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/1YRAEOYBMbpTCBjAqZOjiY0FSGyQuTvfz#scrollTo=SwyDuiuUqIDv

```
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.

```
###############################################################################
# Data Loading

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

    Args:
        dataset: Dataset object
        classes: A list of strings denoting the name of each class
        target_classes: A list of strings denoting the name of desired classes
                        Should be a subset of the 'classes'
    Returns:
        indices: list of indices that have labels corresponding to one of the
                 target classes
```

```
    """
    indices = []
    for i in range(len(dataset)):
        # Check if the label is in the target classes
        label_index = dataset[i][1] # ex: 3
        label_class = classes[label_index] # ex: 'cat'
        if label_class in target_classes:
            indices.append(i)
    return indices

def get_data_loader(target_classes, batch_size):
    """ Loads images of cats and dogs, splits the data into training, validation
    and testing datasets. Returns data loaders for the three preprocessed datasets.

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

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

    classes = ('plane', 'car', 'bird', 'cat',
               'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
    ########################################################################
    # The output of torchvision datasets are PILImage images of range [0, 1].
    # We transform them to Tensors of normalized range [-1, 1].
    transform = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
    # Load CIFAR10 training data
    trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                            download=True, transform=transform)
    # Get the list of indices to sample from
    relevant_indices = get_relevant_indices(trainset, classes, target_classes)

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

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

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

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

```
    return path

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

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

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

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

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

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

Part 1. Visualizing the Data [7 pt]

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

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.

```
# This will download the CIFAR-10 dataset to a folder called "data"
# the first time you run this code.
train_loader, val_loader, test_loader, classes = get_data_loader(
    target_classes=["cat", "dog"],
    batch_size=1) # One image per batch
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:05<00:00, 29130199.79it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Part (a) -- 1 pt

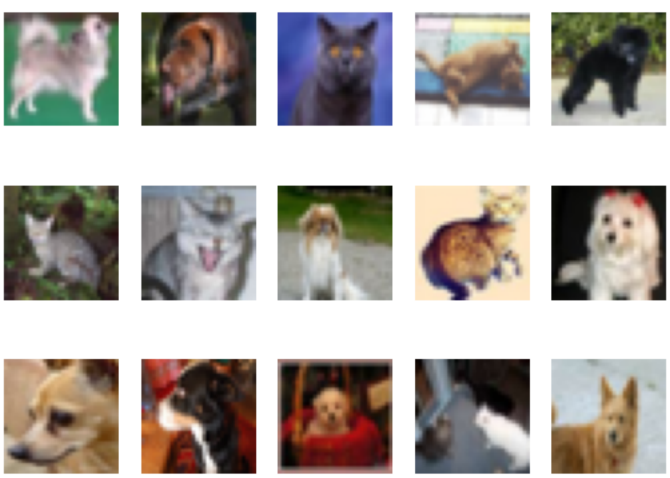
Visualize some of the data by running the code below. Include the visualization in your writeup.

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

```
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?

```
print("There are", len(train_loader), "training,", len(val_loader), "validation, and", len(test_loader), "test examples.")
```

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

Part (c) -- 3pt

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

```
# If we judge the performance based on the loss of the training set, we may run into the issue of overfitting
# the model because the model will be limited to only the training set. This is why we need a validation set
# to train our model because it can verify the performance of the model trained using the training set on
# a new set of data.
```

Part 2. Training [15 pt]

We define two neural networks, a `LargeNet` and `SmallNet`. We'll be training the networks in this section.

You won't understand fully what these networks are doing until the next few classes, and that's okay. For this assignment, please focus on learning how to train networks, and how hyperparameters affect training.

```
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


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


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?)

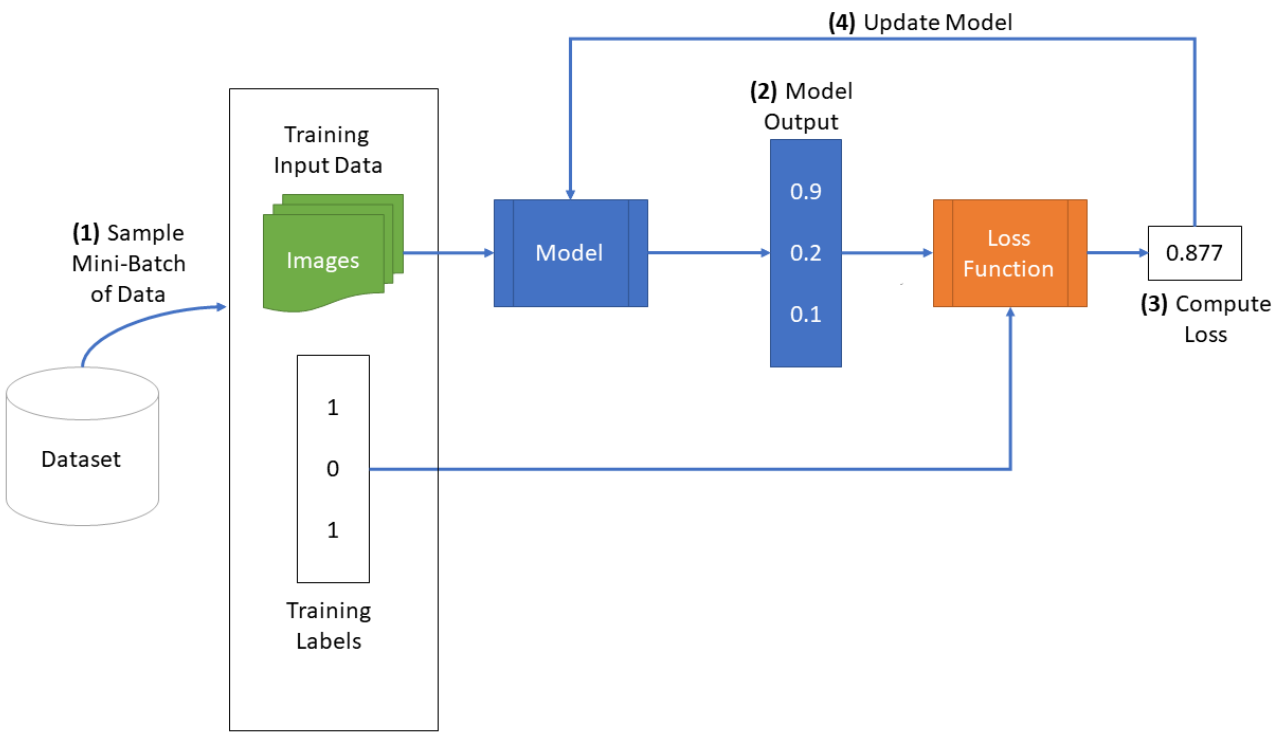
```
size_small = 0
size_large = 0
for param in small_net.parameters():
    size_small += torch.numel(param)
for param in large_net.parameters():
    size_large += torch.numel(param)
print(size_small, "parameters in small_net.")
```

```
print(size_large, "parameters in large_net.")
    #print(param.shape)

     386 parameters in small_net.
     9705 parameters in large_net.
```

