Lab 3: Gesture Recognition using Convolutional Neural Networks

In this lab you will train a convolutional neural network to make classifications on different hand gestures. By the end of the lab, you should be able to:

1. Load and split data for training, validation and testing
2. Train a Convolutional Neural Network
3. Apply transfer learning to improve your model

Note that for this lab we will not be providing you with any starter code. You should be able to take the code used in previous labs, tutorials and lectures and modify it accordingly to complete the tasks outlined below.

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. Make sure to review the PDF submission to ensure that your answers are easy to read. Make sure that your text is not cut off at the margins.

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.

Colab Link

Include a link to your colab file here

Colab Link: https://drive.google.com/file/d/1JCla5Ge0TNi2XObKjcJuHFodlEcWdpJe/view?usp=sharing

Dataset

American Sign Language (ASL) is a complete, complex language that employs signs made by moving the hands combined with facial expressions and postures of the body. It is the primary language of many North Americans who are deaf and is one of several communication options used by people who are deaf or hard-of-hearing. The hand gestures representing English alphabet are shown below. This lab focuses on classifying a subset of these hand gesture images using convolutional neural networks. Specifically, given an image of a hand showing one of the letters A-I, we want to detect which letter is being represented.

Part B. Building a CNN [50 pt]

For this lab, we are not going to give you any starter code. You will be writing a convolutional neural network from scratch. You are welcome to use any code from previous labs, lectures and tutorials. You should also write your own code.

You may use the PyTorch documentation freely. You might also find online tutorials helpful. However, all code that you submit must be your own.

Make sure that your code is vectorized, and does not contain obvious inefficiencies (for example, unecessary for loops, or unnecessary calls to unsqueeze()). Ensure enough comments are included in the code so that your TA can understand what you are doing. It is your responsibility to show that you understand what you write.

This is much more challenging and time-consuming than the previous labs. Make sure that you give yourself plenty of time by starting early.

1. Data Loading and Splitting [5 pt]

Download the anonymized data provided on Quercus. To allow you to get a heads start on this project we will provide you with sample data from previous years. Split the data into training, validation, and test sets.

Note: Data splitting is not as trivial in this lab. We want our test set to closely resemble the setting in which our model will be used. In particular, our test set should contain hands that are never seen in training!

Explain how you split the data, either by describing what you did, or by showing the code that you used. Justify your choice of splitting strategy. How many training, validation, and test images do you have?

For loading the data, you can use plt.imread as in Lab 1, or any other method that you choose. You may find torchvision.datasets.ImageFolder helpful. (see https://pytorch.org/docs/stable/torchvision/datasets.html?highlight=image%20folder#torchvision.datasets.ImageFolder)

In [1]:
```
import numpy as np
import time
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
from torchvision import transforms, datasets

torch.manual_seed(1)
```

Out[1]:
```
<torch._C.Generator at 0x1e7d9696b70>
```

In [2]:
```
# Load entire dataset (not split)
file_dir = "C:/Users/Admin/Downloads/Lab3_Dataset/Lab3_Gestures_Summer"
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),         # Convert to tensor
])

dataset = datasets.ImageFolder(root=file_dir, transform=transform)
```

In [3]:
```
classes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```

In [4]:
```
def get_data_loader(batch_size, overfitting=False):
    """ Splits the dataset into training, validation and testing datasets.
    Returns data loaders for the three preprocessed datasets.

    Args:
        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
    """
    # Define size
    train_size = int(0.8 * len(dataset))
    val_size = int(0.1 * len(dataset))
    test_size = len(dataset) - train_size - val_size
    overfit_size = 0
    if (overfitting):
        val_size = int(0.09 * len(dataset))
        test_size = int(0.09 * len(dataset))
        overfit_size = len(dataset) - train_size - val_size - test_size

    # Random split
    train_dataset, val_dataset, test_dataset, overfit_dataset = torch.utils.data.random_split(
        dataset, [train_size, val_size, test_size, overfit_size]
    )

    # Create data loaders for each set
    train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size)
    val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=batch_size)
```

```
    test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size)
    overfit_loader = torch.utils.data.DataLoader(overfit_dataset, batch_size=batch_size)

    return train_loader, val_loader, test_loader, overfit_loader
```

In [5]:

```
## TEST visualize dataset
import matplotlib.pyplot as plt

train_loader, val_loader, test_loader, overfit_loader = get_data_loader(batch_size = 1)

k = 0
for images, labels in train_loader:
    plt.subplot(3, 5, k+1)
    plt.axis('off')
    plt.imshow(np.transpose(images[0], (1, 2, 0)))

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

In [6]:

```
print("Training data size: ", len(train_loader) * train_loader.batch_size)
print("Validation data size: ", len(val_loader) * val_loader.batch_size)
print("Testing data size: ", len(test_loader) * test_loader.batch_size)
print("Overfitting data size: ", len(overfit_loader) * test_loader.batch_size)
```

```
Training data size:  1775
Validation data size:  221
Testing data size:  223
Overfitting data size:  0
```

2. Model Building and Sanity Checking [15 pt]

Part (a) Convolutional Network - 5 pt

Build a convolutional neural network model that takes the (224x224 RGB) image as input, and predicts the gesture letter. Your model should be a subclass of nn.Module. Explain your choice of neural network architecture: how many layers did you choose? What types of layers did you use? Were they fully-connected or convolutional? What about other decisions like pooling layers, activation functions, number of channels / hidden units?

Architecture:

3x224x224 (conv1) 5x220x220 (maxpool) 5x110x110 (conv2) 10x106x106 (pool) 10x53x53 (flattened, relu, fc1) 244 (fc2) 9 (Output)

- CNN, then feed to FC (good for image classification)
- ReLU activation function (recommended in lecture)
- Pooling: Max pooling (better than average pooling, from lecture)
- Number of channels & kernel size (based on lab 2 - Seems ok for Google Colab computation power)

In [7]:
```
class Gesture(nn.Module):
    def __init__(self, name):
        super(Gesture, self).__init__()
        self.name = name
        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 * 53 * 53, 244)
        self.fc2 = nn.Linear(244, 9)  # ABCDEFGHI

    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 * 53 * 53)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Part (b) Training Code - 5 pt

Write code that trains your neural network given some training data. Your training code should make it easy to tweak the usual hyperparameters, like batch size, learning rate, and the model object itself. Make sure that you are checkpointing your models from time to time (the frequency is up to you). Explain your choice of loss function and optimizer.

In [39]:
```
################################################################################
# Training Function
def train(model, overfitting=False, batch_size=30, learning_rate=0.001, num_epochs=30, replace=[
    ########################################################################
    # Fixed PyTorch random seed for reproducible result
    torch.manual_seed(100)
    ########################################################################
    # Obtain the PyTorch data loader objects to load batches of the datasets
    train_loader, val_loader, test_loader, overfit_loader = get_data_loader(batch_size, overfitt
    training = train_loader
    validating = val_loader
    if (overfitting):
        print("WARNING: only overfit testing should be here")
        training = overfit_loader
        validating = overfit_loader
    if (len(replace) > 0):
        print("WARNING: only transfer learning should be here")
        training = replace[0]
        validating = replace[1]
    ########################################################################
    # Define the Loss function and optimizer
    # The loss function will be Cross Entropy (CE) - since multiple classes involved.
    # We will use the nn.CrossEntropyLoss() - input expects logit (un-normalized output)
    # Optimizer will be Adam (recommended in class)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    ########################################################################
    # Set up some numpy arrays to store the training/test loss/error
    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.
    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(training, 0):
            # Get the inputs
            inputs, labels = data
            labels = F.one_hot(labels, len(classes))
            if torch.cuda.is_available():
                inputs = inputs.cuda()
                labels = labels.cuda()

            # Zero the parameter gradients
            optimizer.zero_grad()
            # Forward pass, backward pass, and optimize
            outputs = model(inputs)
            loss = criterion(outputs, labels.float())
            loss.backward()
            optimizer.step()
            # Calculate the statistics
            _, pred = outputs.max(dim=1)
            total_train_err += len(labels) - pred.eq(labels.argmax(dim=1)).sum().item()
            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(model, validating, 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(model.name, batch_size, learning_rate, epoch)
        torch.save(model.state_dict(), model_path)
    print('Finished Training')
    # 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)

################################################################################
# Helper Functions
def evaluate(model, loader, criterion):
    """ Evaluate the network on the validation set.

