091225713491
Epoch 24: Train err: 0.040125, Train loss: 0.10390109325945378 |Validation er
r: 0.3325, Validation loss: 1.6339049618691206
```

```
Epoch 25: Train err: 0.052875, Train loss: 0.1363923499882221 |Validation er
r: 0.3185, Validation loss: 1.7013208791613579
Epoch 26: Train err: 0.037875, Train loss: 0.10037516890466214 |Validation er
r: 0.3195, Validation loss: 1.859364278614521
Epoch 27: Train err: 0.0315, Train loss: 0.07972868004441261 |Validation err:
0.323, Validation loss: 1.8405686877667904
Epoch 28: Train err: 0.039, Train loss: 0.1060314309746027 |Validation err:
0.321, Validation loss: 1.7722697407007217
Epoch 29: Train err: 0.0335, Train loss: 0.08645659394562244 |Validation err:
0.33, Validation loss: 1.953849820420146
Epoch 30: Train err: 0.03275, Train loss: 0.08784305027127266 |Validation er
r: 0.3335, Validation loss: 1.7703755740076303
Finished Training
Total time elapsed: 170.43 seconds
```

Part (e) - 2pt

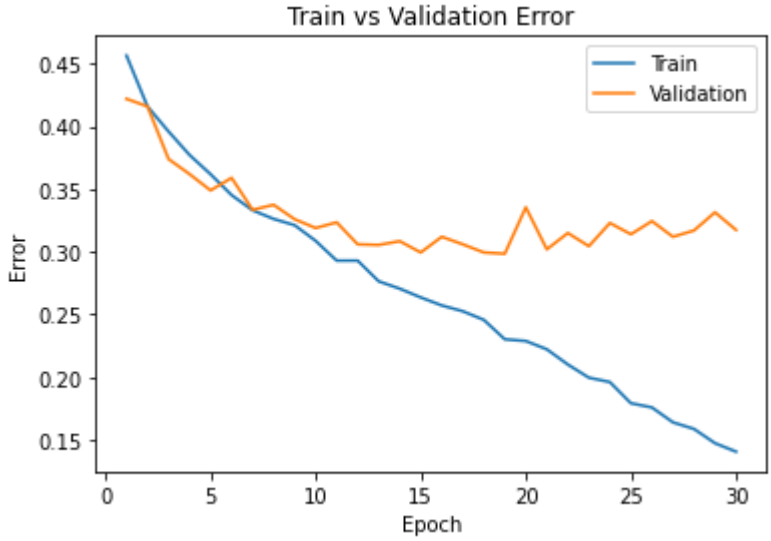
Use the function `plot_training_curve` to display the trajectory of the training/validation error and the training/validation loss. You will need to use the function `get_model_name` to generate the argument to the `plot_training_curve` function.

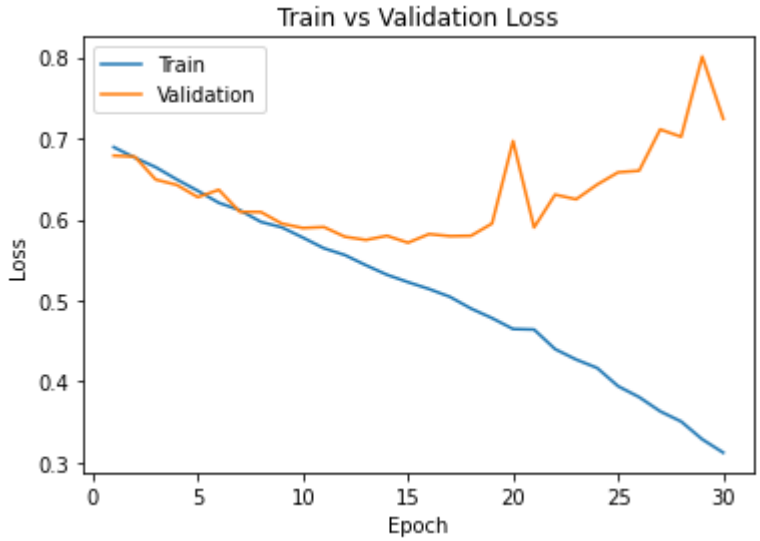
Do this for both the small network and the large network. Include both plots in your writeup.

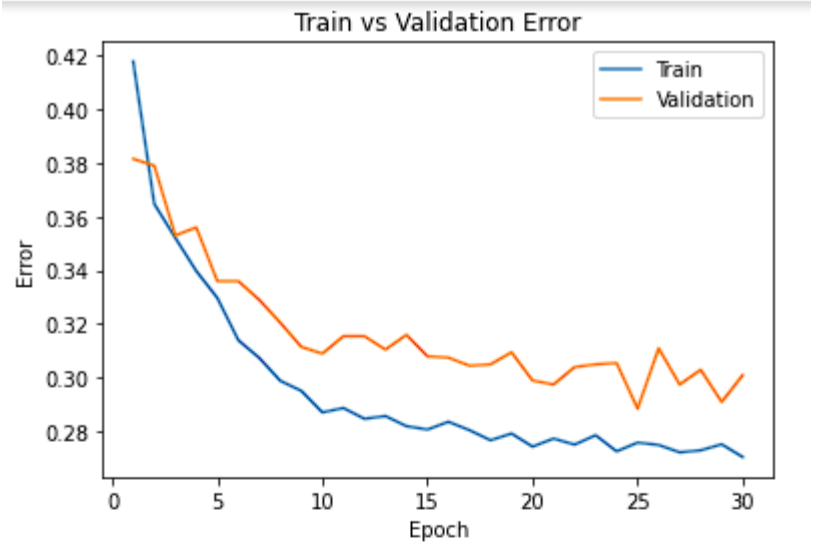
In [ ]:
```
#model_path = get_model_name("small", batch_size=??, learning_rate=??, epoch=29)
plot_training_curve(get_model_name("small", 64, 0.01, 29))
plot_training_curve(get_model_name("large", 64, 0.01, 29))
```

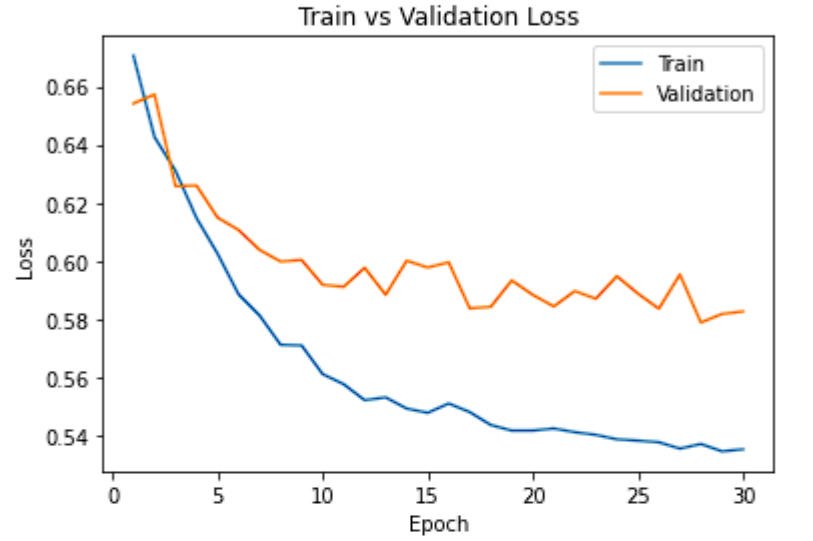

Train vs Validation Error
Train
Validation
Error
Epoch
0.42
0.40
0.38
0.36
0.34
0.32
0.30
0.28
0
5
10
15
20
25
30

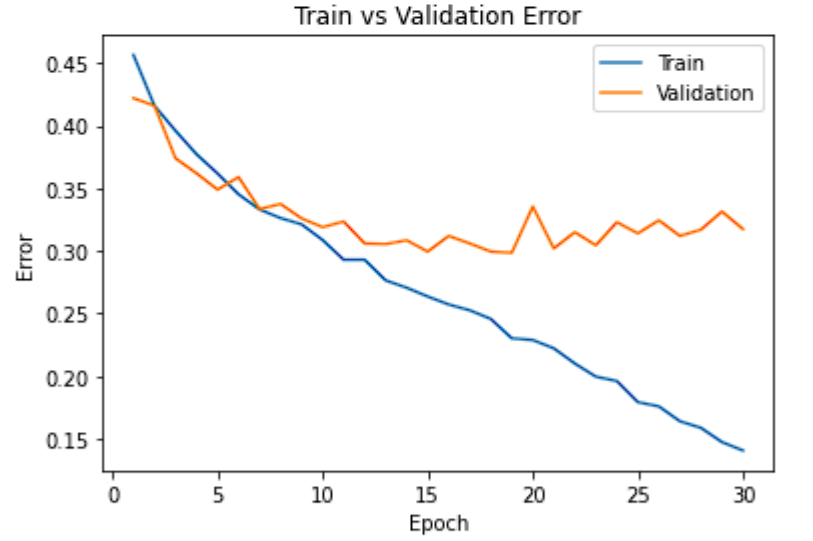

Train vs Validation Loss
Train
Validation
Loss
Epoch
0.66
0.64
0.62
0.60
0.58
0.56
0.54
0
5
10
15
20
25
30

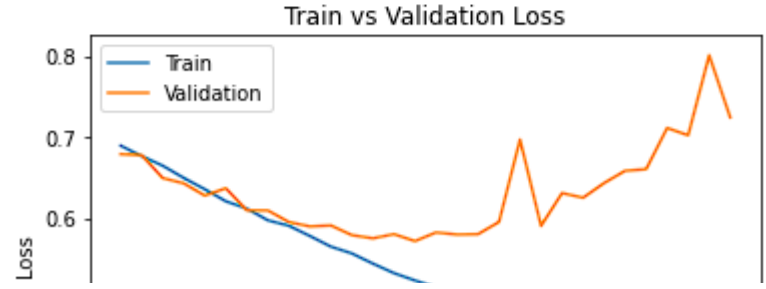

Train vs Validation Error
Train
Validation
Error
Epoch
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0
5
10
15
20
25
30

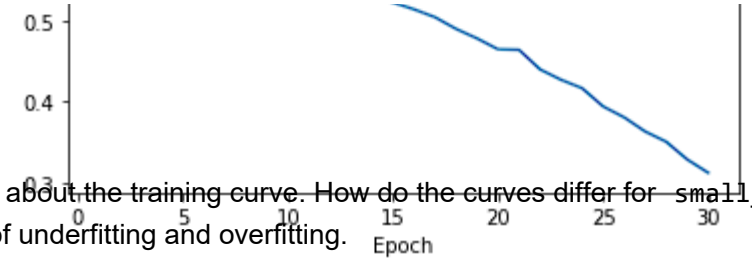

Train vs Validation Loss
Train
Validation
Loss
Epoch
0.8
0.7
0.6
0.5
0.4
0.3
0
5
10
15
20
25
30


Train vs Validation Error
Train
Validation
Error
Epoch
0.42
0.40
0.38
0.36
0.34
0.32
0.30
0.28
0
5
10
15
20
25
30


Train vs Validation Loss
Train
Validation
Loss
Epoch
0.66
0.64
0.62
0.60
0.58
0.56
0.54
0
5
10
15
20
25
30


Train vs Validation Error
Train
Validation
Error
Epoch
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0
5
10
15
20
25
30


Train vs Validation Loss
Train
Validation
Loss
0.8
0.7
0.6



Part (f) - 5pt

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

For the small net, the training error/loss curve is quadratic, it starts with a steep slope and quickly flattens out by epoch 10-15. The training error/loss for the large net however displays a very linear trend, decreasing with more epochs.

Both the small net and the large net are overfitting to the data since there is a significant gap between the training and validation error/loss. However, while the gap between the error/loss seems to remain constant for the small net with increasing epochs, the error/loss for the large net appears to be increasing exponentially/quadratically, suggesting that the large net is both more overfit than the small net when training concludes and is more susceptible to overfitting in general.

Part 3. Optimization Parameters [12 pt]

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

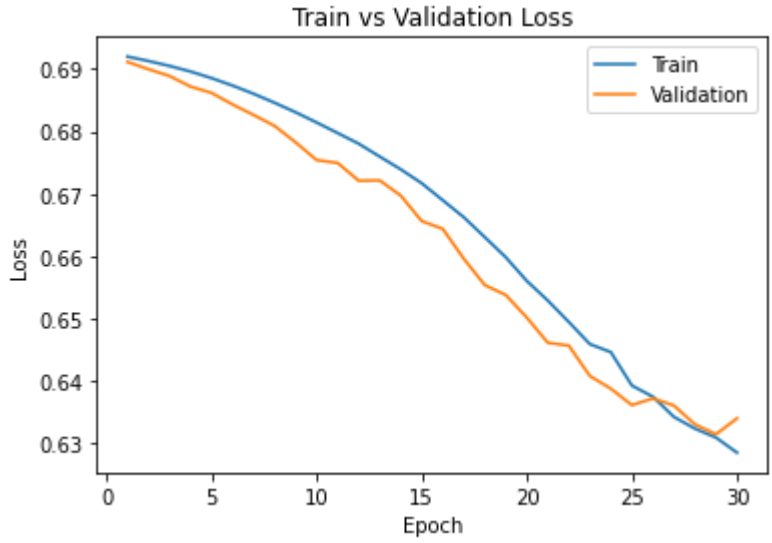
Part (a) - 3pt

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)