The function train_net

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




```
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`?

```
# The default values of the parameters, respectively, are 64, 0.01 and 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.

```
train_net(small_net, 64, 0.01, 5)
# The files written to disk are the CSV files containing the training and validation accuracy and loss, specifically
# model_small_bs64_lr0.01_epoch0
# model_small_bs64_lr0.01_epoch1
# model_small_bs64_lr0.01_epoch2
# model_small_bs64_lr0.01_epoch3
# model_small_bs64_lr0.01_epoch4
# model_small_bs64_lr0.01_epoch4_train_err.csv
# model_small_bs64_lr0.01_epoch4_train_loss.csv
# model_small_bs64_lr0.01_epoch4_val_err.csv
# model_small_bs64_lr0.01_epoch4_val_loss.csv
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.432875, Train loss: 0.6795999398231506 |Validation err: 0.3905, Validation loss: 0.6574821677058935
Epoch 2: Train err: 0.375375, Train loss: 0.649692180633545 |Validation err: 0.387, Validation loss: 0.6631379127502441
Epoch 3: Train err: 0.354125, Train loss: 0.6316869187355042 |Validation err: 0.3365, Validation loss: 0.6204041000455618
Epoch 4: Train err: 0.336625, Train loss: 0.614031611919403 |Validation err: 0.3495, Validation loss: 0.6184336803853512
```

```
Epoch 5: Train err: 0.32725, Train loss: 0.6024144930839539 |Validation err: 0.3185, Validation loss: 0.6101699136197567
Finished Training
Total time elapsed: 24.58 seconds
```

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?

```
# 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
```

```
# training small network with default parameters
train_net(small_net, 64, 0.01, 30)

# training large network with default parameters
train_net(large_net, 64, 0.01, 30)
```

```
# The small network took 148.65 seconds to train, whereas the large network took 164.33 seconds to train.
# The large network took longer to train because it has a drastically large amount of parameters in comparison (as seen in part 2a)
# to the small network.
```

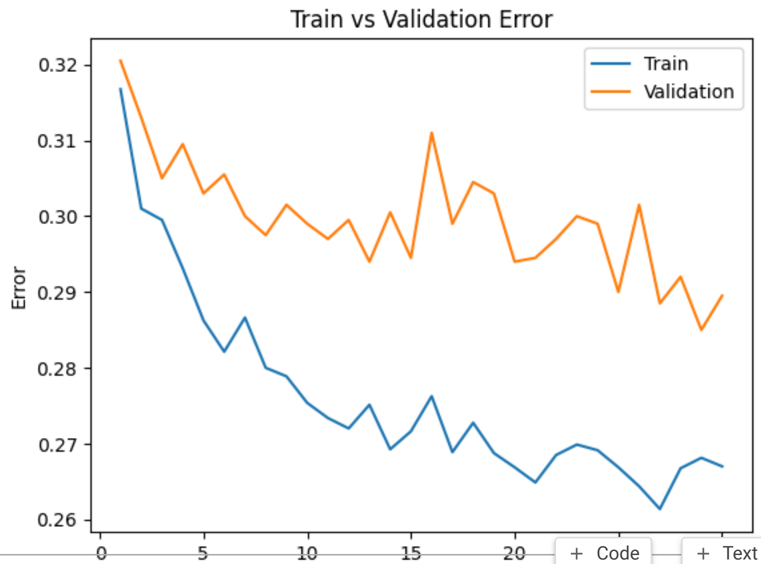
Part (e) - 2pt

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.