     Args:
         model: 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
     """
```

```python
    total_loss = 0.0
    total_err = 0.0
    total_epoch = 0
    for i, data in enumerate(loader, 0):
        inputs, labels = data
        labels = F.one_hot(labels, len(classes))
        if torch.cuda.is_available():
            inputs = inputs.cuda()
            labels = labels.cuda()

        outputs = model(inputs)

        ###### Statistics ######
        loss = criterion(outputs, labels.float())  # CE expects logits
        _, pred = outputs.max(dim=1)
        total_err += len(labels) - pred.eq(labels.argmax(dim=1)).sum().item()
        total_loss += loss.item()
        total_epoch += len(labels)
    err = float(total_err) / total_epoch
    loss = float(total_loss) / (i + 1)
    return err, loss

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 = "C:/Users/Admin/Downloads/lab3points/model_{0}_bs{1}_lr{2}_epoch{3}".format(name,
                                                   batch_size,
                                                   learning_rate,
                                                   epoch)
    return path

###############################################################################
# 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 (c) “Overfit” to a Small Dataset - 5 pt

One way to sanity check our neural network model and training code is to check whether the model is capable of “overfitting” or “memorizing” a small dataset. A properly constructed CNN with correct training code should be able to memorize the answers to a small number of images quickly.

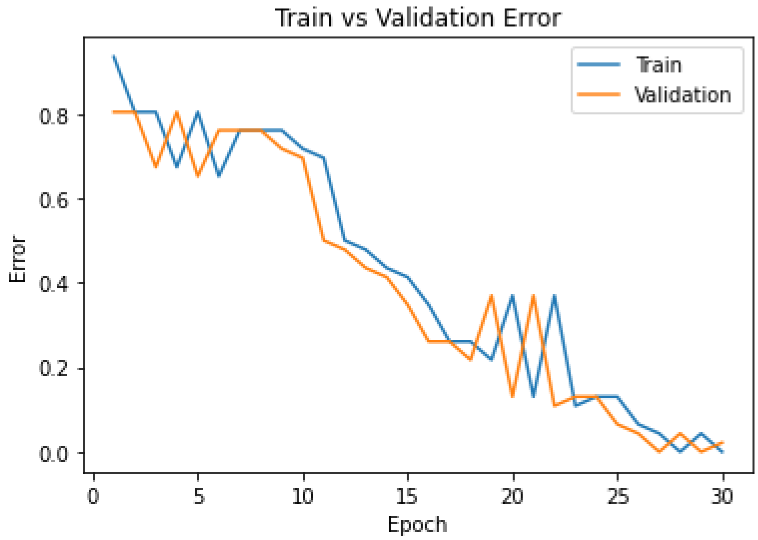
Construct a small dataset (e.g. just the images that you have collected). Then show that your model and training code is capable of memorizing the labels of this small data set.

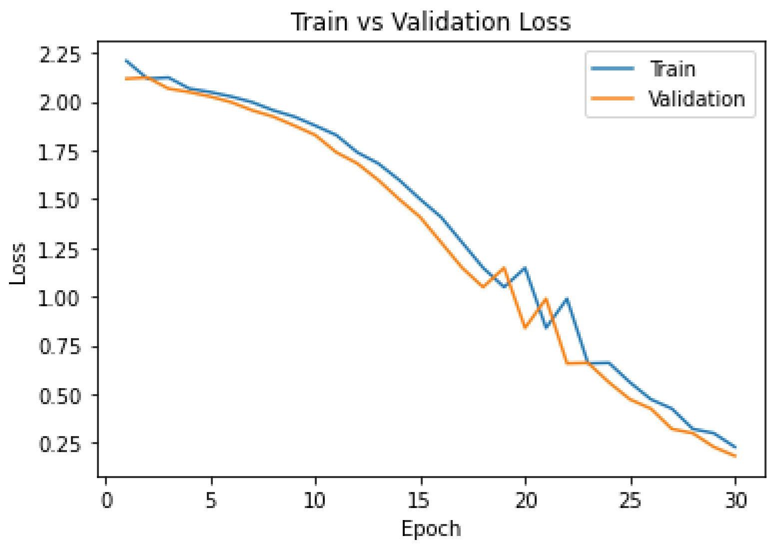
With a large batch size (e.g. the entire small dataset) and learning rate that is not too high, You should be able to obtain a 100% training accuracy on that small dataset relatively quickly (within 200 iterations).

In [9]:
```
model = Gesture("Overfit")
if torch.cuda.is_available():
    model = model.cuda()
train(model, overfitting=True, batch_size=46, learning_rate=0.001, num_epochs=30)
```

```
Epoch 1: Train err: 0.9347826086956522, Train loss: 2.208139657974243 | Validation err: 0.804347
8260869565, Validation loss: 2.117039442062378
Epoch 2: Train err: 0.8043478260869565, Train loss: 2.117039442062378 | Validation err: 0.804347
8260869565, Validation loss: 2.1230697631835938
Epoch 3: Train err: 0.8043478260869565, Train loss: 2.1230697631835938 | Validation err: 0.67391
30434782609, Validation loss: 2.0661749839782715
Epoch 4: Train err: 0.6739130434782609, Train loss: 2.0661749839782715 | Validation err: 0.80434
78260869565, Validation loss: 2.0482635498046875
Epoch 5: Train err: 0.8043478260869565, Train loss: 2.0482635498046875 | Validation err: 0.65217
39130434783, Validation loss: 2.0252439975738525
Epoch 6: Train err: 0.6521739130434783, Train loss: 2.0252439975738525 | Validation err: 0.76086
95652173914, Validation loss: 1.996256709098816
Epoch 7: Train err: 0.7608695652173914, Train loss: 1.996256709098816 | Validation err: 0.760869
5652173914, Validation loss: 1.9547805786132812
Epoch 8: Train err: 0.7608695652173914, Train loss: 1.9547805786132812 | Validation err: 0.76086
95652173914, Validation loss: 1.9224475622177124
Epoch 9: Train err: 0.7608695652173914, Train loss: 1.9224475622177124 | Validation err: 0.71739
1304347826, Validation loss: 1.8764595985412598
Epoch 10: Train err: 0.717391304347826, Train loss: 1.8764595985412598 | Validation err: 0.69565
21739130435, Validation loss: 1.8280154466629028
Epoch 11: Train err: 0.6956521739130435, Train loss: 1.8280154466629028 | Validation err: 0.5, V
alidation loss: 1.7397921085357666
Epoch 12: Train err: 0.5, Train loss: 1.7397921085357666 | Validation err: 0.4782608695652174, V
alidation loss: 1.6828545331954956
Epoch 13: Train err: 0.4782608695652174, Train loss: 1.6828545331954956 | Validation err: 0.4347
8260869565216, Validation loss: 1.597893476486206
Epoch 14: Train err: 0.43478260869565216, Train loss: 1.597893476486206 | Validation err: 0.4130
4347826086957, Validation loss: 1.4992364645004272
Epoch 15: Train err: 0.41304347826086957, Train loss: 1.4992364645004272 | Validation err: 0.347
82608695652173, Validation loss: 1.4072232246398926
Epoch 16: Train err: 0.34782608695652173, Train loss: 1.4072232246398926 | Validation err: 0.260
8695652173913, Validation loss: 1.2776827812194824
Epoch 17: Train err: 0.2608695652173913, Train loss: 1.2776827812194824 | Validation err: 0.2608
695652173913, Validation loss: 1.1478745937347412
Epoch 18: Train err: 0.2608695652173913, Train loss: 1.1478745937347412 | Validation err: 0.2173
9130434782608, Validation loss: 1.0486747026443481
Epoch 19: Train err: 0.21739130434782608, Train loss: 1.0486747026443481 | Validation err: 0.369
5652173913043, Validation loss: 1.1473547220230103
Epoch 20: Train err: 0.3695652173913043, Train loss: 1.1473547220230103 | Validation err: 0.1304
3478260869565, Validation loss: 0.8395739197731018
Epoch 21: Train err: 0.13043478260869565, Train loss: 0.8395739197731018 | Validation err: 0.369
5652173913043, Validation loss: 0.9892588257789612
Epoch 22: Train err: 0.3695652173913043, Train loss: 0.9892588257789612 | Validation err: 0.1086
9565217391304, Validation loss: 0.6576091051101685
Epoch 23: Train err: 0.10869565217391304, Train loss: 0.6576091051101685 | Validation err: 0.130
43478260869565, Validation loss: 0.660605788230896
Epoch 24: Train err: 0.13043478260869565, Train loss: 0.660605788230896 | Validation err: 0.1304
3478260869565, Validation loss: 0.5606409311294556
Epoch 25: Train err: 0.13043478260869565, Train loss: 0.5606409311294556 | Validation err: 0.065
21739130434782, Validation loss: 0.473480224609375
Epoch 26: Train err: 0.06521739130434782, Train loss: 0.473480224609375 | Validation err: 0.0434
78260869565216, Validation loss: 0.4257756173610687
Epoch 27: Train err: 0.043478260869565216, Train loss: 0.4257756173610687 | Validation err: 0.0,
Validation loss: 0.32152456045150757
Epoch 28: Train err: 0.0, Train loss: 0.32152456045150757 | Validation err: 0.04347826086956521
6, Validation loss: 0.29982027411460876
Epoch 29: Train err: 0.043478260869565216, Train loss: 0.29982027411460876 | Validation err: 0.
0, Validation loss: 0.2293560653924942
Epoch 30: Train err: 0.0, Train loss: 0.2293560653924942 | Validation err: 0.021739130434782608,
Validation loss: 0.18350203335285187
Finished Training
```