large_model_path = get_model_name("large", batch_size=64, 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: 0.476625, Train loss: 0.6920007181167602 |Validation err:
0.4615, Validation loss: 0.6911534927785397
Epoch 2: Train err: 0.44925, Train loss: 0.6912919111251831 |Validation err:
0.426, Validation loss: 0.6899733934551477
Epoch 3: Train err: 0.438125, Train loss: 0.6905045418739318 |Validation err:
0.4165, Validation loss: 0.6888899859040976
Epoch 4: Train err: 0.432875, Train loss: 0.6896067056655883 |Validation err:
0.4165, Validation loss: 0.6871877927333117
Epoch 5: Train err: 0.433125, Train loss: 0.6885500984191895 |Validation err:
0.415, Validation loss: 0.6861823759973049
Epoch 6: Train err: 0.433625, Train loss: 0.6873521819114685 |Validation err:
0.4145, Validation loss: 0.6843220796436071
Epoch 7: Train err: 0.43275, Train loss: 0.6860384411811828 |Validation err:
0.4095, Validation loss: 0.6826358549296856
Epoch 8: Train err: 0.431875, Train loss: 0.6845943398475647 |Validation err:
0.4125, Validation loss: 0.6808857917785645
Epoch 9: Train err: 0.421875, Train loss: 0.6830693736076355 |Validation err:
0.402, Validation loss: 0.6782517209649086
Epoch 10: Train err: 0.416875, Train loss: 0.6814315037727356 |Validation er
r: 0.4145, Validation loss: 0.6754029337316751
Epoch 11: Train err: 0.42575, Train loss: 0.6797368054389954 |Validation err:
0.4075, Validation loss: 0.6749423574656248
Epoch 12: Train err: 0.4125, Train loss: 0.6780308566093445 |Validation err:
0.4015, Validation loss: 0.672095162793994
Epoch 13: Train err: 0.412625, Train loss: 0.675962854385376 |Validation err:
0.394, Validation loss: 0.6721436306834221
Epoch 14: Train err: 0.40175, Train loss: 0.6739200620651246 |Validation err:
0.3935, Validation loss: 0.6697118692100048
Epoch 15: Train err: 0.402375, Train loss: 0.6716854658126831 |Validation er
r: 0.3865, Validation loss: 0.665617598220706
Epoch 16: Train err: 0.390875, Train loss: 0.6689650087356568 |Validation er
r: 0.384, Validation loss: 0.6643717419356108
Epoch 17: Train err: 0.388375, Train loss: 0.6662356343269348 |Validation er
r: 0.3915, Validation loss: 0.659590408205986
Epoch 18: Train err: 0.380125, Train loss: 0.6630433263778687 |Validation er
r: 0.3855, Validation loss: 0.6553491819649935
Epoch 19: Train err: 0.380875, Train loss: 0.6598082847595215 |Validation er
r: 0.3775, Validation loss: 0.6537622325122356
Epoch 20: Train err: 0.37175, Train loss: 0.6560162215232849 |Validation err:
0.379, Validation loss: 0.6501419115811586
Epoch 21: Train err: 0.3715, Train loss: 0.6528710188865662 |Validation err:
0.368, Validation loss: 0.6461002621799707
Epoch 22: Train err: 0.36525, Train loss: 0.6494272594451904 |Validation err:
0.3625, Validation loss: 0.6456375457346439
Epoch 23: Train err: 0.361875, Train loss: 0.645886224269867 |Validation err:
0.3615, Validation loss: 0.6407733038067818
Epoch 24: Train err: 0.357875, Train loss: 0.6445935258865356 |Validation er
r: 0.3565, Validation loss: 0.6387317068874836
Epoch 25: Train err: 0.3565, Train loss: 0.6392346773147582 |Validation err:
0.3515, Validation loss: 0.6360961999744177
Epoch 26: Train err: 0.35175, Train loss: 0.6374417686462402 |Validation err:
0.357, Validation loss: 0.6371701117604971
Epoch 27: Train err: 0.353, Train loss: 0.6342106537818909 |Validation err:
0.3495, Validation loss: 0.6360247172415257
Epoch 28: Train err: 0.352, Train loss: 0.6323257007598877 |Validation err:
```

```
0.3525, Validation loss: 0.633002320304513
Epoch 29: Train err: 0.35575, Train loss: 0.6308963932991027 |Validation err:
0.349, Validation loss: 0.6314164530485868
Epoch 30: Train err: 0.345, Train loss: 0.6284754285812378 |Validation err:
0.3535, Validation loss: 0.633969908580184
Finished Training
Total time elapsed: 151.05 seconds
```