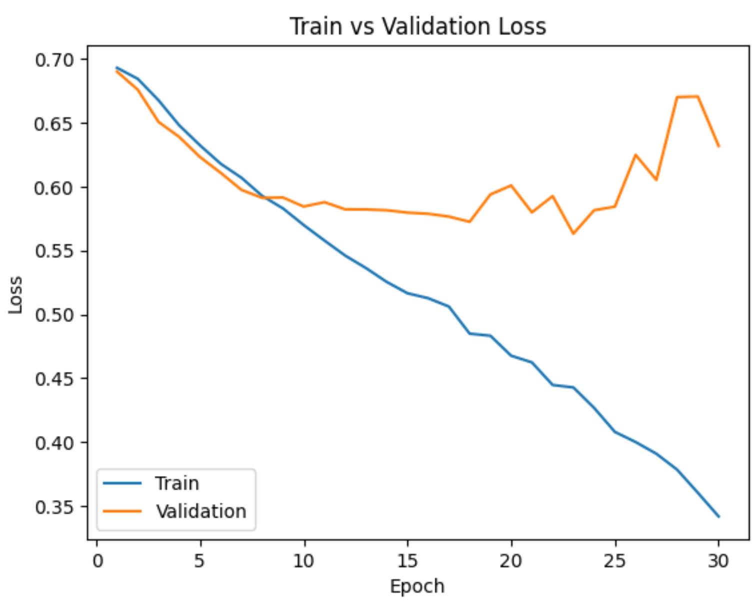
Do this for both the small network and the large network. Include both plots in your writeup.

```
#model_path = get_model_name("small", batch_size=??, learning_rate=??, epoch=29)
#print(get_model_name('small network', 64, 0.01, 5))

small_path = get_model_name('small', 64, 0.01, 29)
plot_training_curve(small_path)
```




```
large_path = get_model_name('large', 64, 0.01, 29)
plot_training_curve(large_path)
```



Part (f) - 5pt

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

```
# For the small network, there are instances of underfitting because the training error and loss of the model
# do not particularly improve with more iterations. In contrast, there are instances of overfitting with the
# large network as the validation loss of this model increases while the training loss decreases (at epochs
# 17-30).
```

Part 3. Optimization Parameters [12 pt]

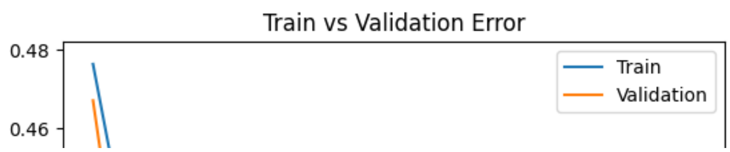
For this section, we will work with `large_net` only.

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.

```
# 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, 64, 0.001, 30)
large_path = get_model_name('large', 64, 0.001, 29)
plot_training_curve(large_path)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.47625, Train loss: 0.6928360013961792 |Validation err: 0.467, Validati
Epoch 2: Train err: 0.448625, Train loss: 0.6922589712142945 |Validation err: 0.4305, Valida
Epoch 3: Train err: 0.43575, Train loss: 0.6916067280769348 |Validation err: 0.4285, Validat
Epoch 4: Train err: 0.43, Train loss: 0.690861343383789 |Validation err: 0.424, Validation l
Epoch 5: Train err: 0.434125, Train loss: 0.6899195008277893 |Validation err: 0.4195, Valida
Epoch 6: Train err: 0.43575, Train loss: 0.6887411961555481 |Validation err: 0.4195, Validat
Epoch 7: Train err: 0.437125, Train loss: 0.6873774147033691 |Validation err: 0.4185, Valida
Epoch 8: Train err: 0.4375, Train loss: 0.6859278454780579 |Validation err: 0.412, Validatio
Epoch 9: Train err: 0.424375, Train loss: 0.6844058036804199 |Validation err: 0.411, Validat
Epoch 10: Train err: 0.424, Train loss: 0.6828502931594849 |Validation err: 0.408, Validatio
Epoch 11: Train err: 0.425375, Train loss: 0.6812348766326904 |Validation err: 0.4125, Valid
Epoch 12: Train err: 0.42, Train loss: 0.6796319708824158 |Validation err: 0.4125, Validatio
Epoch 13: Train err: 0.414875, Train loss: 0.6777918744087219 |Validation err: 0.415, Valida
Epoch 14: Train err: 0.412375, Train loss: 0.6761112003326416 |Validation err: 0.412, Valida
Epoch 15: Train err: 0.40925, Train loss: 0.674472680568695 |Validation err: 0.415, Validati
Epoch 16: Train err: 0.406375, Train loss: 0.6727448840141297 |Validation err: 0.4105, Valid
Epoch 17: Train err: 0.4015, Train loss: 0.6713076601028443 |Validation err: 0.4045, Validat
Epoch 18: Train err: 0.3995, Train loss: 0.6696742882728577 |Validation err: 0.4055, Validat
Epoch 19: Train err: 0.40075, Train loss: 0.6679086356163025 |Validation err: 0.396, Validat
Epoch 20: Train err: 0.392375, Train loss: 0.665787980556488 |Validation err: 0.405, Validat
Epoch 21: Train err: 0.38975, Train loss: 0.6646300601959229 |Validation err: 0.394, Validat
Epoch 22: Train err: 0.388875, Train loss: 0.662373058795929 |Validation err: 0.393, Validat
Epoch 23: Train err: 0.38425, Train loss: 0.6601516346931458 |Validation err: 0.3975, Valida
Epoch 24: Train err: 0.382375, Train loss: 0.6584009389877319 |Validation err: 0.386, Valida
Epoch 25: Train err: 0.37875, Train loss: 0.6554971766471863 |Validation err: 0.388, Validat
Epoch 26: Train err: 0.376625, Train loss: 0.6531173253059387 |Validation err: 0.3875, Valid
Epoch 27: Train err: 0.375, Train loss: 0.6503696331977844 |Validation err: 0.387, Validatio
Epoch 28: Train err: 0.371375, Train loss: 0.6476435809135437 |Validation err: 0.3875, Valid
Epoch 29: Train err: 0.368375, Train loss: 0.6451257643699646 |Validation err: 0.3825, Valid
Epoch 30: Train err: 0.362625, Train loss: 0.6423329524993896 |Validation err: 0.3785, Valid
Finished Training
Total time elapsed: 165.95 seconds
```



By lowering the learning rate to 0.001, the training time takes roughly the same amount of time in comparison to the default settings, taking 166 seconds.