In [10]:
```
model_path = get_model_name("Overfit", batch_size=46, learning_rate=0.001, epoch=29)
plot_training_curve(model_path)
```





3. Hyperparameter Search [10 pt]

Part (a) - 1 pt

List 3 hyperparameters that you think are most worth tuning. Choose at least one hyperparameter related to the model architecture.

- Learning Rate
- Batch Size
- Amount of layers (We need to go deeper!)

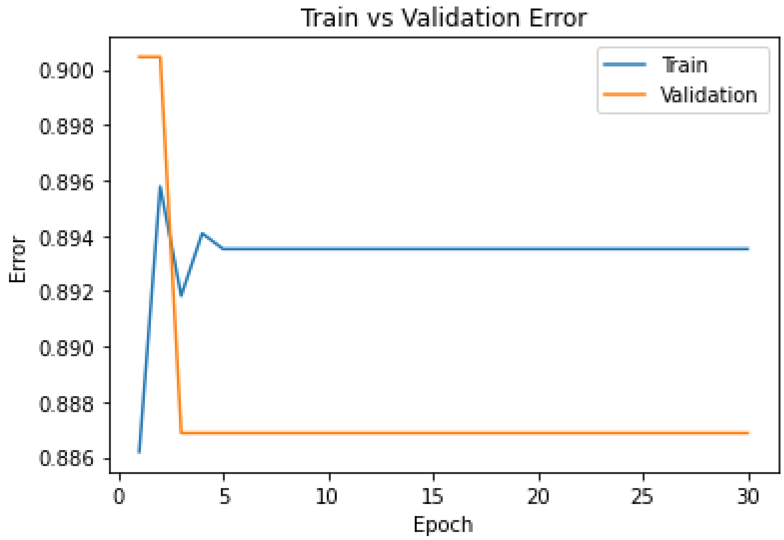
Part (b) - 5 pt

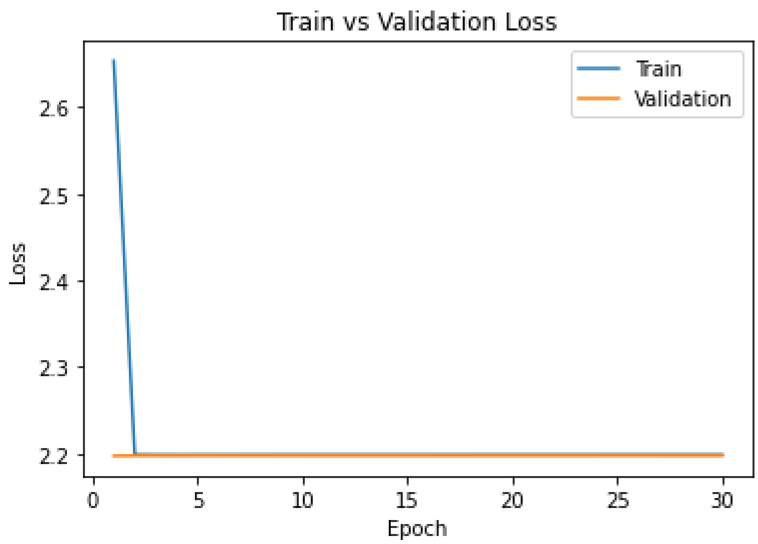
Tune the hyperparameters you listed in Part (a), trying as many values as you need to until you feel satisfied that you are getting a good model. Plot the training curve of at least 4 different hyperparameter settings.