Changing the learning rate did not impact the time it took to train, the total time elapsed was still about 151 seconds. However, a smaller learning rate seems to have greatly decreased the issue of overfitting as there is pretty much no gap between the training and validation accuracies. However, this is likely due to the model taking steps that are too small, resulting in signficant underfitting, which is suggested by the high loss/error compared to previous settings.

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)

large_model_path = get_model_name("large", batch_size=64, learning_rate=0.1, e
poch=29)
plot_training_curve(large_model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.467625, Train loss: 0.6861868515014649 |Validation err:
0.4215, Validation loss: 0.6740307230502367
Epoch 2: Train err: 0.39, Train loss: 0.6569749841690063 |Validation err: 0.3
91, Validation loss: 0.6574704591184855
Epoch 3: Train err: 0.355125, Train loss: 0.6253251066207886 |Validation err:
0.3505, Validation loss: 0.620346087962389
Epoch 4: Train err: 0.329125, Train loss: 0.606244039773941 |Validation err:
0.3575, Validation loss: 0.6671020425856113
Epoch 5: Train err: 0.326, Train loss: 0.5957692120075225 |Validation err: 0.
334, Validation loss: 0.5925362259149551
Epoch 6: Train err: 0.303375, Train loss: 0.575137921333313 |Validation err:
0.3205, Validation loss: 0.6100335158407688
Epoch 7: Train err: 0.304, Train loss: 0.5726823313236237 |Validation err: 0.
317, Validation loss: 0.6183756990358233
Epoch 8: Train err: 0.288625, Train loss: 0.5487984850406646 |Validation err:
0.323, Validation loss: 0.5915514379739761
Epoch 9: Train err: 0.284375, Train loss: 0.540764660358429 |Validation err:
0.332, Validation loss: 0.6098141325637698
Epoch 10: Train err: 0.28525, Train loss: 0.543907998085022 |Validation err:
0.344, Validation loss: 0.6419565323740244
Epoch 11: Train err: 0.276, Train loss: 0.5359194176197052 |Validation err:
0.3245, Validation loss: 0.6399335693567991
Epoch 12: Train err: 0.266875, Train loss: 0.527840470790863 |Validation err:
0.3135, Validation loss: 0.6438787821680307
Epoch 13: Train err: 0.263625, Train loss: 0.5145214052200318 |Validation er
r: 0.316, Validation loss: 0.6905622445046902
Epoch 14: Train err: 0.257, Train loss: 0.5103720374107361 |Validation err:
0.3115, Validation loss: 0.670242452993989
Epoch 15: Train err: 0.24025, Train loss: 0.4889113302230835 |Validation err:
0.3165, Validation loss: 0.6930663185194135
Epoch 16: Train err: 0.250375, Train loss: 0.49782308340072634 |Validation er
r: 0.337, Validation loss: 0.6436392497271299
Epoch 17: Train err: 0.245125, Train loss: 0.4957301735877991 |Validation er
r: 0.3245, Validation loss: 0.6691232575103641
Epoch 18: Train err: 0.237375, Train loss: 0.4781391134262085 |Validation er
r: 0.3275, Validation loss: 0.683715783059597
Epoch 19: Train err: 0.2505, Train loss: 0.5043014485836029 |Validation err:
0.349, Validation loss: 0.7344501707702875
Epoch 20: Train err: 0.233625, Train loss: 0.47596864688396456 |Validation er
r: 0.3285, Validation loss: 0.7142069041728973
Epoch 21: Train err: 0.241375, Train loss: 0.48449467778205874 |Validation er
r: 0.342, Validation loss: 0.7656483091413975
Epoch 22: Train err: 0.24725, Train loss: 0.4931287405490875 |Validation err:
0.332, Validation loss: 0.8585414439439774
Epoch 23: Train err: 0.241625, Train loss: 0.48796604776382446 |Validation er
r: 0.3305, Validation loss: 0.818379333242774
Epoch 24: Train err: 0.24925, Train loss: 0.4914700272083282 |Validation err:
0.3425, Validation loss: 0.8604169189929962
Epoch 25: Train err: 0.266125, Train loss: 0.5153710978031159 |Validation er
r: 0.331, Validation loss: 0.7632325710728765
Epoch 26: Train err: 0.25825, Train loss: 0.5016732423305511 |Validation err:
0.3445, Validation loss: 0.8430985994637012
Epoch 27: Train err: 0.236875, Train loss: 0.4666298909187317 |Validation er
r: 0.346, Validation loss: 0.803265543654561
Epoch 28: Train err: 0.237375, Train loss: 0.4621445622444153 |Validation er
```