Part (b) - 3pt

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

```
large_net = LargeNet()
train_net(large_net, 64, 0.1, 30)
large_path = get_model_name('large', 64, 0.1, 29)
plot_training_curve(large_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
Epoch 2: Train err: 0.36075, Train loss: 0.6411805458068848 |Validation err: 0.3535, Validat
Epoch 3: Train err: 0.365125, Train loss: 0.6321813461780548 |Validation err: 0.3385, Valida
Epoch 4: Train err: 0.352625, Train loss: 0.6233456182479858 |Validation err: 0.3575, Valida
Epoch 5: Train err: 0.34075, Train loss: 0.6108013873100281 |Validation err: 0.3305, Validat
Epoch 6: Train err: 0.323375, Train loss: 0.5921835997104645 |Validation err: 0.317, Validat
Epoch 7: Train err: 0.3145, Train loss: 0.5817317583560944 |Validation err: 0.3365, Validati
Epoch 8: Train err: 0.29825, Train loss: 0.5660300073623658 |Validation err: 0.3285, Validat
Epoch 9: Train err: 0.290875, Train loss: 0.552809501171112 |Validation err: 0.3315, Validat
Epoch 10: Train err: 0.278625, Train loss: 0.539032607793808 |Validation err: 0.306, Validat
Epoch 11: Train err: 0.272375, Train loss: 0.5236025826931 |Validation err: 0.33, Validation
Epoch 12: Train err: 0.267375, Train loss: 0.5220149435997009 |Validation err: 0.2925, Valid
Epoch 13: Train err: 0.266, Train loss: 0.5160510110855102 |Validation err: 0.3125, Validati
Epoch 14: Train err: 0.24875, Train loss: 0.4951590054035187 |Validation err: 0.3145, Valida
Epoch 15: Train err: 0.264625, Train loss: 0.519231944322586 |Validation err: 0.314, Validat
Epoch 16: Train err: 0.252625, Train loss: 0.5020012385845184 |Validation err: 0.3225, Valid
Epoch 17: Train err: 0.23875, Train loss: 0.481714787364006 |Validation err: 0.357, Validati
Epoch 18: Train err: 0.23375, Train loss: 0.47645506453514097 |Validation err: 0.3375, Valid
Epoch 19: Train err: 0.218125, Train loss: 0.45134368968009947 |Validation err: 0.3445, Vali
Epoch 20: Train err: 0.217875, Train loss: 0.45516350817680357 |Validation err: 0.3245, Vali
Epoch 21: Train err: 0.23275, Train loss: 0.47897080445289614 |Validation err: 0.3255, Valid
Epoch 22: Train err: 0.234875, Train loss: 0.4808810565471649 |Validation err: 0.334, Valida
Epoch 23: Train err: 0.21575, Train loss: 0.4563647754192352 |Validation err: 0.316, Validat
Epoch 24: Train err: 0.2355, Train loss: 0.47718250966072084 |Validation err: 0.327, Validat
Epoch 25: Train err: 0.22025, Train loss: 0.4583414270877838 |Validation err: 0.3315, Valida
Epoch 26: Train err: 0.209625, Train loss: 0.4519626965522766 |Validation err: 0.3435, Valid
Epoch 27: Train err: 0.22175, Train loss: 0.4636160457134247 |Validation err: 0.3215, Valida
Epoch 28: Train err: 0.219375, Train loss: 0.46314777398109436 |Validation err: 0.348, Valid
Epoch 29: Train err: 0.235875, Train loss: 0.49053542733192446 |Validation err: 0.326, Valid
Epoch 30: Train err: 0.22, Train loss: 0.4623157248497009 |Validation err: 0.3165, Validatio
Finished Training
Total time elapsed: 164.97 seconds
```

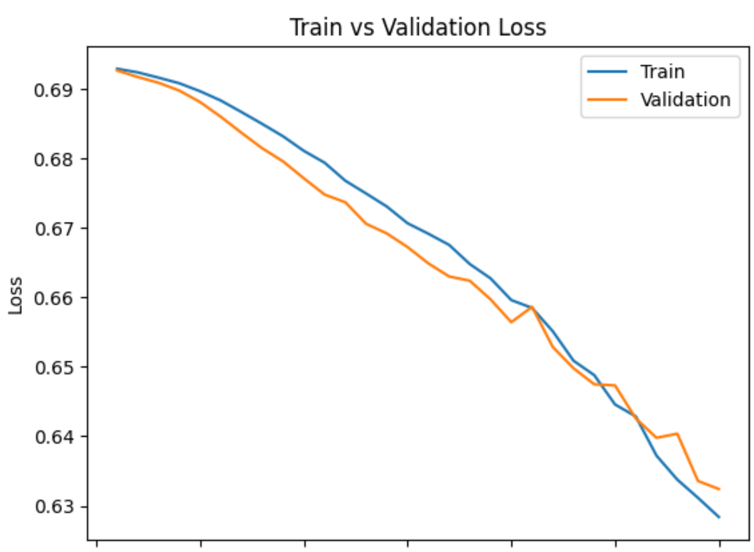
The model took 164.97 seconds to run, which is close to the same time as the default training parameters. Also, the validation error and loss are both increasing mostly throughout the epochs, which is a sign of overfitting.