In [12]:
```
# Using high learning rate
model = Gesture("IamSpeed")
if torch.cuda.is_available():
    model = model.cuda()
train(model, learning_rate=0.01)
```

```
Epoch 1: Train err: 0.8861971830985915, Train loss: 2.653315993150075 | Validation err: 0.900452
4886877828, Validation loss: 2.1972561180591583
Epoch 2: Train err: 0.895774647887324, Train loss: 2.198790693283081 | Validation err: 0.9004524
886877828, Validation loss: 2.1975536346435547
Epoch 3: Train err: 0.8918309859154929, Train loss: 2.198551913102468 | Validation err: 0.886877
8280542986, Validation loss: 2.197644978761673
Epoch 4: Train err: 0.8940845070422535, Train loss: 2.1984989802042643 | Validation err: 0.88687
78280542986, Validation loss: 2.1976890861988068
Epoch 5: Train err: 0.8935211267605634, Train loss: 2.198500128587087 | Validation err: 0.886877
8280542986, Validation loss: 2.19771015644073 5
Epoch 6: Train err: 0.8935211267605634, Train loss: 2.198524598280589 | Validation err: 0.886877
8280542986, Validation loss: 2.19770884513855
Epoch 7: Train err: 0.8935211267605634, Train loss: 2.1985485235850017 | Validation err: 0.88687
78280542986, Validation loss: 2.197702467441559
Epoch 8: Train err: 0.8935211267605634, Train loss: 2.1985684593518573 | Validation err: 0.88687
78280542986, Validation loss: 2.197695314884186
Epoch 9: Train err: 0.8935211267605634, Train loss: 2.1985846757888794 | Validation err: 0.88687
78280542986, Validation loss: 2.197688966989517
Epoch 10: Train err: 0.8935211267605634, Train loss: 2.1985979715983075 | Validation err: 0.8868
778280542986, Validation loss: 2.197684019804001
Epoch 11: Train err: 0.8935211267605634, Train loss: 2.1986090421676634 | Validation err: 0.8868
778280542986, Validation loss: 2.1976794600486755
Epoch 12: Train err: 0.8935211267605634, Train loss: 2.1986183961232504 | Validation err: 0.8868
778280542986, Validation loss: 2.1976757049560547
Epoch 13: Train err: 0.8935211267605634, Train loss: 2.1986263434092206 | Validation err: 0.8868
778280542986, Validation loss: 2.197672575712204
Epoch 14: Train err: 0.8935211267605634, Train loss: 2.1986332337061563 | Validation err: 0.8868
778280542986, Validation loss: 2.1976699233055115
Epoch 15: Train err: 0.8935211267605634, Train loss: 2.198639174302419 | Validation err: 0.88687
78280542986, Validation loss: 2.197667568922043
Epoch 16: Train err: 0.8935211267605634, Train loss: 2.1986443758010865 | Validation err: 0.8868
778280542986, Validation loss: 2.1976655423641205
Epoch 17: Train err: 0.8935211267605634, Train loss: 2.198648937543233 | Validation err: 0.88687
78280542986, Validation loss: 2.1976635456085205
Epoch 18: Train err: 0.8935211267605634, Train loss: 2.1986531019210815 | Validation err: 0.8868
778280542986, Validation loss: 2.1976619362831116
Epoch 19: Train err: 0.8935211267605634, Train loss: 2.198656709988912 | Validation err: 0.88687
78280542986, Validation loss: 2.1976605653762817
Epoch 20: Train err: 0.8935211267605634, Train loss: 2.1986599365870156 | Validation err: 0.8868
778280542986, Validation loss: 2.197659313678741 5
Epoch 21: Train err: 0.8935211267605634, Train loss: 2.19866289695104 | Validation err: 0.886877
8280542986, Validation loss: 2.1976580917835236
Epoch 22: Train err: 0.8935211267605634, Train loss: 2.198665503660838 | Validation err: 0.88687
78280542986, Validation loss: 2.1976568400859833
Epoch 23: Train err: 0.8935211267605634, Train loss: 2.198667903741201 | Validation err: 0.88687
78280542986, Validation loss: 2.1976559460163116
Epoch 24: Train err: 0.8935211267605634, Train loss: 2.1986700534820556 | Validation err: 0.8868
778280542986, Validation loss: 2.197655111551285
Epoch 25: Train err: 0.8935211267605634, Train loss: 2.198672036329905 | Validation err: 0.88687
78280542986, Validation loss: 2.1976543366909027
Epoch 26: Train err: 0.8935211267605634, Train loss: 2.19867384036382 | Validation err: 0.886877
8280542986, Validation loss: 2.19765368103981
Epoch 27: Train err: 0.8935211267605634, Train loss: 2.198675497372945 | Validation err: 0.88687
78280542986, Validation loss: 2.19765305519104
Epoch 28: Train err: 0.8935211267605634, Train loss: 2.198677043120066 | Validation err: 0.88687
78280542986, Validation loss: 2.1976523995399475
Epoch 29: Train err: 0.8935211267605634, Train loss: 2.1986783822377522 | Validation err: 0.8868
778280542986, Validation loss: 2.197651892900467
Epoch 30: Train err: 0.8935211267605634, Train loss: 2.1986796657244363 | Validation err: 0.8868
778280542986, Validation loss: 2.197651356458664
Finished Training
```

In [13]:
```
model_path = get_model_name("IamSpeed", batch_size=30, learning_rate=0.01, epoch=29)
plot_training_curve(model_path)
```





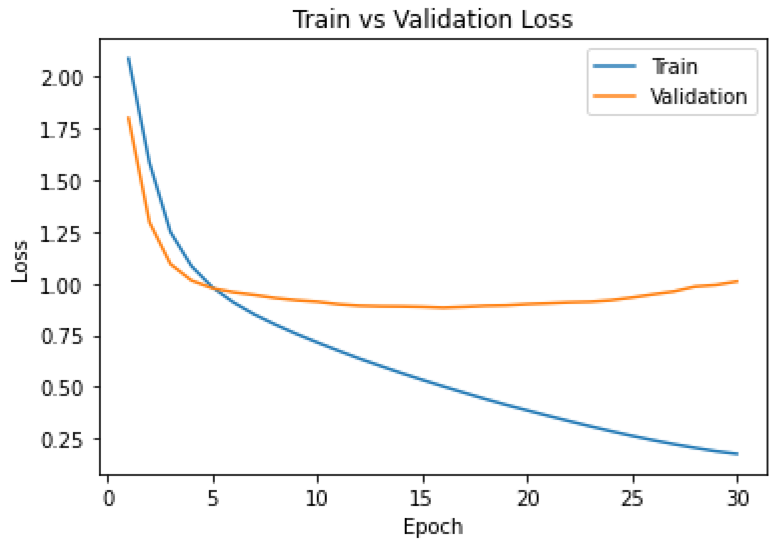
In [14]:

```
# Using small learning rate
model = Gesture("IamSlow")
if torch.cuda.is_available():
    model = model.cuda()
train(model, learning_rate=0.0001)
```

```
Epoch 1: Train err: 0.7397183098591549, Train loss: 2.089482625325521 | Validation err: 0.6108597285067874, Validation loss: 1.80134916305542
Epoch 2: Train err: 0.5397183098591549, Train loss: 1.5850840985774994 | Validation err: 0.4072398190045249, Validation loss: 1.2967552542686462
Epoch 3: Train err: 0.39774647887323944, Train loss: 1.2496534526348113 | Validation err: 0.33031674208144796, Validation loss: 1.0944920480251312
Epoch 4: Train err: 0.3340845070422535, Train loss: 1.0848199675480525 | Validation err: 0.31221719457013575, Validation loss: 1.0153076574206352
Epoch 5: Train err: 0.3104225352112676, Train loss: 0.9822133779525757 | Validation err: 0.31221719457013575, Validation loss: 0.9776335209608078
Epoch 6: Train err: 0.29070422535211266, Train loss: 0.9092458556095759 | Validation err: 0.29411764705882354, Validation loss: 0.9581030979752541
Epoch 7: Train err: 0.27098591549295775, Train loss: 0.850671782096227 | Validation err: 0.28054298642533937, Validation loss: 0.9451870694756508
Epoch 8: Train err: 0.2563380281690141, Train loss: 0.801503864924113 | Validation err: 0.27601809954751133, Validation loss: 0.9300251342356205
Epoch 9: Train err: 0.24845070422535212, Train loss: 0.7565458506345749 | Validation err: 0.27149321266968324, Validation loss: 0.9195008017122746
Epoch 10: Train err: 0.23380281690140844, Train loss: 0.7151138871908188 | Validation err: 0.27149321266968324, Validation loss: 0.9121885187923908
Epoch 11: Train err: 0.22422535211267605, Train loss: 0.6754198268055915 | Validation err: 0.26244343891402716, Validation loss: 0.8999729380011559
Epoch 12: Train err: 0.2123943661971831, Train loss: 0.6372159322102865 | Validation err: 0.2579185520361991, Validation loss: 0.8927652910351753
Epoch 13: Train err: 0.19830985915492957, Train loss: 0.6017636880278587 | Validation err: 0.248868778280543, Validation loss: 0.8902707509696484
Epoch 14: Train err: 0.1808450704225352, Train loss: 0.5671152859926224 | Validation err: 0.248868778280543, Validation loss: 0.8895285502076149
Epoch 15: Train err: 0.1667605633802817, Train loss: 0.5341965203483899 | Validation err: 0.24434389140271492, Validation loss: 0.8874035775661469
Epoch 16: Train err: 0.1543661971830986, Train loss: 0.5024464497963588 | Validation err: 0.23529411764705882, Validation loss: 0.8833070397377014
Epoch 17: Train err: 0.14535211267605633, Train loss: 0.4717484273016453 | Validation err: 0.22624434389140272, Validation loss: 0.8874070160090923
Epoch 18: Train err: 0.13859154929577464, Train loss: 0.4415501577158769 | Validation err: 0.23529411764705882, Validation loss: 0.8924807906150818
Epoch 19: Train err: 0.1295774647887324, Train loss: 0.41306864445408187 | Validation err: 0.248868778280543, Validation loss: 0.8946541845798492
Epoch 20: Train err: 0.11830985915492957, Train loss: 0.38640020253757634 | Validation err: 0.24434389140271492, Validation loss: 0.9005938209593296
Epoch 21: Train err: 0.10197183098591549, Train loss: 0.36006149016320704 | Validation err: 0.23076923076923078, Validation loss: 0.904574379324913
Epoch 22: Train err: 0.09352112676056339, Train loss: 0.334469689677159 | Validation err: 0.22624434389140272, Validation loss: 0.9092593491077423
Epoch 23: Train err: 0.08394366197183098, Train loss: 0.30943756687144436 | Validation err: 0.23529411764705882, Validation loss: 0.9118237048387527
Epoch 24: Train err: 0.07661971830985916, Train loss: 0.28565388011435666 | Validation err: 0.23076923076923078, Validation loss: 0.9194266386330128
Epoch 25: Train err: 0.06535211267605634, Train loss: 0.2631293329099814 | Validation err: 0.24434389140271492, Validation loss: 0.9325841702520847
Epoch 26: Train err: 0.06253521126760564, Train loss: 0.24215672469387453 | Validation err: 0.23529411764705882, Validation loss: 0.947666946798563
Epoch 27: Train err: 0.05802816901408451, Train loss: 0.22274858175466458 | Validation err: 0.24434389140271492, Validation loss: 0.9617388695478439
Epoch 28: Train err: 0.0523943661971831, Train loss: 0.20510432831943035 | Validation err: 0.248868778280543, Validation loss: 0.9857555069029331
Epoch 29: Train err: 0.044507042253521124, Train loss: 0.18863991120209297 | Validation err: 0.2579185520361991, Validation loss: 0.99337213113904
Epoch 30: Train err: 0.038873239436619716, Train loss: 0.17585000383357208 | Validation err: 0.26244343891402716, Validation loss: 1.0103040263056755
Finished Training
```