```
r: 0.345, Validation loss: 0.8684306722134352
Epoch 29: Train err: 0.2405, Train loss: 0.4829731698036194 |Validation err:
0.3515, Validation loss: 0.906199125573039
Epoch 30: Train err: 0.2805, Train loss: 0.5351826040744782 |Validation err:
0.389, Validation loss: 0.7011800371110439
Finished Training
Total time elapsed: 164.13 seconds
```

Increasing the learning rate still results in approximately the same training time.

Increasing the learning rate exacerbates the overfitting seen with the default settings. The average gap between the validation error and training error widens as the model is able to take larger steps to more quickly "memorize" the training examples.

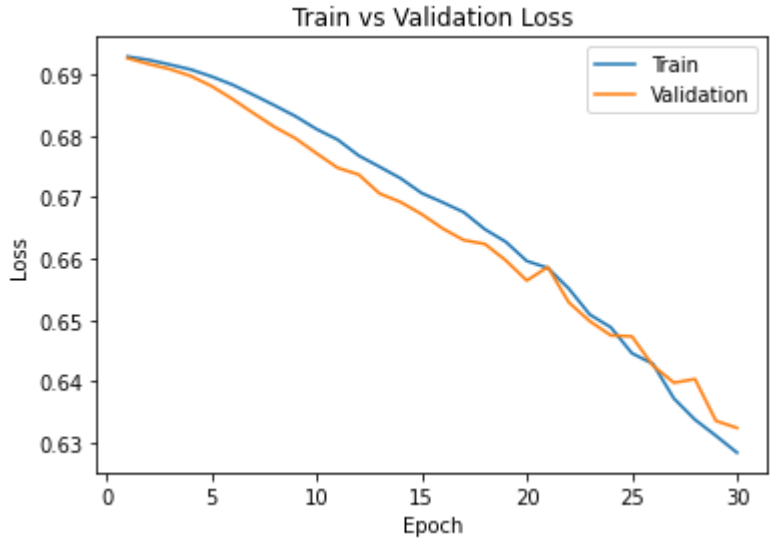
Part (c) - 3pt

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, learning_rate=0.01)

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

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.48175, Train loss: 0.6929379552602768 |Validation err:
0.478, Validation loss: 0.6926824003458023
Epoch 2: Train err: 0.457625, Train loss: 0.6924104019999504 |Validation err:
0.434, Validation loss: 0.6917425245046616
Epoch 3: Train err: 0.437, Train loss: 0.6916500590741634 |Validation err: 0.
4265, Validation loss: 0.6909129917621613
Epoch 4: Train err: 0.433625, Train loss: 0.6908449940383434 |Validation err:
0.424, Validation loss: 0.6897870451211929
Epoch 5: Train err: 0.434, Train loss: 0.6896935552358627 |Validation err: 0.
424, Validation loss: 0.6881355047225952
Epoch 6: Train err: 0.438, Train loss: 0.688353206962347 |Validation err: 0.4
285, Validation loss: 0.686011865735054
Epoch 7: Train err: 0.439375, Train loss: 0.6866871677339077 |Validation err:
0.426, Validation loss: 0.6836968809366226
Epoch 8: Train err: 0.43525, Train loss: 0.6849770769476891 |Validation err:
0.4115, Validation loss: 0.6814671903848648
Epoch 9: Train err: 0.42375, Train loss: 0.6832009293138981 |Validation err:
0.414, Validation loss: 0.679591491818428
Epoch 10: Train err: 0.421, Train loss: 0.6811089366674423 |Validation err:
0.416, Validation loss: 0.6771548539400101
Epoch 11: Train err: 0.420875, Train loss: 0.6794026419520378 |Validation er
r: 0.4095, Validation loss: 0.6748111099004745
Epoch 12: Train err: 0.41475, Train loss: 0.6768048219382763 |Validation err:
0.412, Validation loss: 0.6737060546875
Epoch 13: Train err: 0.4105, Train loss: 0.6749702803790569 |Validation err:
0.412, Validation loss: 0.6706101596355438
Epoch 14: Train err: 0.407125, Train loss: 0.6730880849063396 |Validation er
r: 0.4125, Validation loss: 0.6692148000001907
Epoch 15: Train err: 0.4005, Train loss: 0.6706806942820549 |Validation err:
0.4105, Validation loss: 0.667252704501152
Epoch 16: Train err: 0.397625, Train loss: 0.6691771410405636 |Validation er
r: 0.405, Validation loss: 0.6649097055196762
Epoch 17: Train err: 0.393875, Train loss: 0.6675694733858109 |Validation er
r: 0.401, Validation loss: 0.6630224883556366
Epoch 18: Train err: 0.393, Train loss: 0.6648042872548103 |Validation err:
0.3945, Validation loss: 0.6624014377593994
Epoch 19: Train err: 0.38625, Train loss: 0.662746611982584 |Validation err:
0.388, Validation loss: 0.6597220152616501
Epoch 20: Train err: 0.38175, Train loss: 0.6596181839704514 |Validation err:
0.4005, Validation loss: 0.6564337313175201
Epoch 21: Train err: 0.38575, Train loss: 0.6584899798035622 |Validation err:
0.3885, Validation loss: 0.6586423963308334
Epoch 22: Train err: 0.378125, Train loss: 0.655123382806778 |Validation err:
0.3855, Validation loss: 0.6528600305318832
Epoch 23: Train err: 0.372125, Train loss: 0.6508794128894806 |Validation er
r: 0.3835, Validation loss: 0.6497963815927505
Epoch 24: Train err: 0.37675, Train loss: 0.6488028429448605 |Validation err:
0.385, Validation loss: 0.6474899500608444
Epoch 25: Train err: 0.368625, Train loss: 0.6445869170129299 |Validation er
r: 0.382, Validation loss: 0.6473268568515778
Epoch 26: Train err: 0.372625, Train loss: 0.6428566053509712 |Validation er
r: 0.3745, Validation loss: 0.6425703465938568
Epoch 27: Train err: 0.359375, Train loss: 0.6372117549180984 |Validation er
r: 0.379, Validation loss: 0.6397799849510193
Epoch 28: Train err: 0.35425, Train loss: 0.6337667480111122 |Validation err:
```

```
0.3695, Validation loss: 0.6403783112764359
Epoch 29: Train err: 0.3535, Train loss: 0.6311353109776974 |Validation err:
0.366, Validation loss: 0.6335585117340088
Epoch 30: Train err: 0.353, Train loss: 0.6283832415938377 |Validation err:
0.3675, Validation loss: 0.6324127316474915
Finished Training
Total time elapsed: 116.21 seconds
```



Increasing the batch size of the model to 512 greatly reduces training time (by about 30 seconds). This is because the model is averaging the loss from many more examples and then conducting back-propagation, greatly reducing the number of computations required for gradient calculations and updating weights.