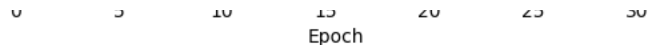
Part (c) - 3pt

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.

```
large_net = LargeNet()
train_net(large_net, 512, 0.01, 30)
large_path = get_model_name('large', 512, 0.01, 29)
plot_training_curve(large_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, Validati
Epoch 2: Train err: 0.457625, Train loss: 0.6924104019999504 |Validation err: 0.434, Validat
Epoch 3: Train err: 0.437, Train loss: 0.6916500590741634 |Validation err: 0.4265, Validatio
Epoch 4: Train err: 0.433625, Train loss: 0.6908449940383434 |Validation err: 0.424, Validat
Epoch 5: Train err: 0.434, Train loss: 0.6896935552358627 |Validation err: 0.424, Validation
Epoch 6: Train err: 0.438, Train loss: 0.688353206962347 |Validation err: 0.4285, Validation
Epoch 7: Train err: 0.439375, Train loss: 0.6866871677339077 |Validation err: 0.426, Validat
Epoch 8: Train err: 0.43525, Train loss: 0.6849770769476891 |Validation err: 0.4115, Validat
Epoch 9: Train err: 0.42375, Train loss: 0.6832009293138981 |Validation err: 0.414, Validati
Epoch 10: Train err: 0.421, Train loss: 0.6811089366674423 |Validation err: 0.416, Validatio
Epoch 11: Train err: 0.420875, Train loss: 0.6794026419520378 |Validation err: 0.4095, Valid
Epoch 12: Train err: 0.41475, Train loss: 0.6768048219382763 |Validation err: 0.412, Validat
Epoch 13: Train err: 0.4105, Train loss: 0.6749702803790569 |Validation err: 0.412, Validati
Epoch 14: Train err: 0.407125, Train loss: 0.6730880849063396 |Validation err: 0.4125, Valid
Epoch 15: Train err: 0.4005, Train loss: 0.6706806942820549 |Validation err: 0.4105, Validat
Epoch 16: Train err: 0.397625, Train loss: 0.6691771410405636 |Validation err: 0.405, Valida
Epoch 17: Train err: 0.393875, Train loss: 0.6675694733858109 |Validation err: 0.401, Valida
Epoch 18: Train err: 0.393, Train loss: 0.6648042872548103 |Validation err: 0.3945, Validati
Epoch 19: Train err: 0.38625, Train loss: 0.662746611982584 |Validation err: 0.388, Validati
Epoch 20: Train err: 0.38175, Train loss: 0.6596181839704514 |Validation err: 0.4005, Valida
Epoch 21: Train err: 0.38575, Train loss: 0.6584899798035622 |Validation err: 0.3885, Valida
Epoch 22: Train err: 0.378125, Train loss: 0.655123382806778 |Validation err: 0.3855, Valida
Epoch 23: Train err: 0.372125, Train loss: 0.6508794128894806 |Validation err: 0.3835, Valid
Epoch 24: Train err: 0.37675, Train loss: 0.6488028429448605 |Validation err: 0.385, Validat
Epoch 25: Train err: 0.368625, Train loss: 0.6445869170129299 |Validation err: 0.382, Valida
Epoch 26: Train err: 0.372625, Train loss: 0.6428566053509712 |Validation err: 0.3745, Valid
Epoch 27: Train err: 0.359375, Train loss: 0.6372117549180984 |Validation err: 0.379, Valida
Epoch 28: Train err: 0.35425, Train loss: 0.6337667480111122 |Validation err: 0.3695, Valida
Epoch 29: Train err: 0.3535, Train loss: 0.6311353109776974 |Validation err: 0.366, Validati
Epoch 30: Train err: 0.353, Train loss: 0.6283832415938377 |Validation err: 0.3675, Validati
Finished Training
Total time elapsed: 151.03 seconds
```





Increasing the batch size to 512 decreases the runtime of the training by around 15 seconds with a runtime of 151.03 seconds. It will result in an occurrence of overfitting since the training loss decreases while the validation loss increases.