In [15]:

```
model_path = get_model_name("IamSlow", batch_size=30, learning_rate=0.0001, epoch=29)
plot_training_curve(model_path)
```

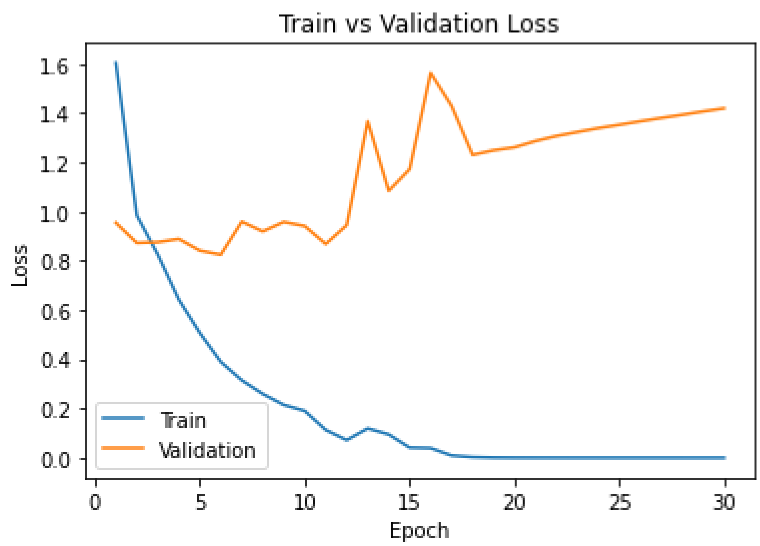


In [17]:
```
# Using smaller batch size
model = Gesture("IamNiche")
if torch.cuda.is_available():
    model = model.cuda()
train(model,learning_rate=0.001,batch_size=10)
```

```
Epoch 1: Train err: 0.5588732394366197, Train loss: 1.6038683247700165 | Validation err: 0.35746
60633484163, Validation loss: 0.9542100662770479
Epoch 2: Train err: 0.3335211267605634, Train loss: 0.9839736539661215 | Validation err: 0.30316
74208144796, Validation loss: 0.8721595665682917
Epoch 3: Train err: 0.2704225352112676, Train loss: 0.8247075809186764 | Validation err: 0.29411
764705882354, Validation loss: 0.8753649592399597
Epoch 4: Train err: 0.20788732394366197, Train loss: 0.6420469575802262 | Validation err: 0.2760
1809954751133, Validation loss: 0.8883483967055446
Epoch 5: Train err: 0.16901408450704225, Train loss: 0.506746675693587 | Validation err: 0.25791
85520361991, Validation loss: 0.8410679847002029
Epoch 6: Train err: 0.12845070422535212, Train loss: 0.3892295352612319 | Validation err: 0.2307
6923076923078, Validation loss: 0.8243717107435932
Epoch 7: Train err: 0.10535211267605633, Train loss: 0.31452892173808905 | Validation err: 0.248
868778280543, Validation loss: 0.9585151086034982
Epoch 8: Train err: 0.08732394366197183, Train loss: 0.25921955714891715 | Validation err: 0.235
29411764705882, Validation loss: 0.9190404159826753
Epoch 9: Train err: 0.06816901408450704, Train loss: 0.21483508553460576 | Validation err: 0.221
71945701357465, Validation loss: 0.9573011669455572
Epoch 10: Train err: 0.05971830985915493, Train loss: 0.1904104334351952 | Validation err: 0.266
9683257918552, Validation loss: 0.940740624928604
Epoch 11: Train err: 0.036619718309859155, Train loss: 0.11312431484602099 | Validation err: 0.2
0361990950226244, Validation loss: 0.867730961289572
Epoch 12: Train err: 0.023098591549295774, Train loss: 0.07192807377079304 | Validation err: 0.2
2171945701357465, Validation loss: 0.9445845984629831
Epoch 13: Train err: 0.03830985915492958, Train loss: 0.11929236457331938 | Validation err: 0.27
601809954751133, Validation loss: 1.3667503940341506
Epoch 14: Train err: 0.024788732394366197, Train loss: 0.09508363564171386 | Validation err: 0.2
2624434389140272, Validation loss: 1.0841436822412631
Epoch 15: Train err: 0.015211267605633802, Train loss: 0.04133176174766274 | Validation err: 0.2
171945701357466, Validation loss: 1.1720354431511268
Epoch 16: Train err: 0.009577464788732394, Train loss: 0.03949220019243962 | Validation err: 0.2
3076923076923078, Validation loss: 1.5628735855545686
Epoch 17: Train err: 0.0028169014084507044, Train loss: 0.009453807230650025 | Validation err:
0.19457013574660634, Validation loss: 1.4289549893355402
Epoch 18: Train err: 0.0, Train loss: 0.0031541401207309822 | Validation err: 0.1764705882352941
3, Validation loss: 1.2303220608483552
Epoch 19: Train err: 0.0, Train loss: 0.0010509374581461547 | Validation err: 0.1900452488687782
7, Validation loss: 1.2487919085211694
Epoch 20: Train err: 0.0, Train loss: 0.0005100881446175505 | Validation err: 0.1900452488687782
7, Validation loss: 1.261064467060746
Epoch 21: Train err: 0.0, Train loss: 0.0003401879156380285 | Validation err: 0.1900452488687782
7, Validation loss: 1.2864859074672752
Epoch 22: Train err: 0.0, Train loss: 0.00025972539161225227 | Validation err: 0.180995475113122
17, Validation loss: 1.3069774570606412
Epoch 23: Train err: 0.0, Train loss: 0.00021257766785525705 | Validation err: 0.180995475113122
17, Validation loss: 1.3232316346155615
Epoch 24: Train err: 0.0, Train loss: 0.00018001613685016983 | Validation err: 0.180995475113122
17, Validation loss: 1.3389510469214954
Epoch 25: Train err: 0.0, Train loss: 0.0001547368175876473 | Validation err: 0.1855203619909502
3, Validation loss: 1.3525881379647815
Epoch 26: Train err: 0.0, Train loss: 0.0001349771955629484 | Validation err: 0.1809954751131221
7, Validation loss: 1.366693511075062
Epoch 27: Train err: 0.0, Train loss: 0.00011886167661238728 | Validation err: 0.180995475113122
17, Validation loss: 1.380254660206642
Epoch 28: Train err: 0.0, Train loss: 0.00010513397822521501 | Validation err: 0.180995475113122
17, Validation loss: 1.3931754139105197
Epoch 29: Train err: 0.0, Train loss: 9.33245063400075e-05 | Validation err: 0.1809954751131221
7, Validation loss: 1.4068815842594777
Epoch 30: Train err: 0.0, Train loss: 8.324854170546407e-05 | Validation err: 0.1809954751131221
7, Validation loss: 1.41935036576773
Finished Training
```