Increasing the batch size seems to also have the same effect as decreasing the learning rate to 0.001. The model seems to have underfit the data, exhibiting a high error/loss and a small gap between the training and validation error/loss.

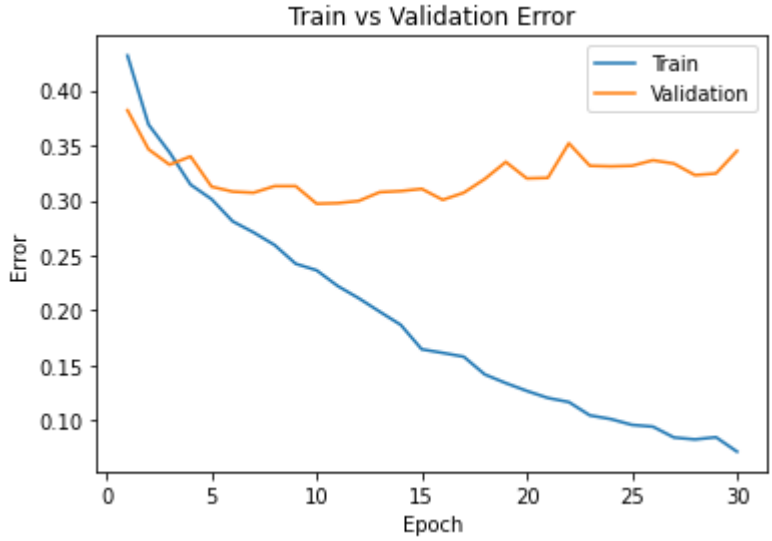
Part (d) - 3pt

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, learning_rate=0.01)

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

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.43175, Train loss: 0.6774994022846222 |Validation err:
0.382, Validation loss: 0.6513170118331909
Epoch 2: Train err: 0.369, Train loss: 0.639639899969101 |Validation err: 0.3
465, Validation loss: 0.6161113576889038
Epoch 3: Train err: 0.34375, Train loss: 0.6098222947120666 |Validation err:
0.3325, Validation loss: 0.6260210764408112
Epoch 4: Train err: 0.314375, Train loss: 0.5849691489338875 |Validation err:
0.34, Validation loss: 0.6044013917446136
Epoch 5: Train err: 0.301125, Train loss: 0.5689119303822517 |Validation err:
0.3125, Validation loss: 0.576918310880661
Epoch 6: Train err: 0.281, Train loss: 0.5452213581204415 |Validation err: 0.
308, Validation loss: 0.5708447456359863
Epoch 7: Train err: 0.270875, Train loss: 0.5272981298565864 |Validation err:
0.307, Validation loss: 0.5854293291568756
Epoch 8: Train err: 0.259375, Train loss: 0.5070905526578426 |Validation err:
0.313, Validation loss: 0.5877130818367005
Epoch 9: Train err: 0.242375, Train loss: 0.4968344421982765 |Validation err:
0.313, Validation loss: 0.5922425072193146
Epoch 10: Train err: 0.236375, Train loss: 0.4756101597249508 |Validation er
r: 0.297, Validation loss: 0.5718690166473389
Epoch 11: Train err: 0.222125, Train loss: 0.4599769461452961 |Validation er
r: 0.2975, Validation loss: 0.6376970833539963
Epoch 12: Train err: 0.211, Train loss: 0.4454492371380329 |Validation err:
0.2995, Validation loss: 0.609202565908432
Epoch 13: Train err: 0.19875, Train loss: 0.4245421719551086 |Validation err:
0.3075, Validation loss: 0.6494987765550614
Epoch 14: Train err: 0.18675, Train loss: 0.4007472907453775 |Validation err:
0.3085, Validation loss: 0.6610016552209854
Epoch 15: Train err: 0.1645, Train loss: 0.3759974058121443 |Validation err:
0.3105, Validation loss: 0.7106090537309646
Epoch 16: Train err: 0.16125, Train loss: 0.3591455406397581 |Validation err:
0.3005, Validation loss: 0.7310364942550659
Epoch 17: Train err: 0.15775, Train loss: 0.3463234790861607 |Validation err:
0.307, Validation loss: 0.7263009325265884
Epoch 18: Train err: 0.141625, Train loss: 0.32175366275012496 |Validation er
r: 0.3195, Validation loss: 0.7913952842950821
Epoch 19: Train err: 0.13375, Train loss: 0.30618105667084455 |Validation er
r: 0.335, Validation loss: 0.8032052783966065
Epoch 20: Train err: 0.126625, Train loss: 0.3029071792438626 |Validation er
r: 0.32, Validation loss: 0.8106685240268707
Epoch 21: Train err: 0.12025, Train loss: 0.28682796490937473 |Validation er
r: 0.3205, Validation loss: 0.8259474284648896
Epoch 22: Train err: 0.1165, Train loss: 0.27489088076353074 |Validation err:
0.352, Validation loss: 0.8937610774040222
Epoch 23: Train err: 0.104375, Train loss: 0.2467898527495563 |Validation er
r: 0.3315, Validation loss: 1.0021928198337555
Epoch 24: Train err: 0.101, Train loss: 0.23970085787773132 |Validation err:
0.331, Validation loss: 1.1290796399116516
Epoch 25: Train err: 0.09575, Train loss: 0.23643119425699116 |Validation er
r: 0.3315, Validation loss: 1.1338514368534087
Epoch 26: Train err: 0.094125, Train loss: 0.2325953512713313 |Validation er
r: 0.3365, Validation loss: 1.1414263204336166
Epoch 27: Train err: 0.08425, Train loss: 0.21040759468451142 |Validation er
r: 0.3335, Validation loss: 1.1823678107261657
Epoch 28: Train err: 0.0825, Train loss: 0.20643112615589052 |Validation err:
```