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.

```
large_net = LargeNet()
train_net(large_net, 16, 0.01, 30)
large_path = get_model_name('large', 16, 0.01, 29)
plot_training_curve(large_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, Validati
Epoch 2: Train err: 0.369, Train loss: 0.639639899969101 |Validation err: 0.3465, Validation
Epoch 3: Train err: 0.34375, Train loss: 0.6098222947120666 |Validation err: 0.3325, Validat
Epoch 4: Train err: 0.314375, Train loss: 0.5849691489338875 |Validation err: 0.34, Validati
Epoch 5: Train err: 0.301125, Train loss: 0.5689119303822517 |Validation err: 0.3125, Valida
Epoch 6: Train err: 0.281, Train loss: 0.5452213581204415 |Validation err: 0.308, Validation
Epoch 7: Train err: 0.270875, Train loss: 0.5272981298565864 |Validation err: 0.307, Validat
Epoch 8: Train err: 0.259375, Train loss: 0.5070905526578426 |Validation err: 0.313, Validat
Epoch 9: Train err: 0.242375, Train loss: 0.4968344421982765 |Validation err: 0.313, Validat
Epoch 10: Train err: 0.236375, Train loss: 0.4756101597249508 |Validation err: 0.297, Valida
Epoch 11: Train err: 0.222125, Train loss: 0.4599769461452961 |Validation err: 0.2975, Valid
Epoch 12: Train err: 0.211, Train loss: 0.4454492371380329 |Validation err: 0.2995, Validati
Epoch 13: Train err: 0.19875, Train loss: 0.4245421719551086 |Validation err: 0.3075, Valida
Epoch 14: Train err: 0.18675, Train loss: 0.4007472907453775 |Validation err: 0.3085, Valida
Epoch 15: Train err: 0.1645, Train loss: 0.3759974058121443 |Validation err: 0.3105, Validat
Epoch 16: Train err: 0.16125, Train loss: 0.3591455406397581 |Validation err: 0.3005, Valida
Epoch 17: Train err: 0.15775, Train loss: 0.3463234790861607 |Validation err: 0.307, Validat
Epoch 18: Train err: 0.141625, Train loss: 0.32175366275012496 |Validation err: 0.3195, Vali
Epoch 19: Train err: 0.13375, Train loss: 0.30618105667084455 |Validation err: 0.335, Valida
Epoch 20: Train err: 0.126625, Train loss: 0.3029071792438626 |Validation err: 0.32, Validat
Epoch 21: Train err: 0.12025, Train loss: 0.28682796490937473 |Validation err: 0.3205, Valid
Epoch 22: Train err: 0.1165, Train loss: 0.27489088076353074 |Validation err: 0.352, Validat
```

The training error and loss are both decreasing throughout the 30 epochs, whereas the validation loss is sharply increasing throughout these 30 epochs. In addition, by decreasing the batch size, the runtime increases by almost a minute at 218.66 seconds.

Part 4. Hyperparameter Search [6 pt]

Part (a) - 2pt

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

Based on the above plots, another set of hyperparameters would be to use a batch size of 512 and a learning rate of 0.001 in the large network size. This is because when the batch size is 512 and the learning rate is 0.001 (from above), it is able to reduce the occurrences of overfitting in comparison to batch size 16 or learning rate 0.1.

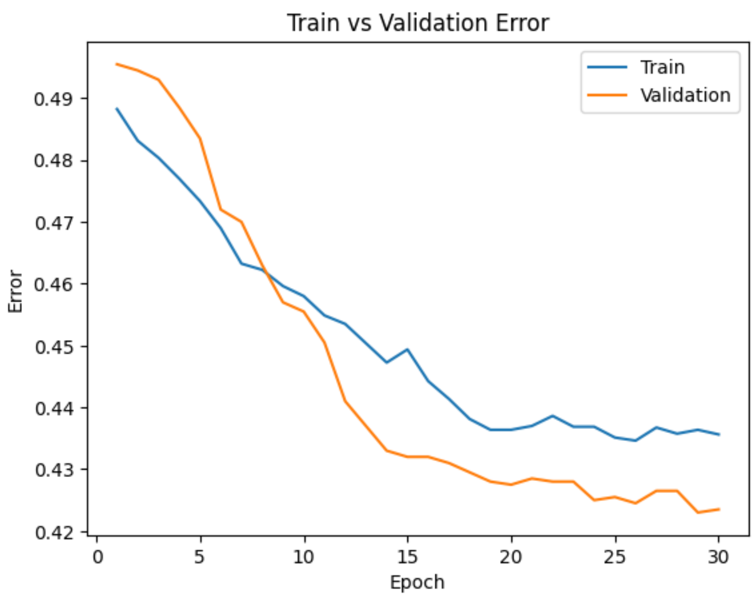
The reasoning for using the large network over the small network is because the large network less affected by noise.