In [18]:

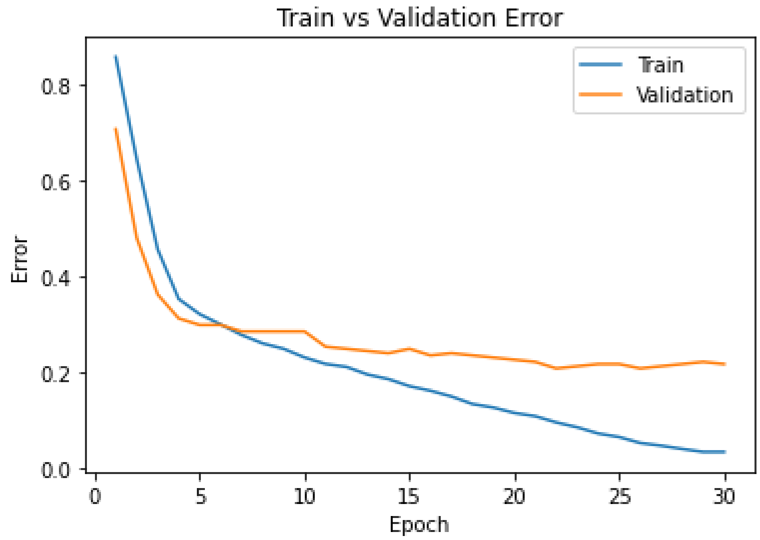
```python
model_path = get_model_name("IamNiche", batch_size=10, learning_rate=0.001, epoch=29)
plot_training_curve(model_path)
```

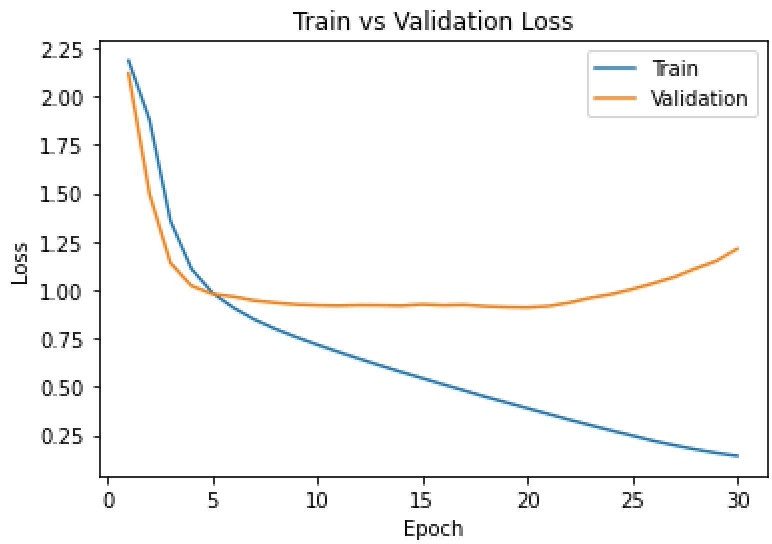


In [19]:
```
# Try out a different architecture with deeper layers
class DeepGesture(nn.Module):
    def __init__(self, name):
        super(DeepGesture, self).__init__()
        self.name = name
        self.conv1 = nn.Conv2d(3, 10, 5)
        self.conv2 = nn.Conv2d(10, 20, 3)
        self.conv3= nn.Conv2d(20, 40, 3)

        self.pool = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(40 * 26 * 26, 224)
        self.fc2 = nn.Linear(224, 9)  # ABCDEFGHI

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 40 * 26 * 26)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = DeepGesture("IamDeep")
if torch.cuda.is_available():
    model = model.cuda()
train(model, learning_rate=0.0001)
```

```
Epoch 1: Train err: 0.856338028169014, Train loss: 2.1845354795455934 | Validation err: 0.7058823529411765, Validation loss: 2.1191424429416656
Epoch 2: Train err: 0.643943661971831, Train loss: 1.8785146017869314 | Validation err: 0.4796380090497738, Validation loss: 1.4971020072698593
Epoch 3: Train err: 0.45577464788732397, Train loss: 1.3559428920348486 | Validation err: 0.36199095022624433, Validation loss: 1.140036553144455
Epoch 4: Train err: 0.3526760563380282, Train loss: 1.1086917221546173 | Validation err: 0.31221719457013575, Validation loss: 1.0222617462277412
Epoch 5: Train err: 0.3211267605633803, Train loss: 0.985493556658427 | Validation err: 0.2986425339366516, Validation loss: 0.9804055616259575
Epoch 6: Train err: 0.29971830985915493, Train loss: 0.908753048380216 | Validation err: 0.2986425339366516, Validation loss: 0.9664740189909935
Epoch 7: Train err: 0.27774647887323944, Train loss: 0.847885578374068 | Validation err: 0.2850678733031674, Validation loss: 0.9463155120611191
Epoch 8: Train err: 0.26028169014084507, Train loss: 0.7990952571233113 | Validation err: 0.2850678733031674, Validation loss: 0.9349631294608116
Epoch 9: Train err: 0.24901408450704227, Train loss: 0.7566410670677821 | Validation err: 0.2850678733031674, Validation loss: 0.9261490181088448
Epoch 10: Train err: 0.23098591549295774, Train loss: 0.7177986587087314 | Validation err: 0.2850678733031674, Validation loss: 0.9218388423323631
Epoch 11: Train err: 0.21746478873239436, Train loss: 0.6806270410617192 | Validation err: 0.25339366515837103, Validation loss: 0.9198045544326305
Epoch 12: Train err: 0.2112676056338028, Train loss: 0.6449140648047129 | Validation err: 0.248868778280543, Validation loss: 0.9226311519742012
Epoch 13: Train err: 0.19549295774647887, Train loss: 0.6102942059437434 | Validation err: 0.24434389140271492, Validation loss: 0.9218592904508114
Epoch 14: Train err: 0.18591549295774648, Train loss: 0.5770264692604542 | Validation err: 0.2398190045248869, Validation loss: 0.9197849631309509
Epoch 15: Train err: 0.1712676056338028, Train loss: 0.5449417454500993 | Validation err: 0.248868778280543, Validation loss: 0.9266286641359329
Epoch 16: Train err: 0.1616901408450704, Train loss: 0.5128569071491559 | Validation err: 0.23529411764705882, Validation loss: 0.9217727184295654
Epoch 17: Train err: 0.14985915492957746, Train loss: 0.4810715968410174 | Validation err: 0.2398190045248869, Validation loss: 0.9244362488389015
Epoch 18: Train err: 0.13408450704225353, Train loss: 0.44897813697655997 | Validation err: 0.23529411764705882, Validation loss: 0.9163704141974449
Epoch 19: Train err: 0.1267605633802817, Train loss: 0.4199917189776897 | Validation err: 0.23076923076923078, Validation loss: 0.9124659150838852
Epoch 20: Train err: 0.11549295774647887, Train loss: 0.3897153403609991 | Validation err: 0.22624434389140272, Validation loss: 0.9107358083128929
Epoch 21: Train err: 0.10873239436619718, Train loss: 0.36040155626833437 | Validation err: 0.22171945701357465, Validation loss: 0.9170565493404865
Epoch 22: Train err: 0.09577464788732394, Train loss: 0.33055752428869406 | Validation err: 0.2081447963800905, Validation loss: 0.935352448374033
Epoch 23: Train err: 0.0856338028169014, Train loss: 0.30267224833369255 | Validation err: 0.21266968325791855, Validation loss: 0.9595853686332703
Epoch 24: Train err: 0.07267605633802816, Train loss: 0.2749189084395766 | Validation err: 0.2171945701357466, Validation loss: 0.9787172898650169
Epoch 25: Train err: 0.06535211267605634, Train loss: 0.24870025012642144 | Validation err: 0.2171945701357466, Validation loss: 1.0048662573099136
Epoch 26: Train err: 0.05295774647887324, Train loss: 0.22200418865929047 | Validation err: 0.2081447963800905, Validation loss: 1.034901313483715
Epoch 27: Train err: 0.04732394366197183, Train loss: 0.19922662259389956 | Validation err: 0.21266968325791855, Validation loss: 1.0672782063484192
Epoch 28: Train err: 0.04056338028169014, Train loss: 0.177862517721951 | Validation err: 0.2171945701357466, Validation loss: 1.1108272522687912
Epoch 29: Train err: 0.03436619718309859, Train loss: 0.15950245624408127 | Validation err: 0.22171945701357465, Validation loss: 1.1500873267650604
Epoch 30: Train err: 0.03436619718309859, Train loss: 0.14405676880851387 | Validation err: 0.2171945701357466, Validation loss: 1.213636301457882
Finished Training
```