```
0.323, Validation loss: 1.266836181640625
Epoch 29: Train err: 0.0845, Train loss: 0.21273409337876364 |Validation err:
0.3245, Validation loss: 1.406717705130577
Epoch 30: Train err: 0.071375, Train loss: 0.18387044295761734 |Validation er
r: 0.345, Validation loss: 1.4871552000045776
Finished Training
Total time elapsed: 189.93 seconds
```



Decreasing the batch size to 16 greatly increased the training time of the model (by about 40 seconds). This is because reducing the batch size will result in back-propagation computations happening much more frequently, increasing the computational workload and thus the time required for training.

Decreasing the batch size resulted in more overfitting, which can be seen by an even lower training error/loss and larger gap between the training and validation error/loss. This result is surprising since smaller batch sizes introduce noise into the training process, which I would have expected to produce a regularizing effect, reducing generalization error.

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.

I would use a large network with a batch size of 128 and a learning rate of 0.007.
I choose to use a large network because the small network underfit the data, suggesting it may not have the capacity required to represent the patterns in the data accurately.
However, since the large network suffers from overfitting under default settings, new hyperparameters are required to correct this problem. Since previous data showed that a batch size of 512 and learning rate of 0.001 caused the large model to underfit, I chose to use a batch size of 128 and a learning rate of 0.007 (values between the default and underfitting) to try and reduce the overfitting.

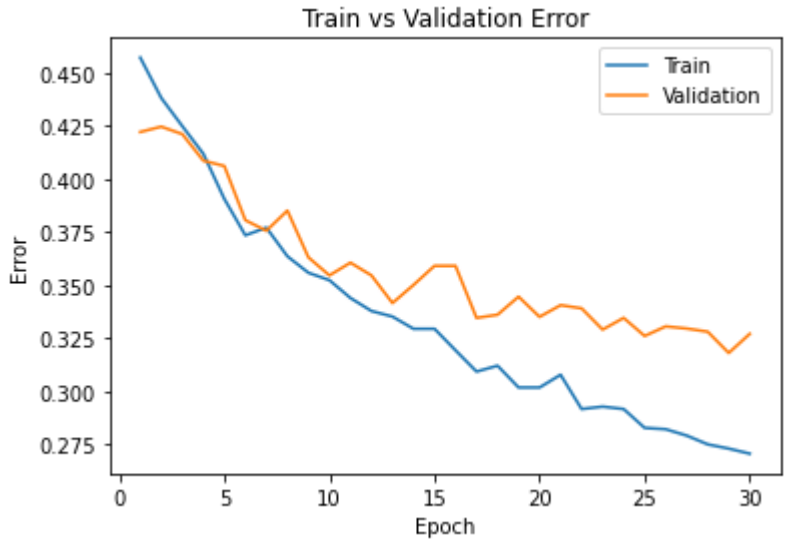
Part (b) - 1pt

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=128, learning_rate=0.007)

large_model_path = get_model_name("large", batch_size=128, learning_rate=0.00
7, epoch=29)
plot_training_curve(large_model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.457, Train loss: 0.6910408754197378 |Validation err: 0.
422, Validation loss: 0.6873135790228844
Epoch 2: Train err: 0.438, Train loss: 0.6860193023605953 |Validation err: 0.
4245, Validation loss: 0.6810816451907158
Epoch 3: Train err: 0.42475, Train loss: 0.6806464431777833 |Validation err:
0.421, Validation loss: 0.6757452934980392
Epoch 4: Train err: 0.41175, Train loss: 0.6738583748302762 |Validation err:
0.4085, Validation loss: 0.6683167852461338
Epoch 5: Train err: 0.3905, Train loss: 0.6648656669117156 |Validation err:
0.406, Validation loss: 0.6591531038284302
Epoch 6: Train err: 0.373375, Train loss: 0.6526404704366412 |Validation err:
0.3805, Validation loss: 0.6483876891434193
Epoch 7: Train err: 0.377, Train loss: 0.6475891300610134 |Validation err: 0.
3755, Validation loss: 0.6423848606646061
Epoch 8: Train err: 0.3635, Train loss: 0.6384122816343156 |Validation err:
0.385, Validation loss: 0.6452219411730766
Epoch 9: Train err: 0.35575, Train loss: 0.6332868895833454 |Validation err:
0.363, Validation loss: 0.637889489531517
Epoch 10: Train err: 0.35225, Train loss: 0.6297400404536535 |Validation err:
0.3545, Validation loss: 0.6304467022418976
Epoch 11: Train err: 0.343875, Train loss: 0.6220508662481157 |Validation er
r: 0.3605, Validation loss: 0.6359161175787449
Epoch 12: Train err: 0.33775, Train loss: 0.6138965875383408 |Validation err:
0.3545, Validation loss: 0.6286138482391834
Epoch 13: Train err: 0.335125, Train loss: 0.6121698797695221 |Validation er
r: 0.3415, Validation loss: 0.625329814851284
Epoch 14: Train err: 0.329375, Train loss: 0.602298945661575 |Validation err:
0.35, Validation loss: 0.6320119127631187
Epoch 15: Train err: 0.329375, Train loss: 0.6004971322559175 |Validation er
r: 0.359, Validation loss: 0.6384634263813496
Epoch 16: Train err: 0.31925, Train loss: 0.5939466007172115 |Validation err:
0.359, Validation loss: 0.6275628842413425
Epoch 17: Train err: 0.30925, Train loss: 0.5886616460860722 |Validation err:
0.3345, Validation loss: 0.6121050827205181
Epoch 18: Train err: 0.312, Train loss: 0.5845226380560133 |Validation err:
0.336, Validation loss: 0.6096345745027065
Epoch 19: Train err: 0.30175, Train loss: 0.5770058073694744 |Validation err:
0.3445, Validation loss: 0.6229317262768745
Epoch 20: Train err: 0.30175, Train loss: 0.5747444421525986 |Validation err:
0.335, Validation loss: 0.6092618443071842
Epoch 21: Train err: 0.30775, Train loss: 0.5796105918430147 |Validation err:
0.3405, Validation loss: 0.6118515804409981
Epoch 22: Train err: 0.291625, Train loss: 0.5661955319699787 |Validation er
r: 0.339, Validation loss: 0.6102893352508545
Epoch 23: Train err: 0.29275, Train loss: 0.5658734459725637 |Validation err:
0.329, Validation loss: 0.6025920435786247
Epoch 24: Train err: 0.291625, Train loss: 0.5612757480333722 |Validation er
r: 0.3345, Validation loss: 0.6085603758692741
Epoch 25: Train err: 0.28275, Train loss: 0.5539779956378634 |Validation err:
0.326, Validation loss: 0.6042552664875984
Epoch 26: Train err: 0.282125, Train loss: 0.5523572502628206 |Validation er
r: 0.3305, Validation loss: 0.5947951711714268
Epoch 27: Train err: 0.279125, Train loss: 0.5463265208970933 |Validation er
r: 0.3295, Validation loss: 0.6040886007249355
Epoch 28: Train err: 0.275, Train loss: 0.5414972518171582 |Validation err:
```

```
0.328, Validation loss: 0.6012972388416529
Epoch 29: Train err: 0.273, Train loss: 0.53730094811273 |Validation err: 0.3
18, Validation loss: 0.5929554514586926
Epoch 30: Train err: 0.270625, Train loss: 0.5323283502033779 |Validation er
r: 0.327, Validation loss: 0.5978298410773277
Finished Training
Total time elapsed: 136.15 seconds
```



Part (c) - 2pt

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

Based on the previous result, the model still appears to be overfitting at the end. However, the model also appears to be underfitting since the training error/loss is still very high. Trying a higher batch size (256) to further reduce overfitting but increasing learning rate (0.015) to reduce underfitting may help.

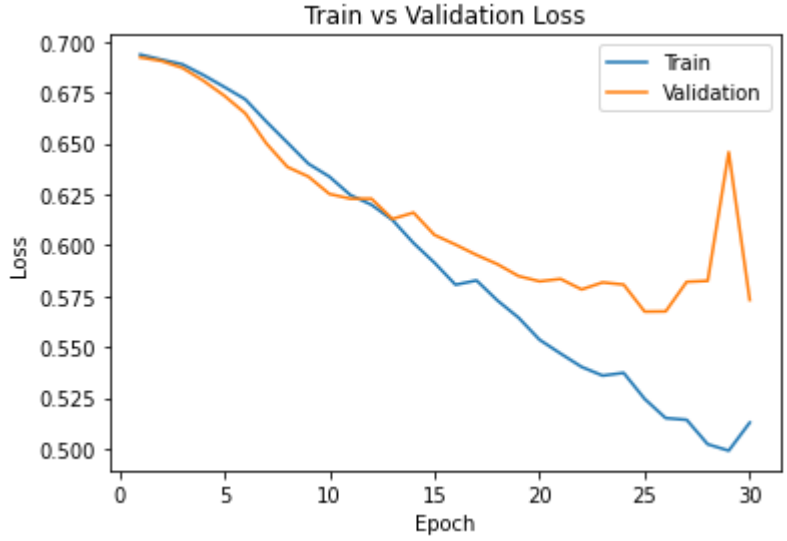
Part (d) - 1pt

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=256, learning_rate=0.015)