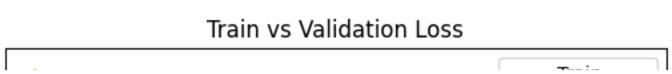
Part (b) - 1pt

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

```
large_net = LargeNet()
train_net(large_net, 512, 0.001, 30)
new_model = get_model_name('large', 512, 0.001, 29)
plot_training_curve(new_model)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.48825, Train loss: 0.6930677480995655 |Validation err: 0.4955, Validat
Epoch 2: Train err: 0.483125, Train loss: 0.692995510995388 |Validation err: 0.4945, Validat
Epoch 3: Train err: 0.480375, Train loss: 0.6929280497133732 |Validation err: 0.493, Validat
Epoch 4: Train err: 0.477, Train loss: 0.6928808391094208 |Validation err: 0.4885, Validatio
Epoch 5: Train err: 0.473375, Train loss: 0.692774411290884 |Validation err: 0.4835, Validat
Epoch 6: Train err: 0.469, Train loss: 0.6926896274089813 |Validation err: 0.472, Validation
Epoch 7: Train err: 0.46325, Train loss: 0.692620363086462 |Validation err: 0.47, Validation
Epoch 8: Train err: 0.46225, Train loss: 0.6925435550510883 |Validation err: 0.463, Validati
Epoch 9: Train err: 0.459625, Train loss: 0.6924680322408676 |Validation err: 0.457, Validat
Epoch 10: Train err: 0.458, Train loss: 0.6923965662717819 |Validation err: 0.4555, Validati
Epoch 11: Train err: 0.454875, Train loss: 0.6923230737447739 |Validation err: 0.4505, Valid
Epoch 12: Train err: 0.4535, Train loss: 0.6922412514686584 |Validation err: 0.441, Validati
Epoch 13: Train err: 0.450375, Train loss: 0.6921614557504654 |Validation err: 0.437, Valida
Epoch 14: Train err: 0.44725, Train loss: 0.6921032443642616 |Validation err: 0.433, Validat
Epoch 15: Train err: 0.449375, Train loss: 0.6920064650475979 |Validation err: 0.432, Valida
Epoch 16: Train err: 0.44425, Train loss: 0.6919283680617809 |Validation err: 0.432, Validat
Epoch 17: Train err: 0.441375, Train loss: 0.6918644718825817 |Validation err: 0.431, Valida
Epoch 18: Train err: 0.438125, Train loss: 0.6917712315917015 |Validation err: 0.4295, Valid
Epoch 19: Train err: 0.436375, Train loss: 0.6917018257081509 |Validation err: 0.428, Valida
Epoch 20: Train err: 0.436375, Train loss: 0.6915871091187 |Validation err: 0.4275, Validati
Epoch 21: Train err: 0.437, Train loss: 0.6915052235126495 |Validation err: 0.4285, Validati
Epoch 22: Train err: 0.438625, Train loss: 0.6914149634540081 |Validation err: 0.428, Valida
Epoch 23: Train err: 0.436875, Train loss: 0.6912974379956722 |Validation err: 0.428, Valida
Epoch 24: Train err: 0.436875, Train loss: 0.6912120543420315 |Validation err: 0.425, Valida
Epoch 25: Train err: 0.435125, Train loss: 0.6910865269601345 |Validation err: 0.4255, Valid
Epoch 26: Train err: 0.434625, Train loss: 0.6910119205713272 |Validation err: 0.4245, Valid
Epoch 27: Train err: 0.43675, Train loss: 0.6909283325076103 |Validation err: 0.4265, Valida
Epoch 28: Train err: 0.43575, Train loss: 0.6908275187015533 |Validation err: 0.4265, Valida
Epoch 29: Train err: 0.436375, Train loss: 0.6906765103340149 |Validation err: 0.423, Valida
Epoch 30: Train err: 0.435625, Train loss: 0.6905755028128624 |Validation err: 0.4235, Valid
Finished Training
Total time elapsed: 150.50 seconds
```





Part (c) - 2pt

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

The results from part a) takes into account overfitting as an issue, shown by the bottom graph. The new parameters that can be tried out are maintaining the 512 batches (since it seems to be working effectively) but lowering the iterations trained (to 20 epochs). This is because at ~epoch 15-20, the difference between the validation and training error/loss begins to deviate increasingly.

Part (d) - 1pt

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

```
large_net = LargeNet()
train_net(large_net, 512, 0.001, 20)
```

```
new_model = get_model_name('large', 512, 0.001, 19)
plot_training_curve(new_model)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.48825, Train loss: 0.6930677480995655 |Validation err: 0.4955, Validat
Epoch 2: Train err: 0.483125, Train loss: 0.692995510995388 |Validation err: 0.4945, Validat
Epoch 3: Train err: 0.480375, Train loss: 0.6929280497133732 |Validation err: 0.493, Validat
Epoch 4: Train err: 0.477, Train loss: 0.6928808391094208 |Validation err: 0.4885, Validatio
Epoch 5: Train err: 0.473375, Train loss: 0.692774411290884 |Validation err: 0.4835, Validat
Epoch 6: Train err: 0.469, Train loss: 0.6926896274089813 |Validation err: 0.472, Validation
Epoch 7: Train err: 0.46325, Train loss: 0.692620363086462 |Validation err: 0.47, Validation
Epoch 8: Train err: 0.46225, Train loss: 0.6925435550510883 |Validation err: 0.463, Validati
Epoch 9: Train err: 0.459625, Train loss: 0.6924680322408676 |Validation err: 0.457, Validat
Epoch 10: Train err: 0.458, Train loss: 0.6923965662717819 |Validation err: 0.4555, Validati
Epoch 11: Train err: 0.454875, Train loss: 0.6923230737447739 |Validation err: 0.4505, Valid
Epoch 12: Train err: 0.4535, Train loss: 0.6922412514686584 |Validation err: 0.441, Validati
Epoch 13: Train err: 0.450375, Train loss: 0.6921614557504654 |Validation err: 0.437, Valida
Epoch 14: Train err: 0.44725, Train loss: 0.6921032443642616 |Validation err: 0.433, Validat
Epoch 15: Train err: 0.449375, Train loss: 0.6920064650475979 |Validation err: 0.432, Valida
Epoch 16: Train err: 0.44425, Train loss: 0.6919283680617809 |Validation err: 0.432, Validat
Epoch 17: Train err: 0.441375, Train loss: 0.6918644718825817 |Validation err: 0.431, Valida
Epoch 18: Train err: 0.438125, Train loss: 0.6917712315917015 |Validation err: 0.4295, Valid
Epoch 19: Train err: 0.436375, Train loss: 0.6917018257081509 |Validation err: 0.428, Valida
Epoch 20: Train err: 0.436375, Train loss: 0.6915871091187 |Validation err: 0.4275, Validati
Finished Training
Total time elapsed: 101.08 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`.

```
net = LargeNet()
model_path = get_model_name(net.name, batch_size=512, learning_rate=0.01, epoch=20)
state = torch.load(model_path)
net.load_state_dict(state)
```

```
<All keys matched successfully>
```

Part (b) - 2pt

Justify your choice of model from part (a).

The reason for choosing a batch size of 512 is because in part 2, when increasing the batch size, it had reduced the effects of overfitting that were observed when using a batch size of 64.

The reason for choosing the large network over the small network is due to sheer number of parameters of the large network, it is less susceptible that the model will be affected by meaningless data, whereas meaningless data in a small network can potentially affect the performance of the model.

Finally, choosing to have 20 epochs was because it was noted in parts 3a and b that by the 20th epoch, the overfitting was beginning to become noticeable.

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.