In [20]:
```python
model_path = get_model_name("IamDeep", batch_size=30, learning_rate=0.0001, epoch=29)
plot_training_curve(model_path)
```





Part (c) - 2 pt

Choose the best model out of all the ones that you have trained. Justify your choice.

All models we tried seem to have high Validation Loss over time, which is a sign of Overfitting (bad). However, between all of them, I would choose the last model (Deeper network, batch_size=30, learning_rate=0.0001, num_epoch=30) because its Validation error reaches the lowest value AND is flattened (consistent) overtime.

Part (d) - 2 pt

Report the test accuracy of your best model. You should only do this step once and prior to this step you should have only used the training and validation data.

In [21]:
```
criterion = nn.CrossEntropyLoss()
test_err, test_loss = evaluate(model, test_loader, criterion)

print("Test error:", test_err)
print("Test loss:", test_loss)
```

```
Test error: 0.10762331838565023
Test loss: 0.42889735606488305
```

4. Transfer Learning [15 pt]

For many image classification tasks, it is generally not a good idea to train a very large deep neural network model from scratch due to the enormous compute requirements and lack of sufficient amounts of training data.

One of the better options is to try using an existing model that performs a similar task to the one you need to solve. This method of utilizing a pre-trained network for other similar tasks is broadly termed **Transfer Learning**. In this assignment, we will use Transfer Learning to extract features from the hand gesture images. Then, train a smaller network to use these features as input and classify the hand gestures.

As you have learned from the CNN lecture, convolution layers extract various features from the images which get utilized by the fully connected layers for correct classification. AlexNet architecture played a pivotal role in establishing Deep Neural Nets as a go-to tool for image classification problems and we will use an ImageNet pre-trained AlexNet model to extract features in this assignment.

Part (a) - 5 pt

Here is the code to load the AlexNet network, with pretrained weights. When you first run the code, PyTorch will download the pretrained weights from the internet.

In [28]:
```
import torchvision.models
alexnet = torchvision.models.alexnet(pretrained=True)
```

The alexnet model is split up into two components: *alexnet.features* and *alexnet.classifier*. The first neural network component, *alexnet.features*, is used to compute convolutional features, which are taken as input in *alexnet.classifier*.

The neural network alexnet.features expects an image tensor of shape Nx3x224x224 as input and it will output a tensor of shape Nx256x6x6 . (N = batch size).

Compute the AlexNet features for each of your training, validation, and test data. Here is an example code snippet showing how you can compute the AlexNet features for some images (your actual code might be different):

Save the computed features. You will be using these features as input to your neural network in Part (b), and you do not want to re-compute the features every time. Instead, run *alexnet.features* once for each image, and save the result.

In [43]:
```
import os

# guaranteed that overfit set is separated from training and validation
transfer_train_loader, transfer_val_loader, transfer_test_loader, transfer_overfit_loader = get_

training_store = []
validation_store = []
testing_store = []
overfit_store = []

for img, label in transfer_train_loader:
    features = alexnet.features(img)
    features_tensor = torch.from_numpy(features.detach().numpy()).squeeze(0)
    training_store.append((features_tensor, label))

for img, label in transfer_val_loader:
```

```
    features = alexnet.features(img)
    features_tensor = torch.from_numpy(features.detach().numpy()).squeeze(0)
    validation_store.append((features_tensor, label))

for img, label in transfer_test_loader:
    features = alexnet.features(img)
    features_tensor = torch.from_numpy(features.detach().numpy()).squeeze(0)
    testing_store.append((features_tensor, label))

for img, label in transfer_overfit_loader:
    features = alexnet.features(img)
    features_tensor = torch.from_numpy(features.detach().numpy()).squeeze(0)
    overfit_store.append((features_tensor, label))
```

Part (b) - 3 pt

Build a convolutional neural network model that takes as input these AlexNet features, and makes a prediction. Your model should be a subclass of nn.Module.

Explain your choice of neural network architecture: how many layers did you choose? What types of layers did you use: fully-connected or convolutional? What about other decisions like pooling layers, activation functions, number of channels / hidden units in each layer?

Here is an example of how your model may be called:

In [44]:
```
# Only need classification layer since all CNN feature extractions are done through AlexNet
# Chose FC layer size based on original AlexNet design
# Activation: ReLU

class AlexNet(nn.Module):
    def __init__(self, name):
        super(AlexNet, self).__init__()
        self.name = name
        self.fc1 = nn.Linear(256 * 6 * 6, 32)
        self.fc2 = nn.Linear(32, 9)

    def forward(self, x):
        x = x.view(-1, 256 * 6 * 6)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Part (c) - 5 pt

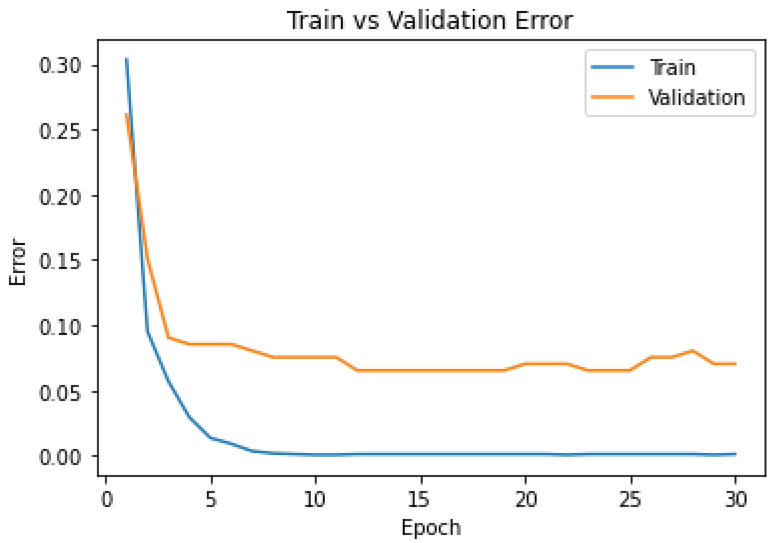
Train your new network, including any hyperparameter tuning. Plot and submit the training curve of your best model only.

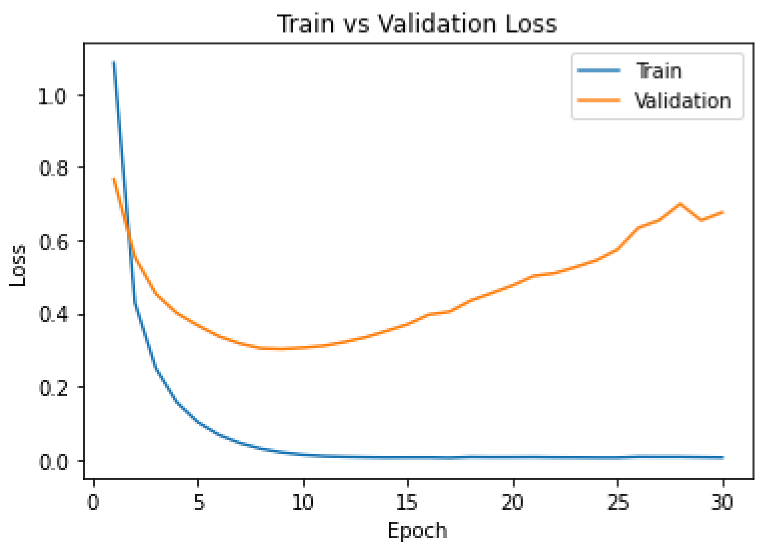
Note: Depending on how you are caching (saving) your AlexNet features, PyTorch might still be tracking updates to the **AlexNet weights**, which we are not tuning. One workaround is to convert your AlexNet feature tensor into a numpy array, and then back into a PyTorch tensor.

In [45]:
```
model = AlexNet("Transfer")
if torch.cuda.is_available():
    model = model.cuda()
train(model, overfitting=True, batch_size=30, learning_rate=0.0001, num_epochs=30, replace=[trai
```