large_model_path = get_model_name("large", batch_size=256, learning_rate=0.01
5, epoch=29)
plot_training_curve(large_model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.494875, Train loss: 0.6936976965516806 |Validation err:
0.463, Validation loss: 0.6923194155097008
Epoch 2: Train err: 0.44225, Train loss: 0.6912475973367691 |Validation err:
0.428, Validation loss: 0.6905912458896637
Epoch 3: Train err: 0.44875, Train loss: 0.6889428850263357 |Validation err:
0.426, Validation loss: 0.6871970891952515
Epoch 4: Train err: 0.43475, Train loss: 0.6836735717952251 |Validation err:
0.4305, Validation loss: 0.6809073016047478
Epoch 5: Train err: 0.426125, Train loss: 0.677789706736803 |Validation err:
0.4085, Validation loss: 0.6734909266233444
Epoch 6: Train err: 0.416625, Train loss: 0.671720227226615 |Validation err:
0.392, Validation loss: 0.6648043766617775
Epoch 7: Train err: 0.392125, Train loss: 0.6607413776218891 |Validation err:
0.377, Validation loss: 0.6500054448843002
Epoch 8: Train err: 0.386625, Train loss: 0.6504422407597303 |Validation err:
0.3575, Validation loss: 0.6384843066334724
Epoch 9: Train err: 0.366625, Train loss: 0.6400607321411371 |Validation err:
0.3635, Validation loss: 0.633726455271244
Epoch 10: Train err: 0.355375, Train loss: 0.633692255243659 |Validation err:
0.353, Validation loss: 0.6251889541745186
Epoch 11: Train err: 0.344125, Train loss: 0.6246365290135145 |Validation er
r: 0.354, Validation loss: 0.6228611022233963
Epoch 12: Train err: 0.343875, Train loss: 0.619882682338357 |Validation err:
0.342, Validation loss: 0.6228565722703934
Epoch 13: Train err: 0.33675, Train loss: 0.6123799439519644 |Validation err:
0.3495, Validation loss: 0.6129128560423851
Epoch 14: Train err: 0.326875, Train loss: 0.6010868307203054 |Validation er
r: 0.3425, Validation loss: 0.6159958615899086
Epoch 15: Train err: 0.317125, Train loss: 0.5913702081888914 |Validation er
r: 0.336, Validation loss: 0.6050457581877708
Epoch 16: Train err: 0.311125, Train loss: 0.5805520266294479 |Validation er
r: 0.3335, Validation loss: 0.6002898812294006
Epoch 17: Train err: 0.30775, Train loss: 0.5826773606240749 |Validation err:
0.32, Validation loss: 0.5951925665140152
Epoch 18: Train err: 0.29975, Train loss: 0.572782464325428 |Validation err:
0.318, Validation loss: 0.5906414538621902
Epoch 19: Train err: 0.296, Train loss: 0.5643681827932596 |Validation err:
0.317, Validation loss: 0.5848095864057541
Epoch 20: Train err: 0.288625, Train loss: 0.5535138603299856 |Validation er
r: 0.3095, Validation loss: 0.5822367370128632
Epoch 21: Train err: 0.28125, Train loss: 0.546731417067349 |Validation err:
0.307, Validation loss: 0.5834114998579025
Epoch 22: Train err: 0.27975, Train loss: 0.540170069783926 |Validation err:
0.3045, Validation loss: 0.5782489627599716
Epoch 23: Train err: 0.270375, Train loss: 0.5358881196007133 |Validation er
r: 0.2975, Validation loss: 0.5817062333226204
Epoch 24: Train err: 0.273, Train loss: 0.5372343137860298 |Validation err:
0.304, Validation loss: 0.5806390792131424
Epoch 25: Train err: 0.2655, Train loss: 0.5244381939992309 |Validation err:
0.2955, Validation loss: 0.567351795732975
Epoch 26: Train err: 0.258625, Train loss: 0.5150039913132787 |Validation er
r: 0.2955, Validation loss: 0.5674484372138977
Epoch 27: Train err: 0.25425, Train loss: 0.5141575522720814 |Validation err:
0.3035, Validation loss: 0.5820109099149704
Epoch 28: Train err: 0.251125, Train loss: 0.5021483516320586 |Validation er
```

```
r: 0.294, Validation loss: 0.5824738666415215
Epoch 29: Train err: 0.250625, Train loss: 0.4990005465224385 |Validation er
r: 0.3175, Validation loss: 0.6457634940743446
Epoch 30: Train err: 0.262875, Train loss: 0.5127848600968719 |Validation er
r: 0.296, Validation loss: 0.5730881094932556
Finished Training
Total time elapsed: 138.00 seconds
```



Part 4. Evaluating the Best Model [15 pt]

Part (a) - 1pt

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

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

In [ ]:
```
net = LargeNet()
model_path = get_model_name(net.name, batch_size=256, learning_rate=0.015, epo
ch=24)
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).

Both of the networks from the previous section produced similar results. However, the first model shows a consistent downward trend in the error/loss towards the end whereas the second model seems to be overfitting towards the end. I believe that if both models were trained for a few more epochs, the first model may experience a further decrease in error/loss while the second model will overfit more. Thus, I choose the first model.

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=64)

loss = nn.BCEWithLogitsLoss()

test_error, test_loss = evaluate(net, test_loader, loss)
print(f"The test error is: {test_error}")
print(f"The test loss is: {test_loss}")

#  For part D
val_error, val_loss = evaluate(net, val_loader, loss)
print(f"The validation error is: {val_error}")
print(f"The validation loss is: {val_loss}")
```