```
# 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)
loss_func = nn.BCEWithLogitsLoss()
err, loss = evaluate(net, test_loader, loss_func)
print(err, loss)
```

```
Files already downloaded and verified
Files already downloaded and verified
0.3785 0.6553804576396942
```

Part (d) - 3pt

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

The test classification error is 0.3785, whereas the validation error is 0.4275. This is interesting because we expect the validation error to be lower than the test classification error. When validating, the model is exposed to the validation dataset repeatedly to properly tune the hyperparameters, while this is the first time for the test classification 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?

It is important to use the test dataset only at the end because we want to simulate a brand-new dataset being tested. If, for example, the model was built with an understanding of the test set, it can result in unwanted biases.

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.

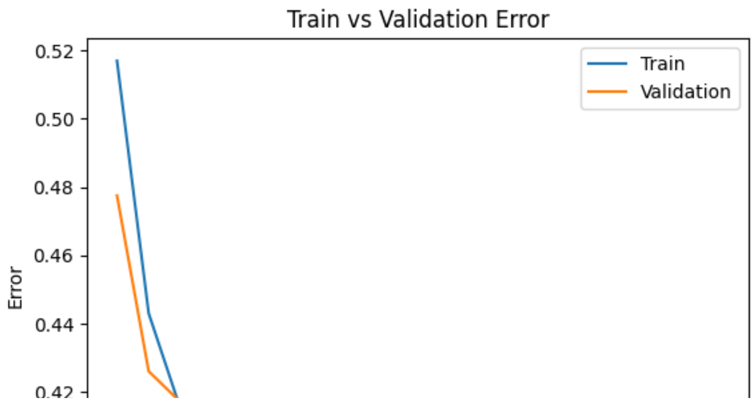
Hint: The ANN in lab 1 was applied on greyscale images. The cat and dog images are colour (RGB) and so you will need to flatted and concatinate all three colour layers before feeding them into an ANN.

```
# ANN taken from Lab 1 but now with 3 color channels instead of 1:
class Pigeon(nn.Module):
    def __init__(self):
      # add a name to the NN so that it can be recognized by get_model_name
        self.name = ('pigeon')
        super(Pigeon, self).__init__()
        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)
        # flattening to 1 dimension
        activation2 = activation2.squeeze(1)
        return activation2


# loading the network
pigeon = Pigeon()

# training the model with the same parameters
train_net(pigeon, 512, 0.001, 20)
# plot results
ANN = get_model_name('pigeon', 512, 0.001, 19)
plot_training_curve(ANN)
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1: Train err: 0.517, Train loss: 0.6953199170529842 |Validation err: 0.4775, Validatio
Epoch 2: Train err: 0.443, Train loss: 0.6882517263293266 |Validation err: 0.426, Validation
Epoch 3: Train err: 0.415125, Train loss: 0.6818411201238632 |Validation err: 0.417, Validat
Epoch 4: Train err: 0.406, Train loss: 0.676814254373312 |Validation err: 0.41, Validation l
Epoch 5: Train err: 0.403875, Train loss: 0.6727557927370071 |Validation err: 0.41, Validati
Epoch 6: Train err: 0.40125, Train loss: 0.6699470169842243 |Validation err: 0.408, Validati
Epoch 7: Train err: 0.39975, Train loss: 0.6673764772713184 |Validation err: 0.4085, Validat
Epoch 8: Train err: 0.398, Train loss: 0.6653114259243011 |Validation err: 0.4065, Validatio
Epoch 9: Train err: 0.3985, Train loss: 0.6636378429830074 |Validation err: 0.4025, Validati
Epoch 10: Train err: 0.398, Train loss: 0.6620821952819824 |Validation err: 0.4045, Validati
Epoch 11: Train err: 0.39625, Train loss: 0.6605739444494247 |Validation err: 0.403, Validat
Epoch 12: Train err: 0.393625, Train loss: 0.6588223725557327 |Validation err: 0.4025, Valid
Epoch 13: Train err: 0.39275, Train loss: 0.6578687913715839 |Validation err: 0.4, Validatio
Epoch 14: Train err: 0.39175, Train loss: 0.6565557643771172 |Validation err: 0.3995, Valida
Epoch 15: Train err: 0.389625, Train loss: 0.655500415712595 |Validation err: 0.399, Validat
Epoch 16: Train err: 0.389125, Train loss: 0.6545671746134758 |Validation err: 0.402, Valida
Epoch 17: Train err: 0.388, Train loss: 0.6536484584212303 |Validation err: 0.401, Validatio
Epoch 18: Train err: 0.386375, Train loss: 0.6525762900710106 |Validation err: 0.399, Valida
Epoch 19: Train err: 0.386125, Train loss: 0.6516962833702564 |Validation err: 0.3955, Valid
Epoch 20: Train err: 0.385625, Train loss: 0.6506311073899269 |Validation err: 0.3945, Valid
Finished Training
Total time elapsed: 82.17 seconds
```




```
train_loader, val_loader, test_loader, classes = get_data_loader(
    target_classes=["cat", "dog"],
    batch_size=512)
loss_func = nn.BCEWithLogitsLoss()
err, loss = evaluate(pigeon, test_loader, loss_func)
print(err, loss)
```

```
Files already downloaded and verified
Files already downloaded and verified
0.3805 0.6516173332929611
```

The performance of the 2 layer ANN on the test classification set is slightly worse than the CNN, with the error being 0.3805.