```
WARNING: only overfit testing should be here
WARNING: only transfer learning should be here
Epoch 1: Train err: 0.3036619718309859, Train loss: 1.0847654354876615 | Validation err: 0.26130
65326633166, Validation loss: 0.7657881107312351
Epoch 2: Train err: 0.09521126760563381, Train loss: 0.42853839818142275 | Validation err: 0.150
7537688442211, Validation loss: 0.5529330226232358
Epoch 3: Train err: 0.056901408450704224, Train loss: 0.2489131877365732 | Validation err: 0.090
45226130653267, Validation loss: 0.45229160907179883
Epoch 4: Train err: 0.029295774647887324, Train loss: 0.15597657999262313 | Validation err: 0.08
542713567839195, Validation loss: 0.39956174211241446
Epoch 5: Train err: 0.013521126760563381, Train loss: 0.10176572216012779 | Validation err: 0.08
542713567839195, Validation loss: 0.36615841644703123
Epoch 6: Train err: 0.009014084507042254, Train loss: 0.06742422185551017 | Validation err: 0.08
542713567839195, Validation loss: 0.33652714205205125
Epoch 7: Train err: 0.0033802816901408453, Train loss: 0.04473412071367596 | Validation err: 0.0
8040201005025126, Validation loss: 0.31684677446842663
Epoch 8: Train err: 0.0016901408450704226, Train loss: 0.02920821819117132 | Validation err: 0.0
7537688442211055, Validation loss: 0.3036408945719775
Epoch 9: Train err: 0.0011267605633802818, Train loss: 0.01914534719272106 | Validation err: 0.0
7537688442211055, Validation loss: 0.3018788439733779
Epoch 10: Train err: 0.0005633802816901409, Train loss: 0.012600966079339125 | Validation err:
0.07537688442211055, Validation loss: 0.3052571710255885
Epoch 11: Train err: 0.0005633802816901409, Train loss: 0.009003958460711952 | Validation err:
0.07537688442211055, Validation loss: 0.31066260782775995
Epoch 12: Train err: 0.0011267605633802818, Train loss: 0.007256976771134037 | Validation err:
0.06532663316582915, Validation loss: 0.32115360193244546
Epoch 13: Train err: 0.0011267605633802818, Train loss: 0.006124770340234892 | Validation err:
0.06532663316582915, Validation loss: 0.3342275365027025
Epoch 14: Train err: 0.0011267605633802818, Train loss: 0.005240989505440927 | Validation err:
0.06532663316582915, Validation loss: 0.3513627858500299
Epoch 15: Train err: 0.0011267605633802818, Train loss: 0.005598771914923018 | Validation err:
0.06532663316582915, Validation loss: 0.36940493418324905
Epoch 16: Train err: 0.0011267605633802818, Train loss: 0.005582923383034132 | Validation err:
0.06532663316582915, Validation loss: 0.39580210003739835
Epoch 17: Train err: 0.0011267605633802818, Train loss: 0.004456707582628901 | Validation err:
0.06532663316582915, Validation loss: 0.403875485935086
Epoch 18: Train err: 0.0011267605633802818, Train loss: 0.006822154298080502 | Validation err:
0.06532663316582915, Validation loss: 0.4341712613727664
Epoch 19: Train err: 0.0011267605633802818, Train loss: 0.006179120398283187 | Validation err:
0.06532663316582915, Validation loss: 0.45454105023543706
Epoch 20: Train err: 0.0011267605633802818, Train loss: 0.006264986240026428 | Validation err:
0.07035175879396985, Validation loss: 0.47560062157676053
Epoch 21: Train err: 0.0011267605633802818, Train loss: 0.006532649485183593 | Validation err:
0.07035175879396985, Validation loss: 0.5013516043531382
Epoch 22: Train err: 0.0005633802816901409, Train loss: 0.005773988908090884 | Validation err:
0.07035175879396985, Validation loss: 0.508955570331471
Epoch 23: Train err: 0.0011267605633802818, Train loss: 0.005479529750946176 | Validation err:
0.06532663316582915, Validation loss: 0.5258917389151211
Epoch 24: Train err: 0.0011267605633802818, Train loss: 0.004958175305614107 | Validation err:
0.06532663316582915, Validation loss: 0.544205167477745
Epoch 25: Train err: 0.0011267605633802818, Train loss: 0.004968356897384879 | Validation err:
0.06532663316582915, Validation loss: 0.5736909306675745
Epoch 26: Train err: 0.0011267605633802818, Train loss: 0.007387196722680551 | Validation err:
0.07537688442211055, Validation loss: 0.633221154507613
Epoch 27: Train err: 0.0011267605633802818, Train loss: 0.00708199438816879 | Validation err: 0.
07537688442211055, Validation loss: 0.6539957123562324
Epoch 28: Train err: 0.0011267605633802818, Train loss: 0.007091564141300404 | Validation err:
0.08040201005025126, Validation loss: 0.6987219198467014
Epoch 29: Train err: 0.0005633802816901409, Train loss: 0.006157495049453532 | Validation err:
0.07035175879396985, Validation loss: 0.653696787518075
Epoch 30: Train err: 0.0011267605633802818, Train loss: 0.005313563793034858 | Validation err:
0.07035175879396985, Validation loss: 0.675511242401347
Finished Training
```

In [46]:
```
model_path = get_model_name("Transfer", batch_size=30, learning_rate=0.0001, epoch=29)
plot_training_curve(model_path)
```





Part (d) - 2 pt

Report the test accuracy of your best model. How does the test accuracy compare to Part 3(d) without transfer learning?

In [47]:
```
criterion = nn.CrossEntropyLoss()
test_err, test_loss = evaluate(model, testing_store, criterion)

print("Test error:", test_err)
print("Test loss:", test_loss)
```

```
Test error: 0.04522613065326633
Test loss: 0.20237362812151202
```

As we can see, this model (4.5% error, 20.2% loss) performs better compared to Part3(d) (10% error, 42.8% loss)

5. Additional Testing [5 pt]

For this question you should just reuse the small dataset that you used to answer question 2c.

Using the best transfer learning model developed in Part 4. Report the test accuracy on your sample images and how it compares to the test accuracy obtained in Part 4(d)? How well did your model do for the different hand guestures? Provide an explanation for why you think your model performed the way it did?

In [48]:
```
criterion = nn.CrossEntropyLoss()
test_err, test_loss = evaluate(model, overfit_store, criterion)

print("Test error:", test_err)
print("Test loss:", test_loss)
```

```
Test error: 0.06521739130434782
Test loss: 0.17022235944834394
```

The test accuracy is approximately the same as Part 4(d), only a little bit lower.

- Good accuracy since the AlexNet CNN has deeper layers and pre-trained weights that can extract image features very well.
- Quite similar to testing accuracy since the datasets are guaranteed to be separate (split from get_data_loader) so for the model, the test set and overfit set are similar (things that is has never seen before)
- Accuracy is a bit less than testing, also possibly because the number of data tested in this set is much smaller, so errors are more expensive