```
Files already downloaded and verified
Files already downloaded and verified
The test error is: 0.2875
The test loss is: 0.5411103935912251
The validation error is: 0.2955
The validation loss is: 0.5681194672361016
```

Part (d) - 3pt

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

The validation error is slightly higher than the test error. This is unexpected since the hyperparameters of the model were tuned based on the validation dataset. Tuning the hyperparameters of the model on the validation dataset biases the model towards those examples since we optimize the model to improve validation error, so we would expect the test dataset, which the model has truly never seen before, to have a higher error than the validation dataset.

Part (e) - 2pt

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

We only use the test data set at the very end because we don't want to bias our decisions towards optimizing the test set. The test set is supposed to serve as a final test of how the model performs. If we test our model repeatedly on the test set, we risk biasing our decisions towards increasing the accuracy of the model specifically on the testing examples, essentially overfitting to those examples, resulting in an overly optimistic estimation of our model's capabilities.

Part (f) - 5pt

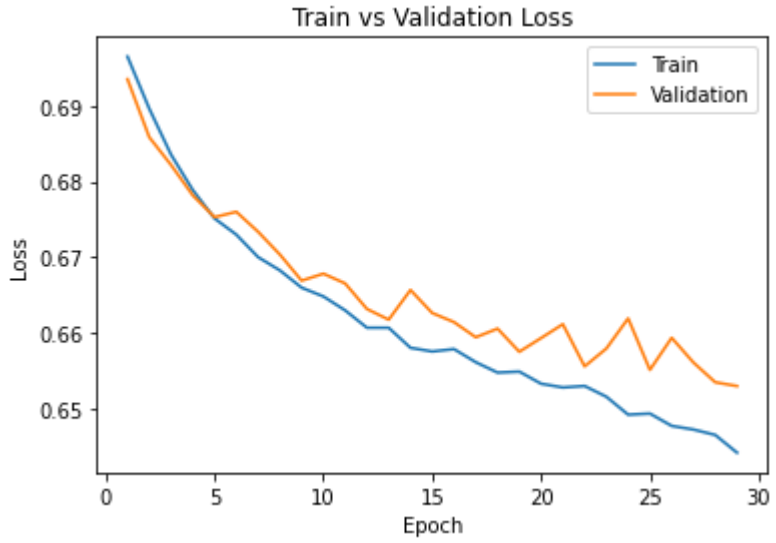
How does the your best CNN model compare with an 2-layer ANN model (no convolutional layers) on classifying cat and dog images. You can use a 2-layer ANN architecture similar to what you used in Lab 1. You should explore different hyperparameter settings to determine how well you can do on the validation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

# Define the ANN
class Pigeon(nn.Module):
    def __init__(self):
        super(Pigeon, self).__init__()
        self.name = "Pigeon"
        self.layer1 = nn.Linear(32*32*3, 64)
        self.layer2 = nn.Linear(64, 1)
    def forward(self, img):
        flattened = img.view(-1, 32*32*3)
        x1 = self.layer1(flattened)
        activation1 = F.relu(x1)
        x2 = self.layer2(activation1)
        x2 = x2.squeeze(1)
        return x2

# Hyperparameters
bs = 600
lr = 0.001
epochs = 29

# Train the ANN
Pigeon = Pigeon()
train_net(Pigeon, batch_size=bs, learning_rate=lr, num_epochs=epochs)

# Plot error
model_path = get_model_name("Pigeon", batch_size=bs, learning_rate=lr, epoch=e
pochs-1)
plot_training_curve(model_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.50975, Train loss: 0.6964778985295977 |Validation err:
0.493, Validation loss: 0.6934930384159088
Epoch 2: Train err: 0.46825, Train loss: 0.6896014894757952 |Validation err:
0.4585, Validation loss: 0.6858328878879547
Epoch 3: Train err: 0.430625, Train loss: 0.6835294025284904 |Validation err:
0.4355, Validation loss: 0.6821668893098831
Epoch 4: Train err: 0.417125, Train loss: 0.6788464742047446 |Validation err:
0.4295, Validation loss: 0.6781504601240158
Epoch 5: Train err: 0.4115, Train loss: 0.6750978188855308 |Validation err:
0.4265, Validation loss: 0.675269290804863
Epoch 6: Train err: 0.406125, Train loss: 0.6729624611990792 |Validation err:
0.422, Validation loss: 0.6759781241416931
Epoch 7: Train err: 0.40175, Train loss: 0.6699913697583335 |Validation err:
0.418, Validation loss: 0.6733369082212448
Epoch 8: Train err: 0.401375, Train loss: 0.6682395764759609 |Validation err:
0.414, Validation loss: 0.6703372150659561
Epoch 9: Train err: 0.3995, Train loss: 0.6659550794533321 |Validation err:
0.412, Validation loss: 0.6668849289417267
Epoch 10: Train err: 0.398375, Train loss: 0.6647960288184029 |Validation er
r: 0.412, Validation loss: 0.6678029298782349
Epoch 11: Train err: 0.39625, Train loss: 0.662956714630127 |Validation err:
0.4065, Validation loss: 0.6665138751268387
Epoch 12: Train err: 0.395125, Train loss: 0.660664188010352 |Validation err:
0.408, Validation loss: 0.6631394922733307
Epoch 13: Train err: 0.39325, Train loss: 0.6606614504541669 |Validation err:
0.4065, Validation loss: 0.6617327779531479
Epoch 14: Train err: 0.390875, Train loss: 0.6580118536949158 |Validation er
r: 0.403, Validation loss: 0.6656453758478165
Epoch 15: Train err: 0.38975, Train loss: 0.6575294647897992 |Validation err:
0.4035, Validation loss: 0.6626085340976715
Epoch 16: Train err: 0.388875, Train loss: 0.6578425765037537 |Validation er
r: 0.4025, Validation loss: 0.6614037901163101
Epoch 17: Train err: 0.38925, Train loss: 0.6561119343553271 |Validation err:
0.4025, Validation loss: 0.659403070807457
Epoch 18: Train err: 0.3895, Train loss: 0.6547381239277976 |Validation err:
0.4, Validation loss: 0.6605418920516968
Epoch 19: Train err: 0.388625, Train loss: 0.6548675341265542 |Validation er
r: 0.399, Validation loss: 0.6574917584657669
Epoch 20: Train err: 0.385125, Train loss: 0.6532684138842991 |Validation er
r: 0.399, Validation loss: 0.6592984050512314
Epoch 21: Train err: 0.38475, Train loss: 0.6527744318757739 |Validation err:
0.396, Validation loss: 0.6611303985118866
Epoch 22: Train err: 0.382375, Train loss: 0.6529684151921954 |Validation er
r: 0.3985, Validation loss: 0.655558243393898
Epoch 23: Train err: 0.381125, Train loss: 0.651549471276147 |Validation err:
0.393, Validation loss: 0.6579194068908691
Epoch 24: Train err: 0.378625, Train loss: 0.6491800078323909 |Validation er
r: 0.392, Validation loss: 0.6618966311216354
Epoch 25: Train err: 0.37825, Train loss: 0.6493269886289325 |Validation err:
0.392, Validation loss: 0.6550945937633514
Epoch 26: Train err: 0.378625, Train loss: 0.6476971464497703 |Validation er
r: 0.39, Validation loss: 0.6593325585126877
Epoch 27: Train err: 0.377125, Train loss: 0.6472180358001164 |Validation er
r: 0.389, Validation loss: 0.6560681909322739
Epoch 28: Train err: 0.37525, Train loss: 0.6465028864996774 |Validation err:
```

```
0.389, Validation loss: 0.6534617394208908
Epoch 29: Train err: 0.373875, Train loss: 0.6441644302436283 |Validation er
r: 0.3895, Validation loss: 0.6529593467712402
Finished Training
Total time elapsed: 89.44 seconds
```



In [ ]:
```
train_loader, val_loader, test_loader, classes = get_data_loader(
    target_classes=["cat", "dog"],
    batch_size=64)

loss = nn.BCEWithLogitsLoss()

test_error, test_loss = evaluate(Pigeon, test_loader, loss)
print(f"The test error is: {test_error}")
print(f"The test loss is: {test_loss}")

#  For part D
val_error, val_loss = evaluate(Pigeon, val_loader, loss)
print(f"The validation error is: {val_error}")
print(f"The validation loss is: {val_loss}")
```

```
Files already downloaded and verified
Files already downloaded and verified
The test error is: 0.3725
The test loss is: 0.649961668998003
The validation error is: 0.3895
The validation loss is: 0.6556138191372156
```

Despite trying a wide variety of parameters, the 2-layer ANN could not perform as well as the CNN. While the best CNN I got performed with a test error of 0.2875, the best ANN I got could only perform with a test error of 0.3725. Thus, the CNN seems to perform much better than the 2-layer ANN in classifying cat and dog images.