# Lab 3: Gesture Recognition using Convolutional Neural Networks

**Deadlines**: Feb 8, 5:00PM

**Late Penalty**: There is a penalty-free grace period of one hour past the deadline. Any work that is submitted between 1 hour and 24 hours past the deadline will receive a 20% grade deduction. No other late work is accepted. Quercus submission time will be used, not your local computer time. You can submit your labs as many times as you want before the deadline, so please submit often and early.

**Grading TAs**: Geoff Donoghue

This lab is based on an assignment developed by Prof. Lisa Zhang.

This lab will be completed in two parts. In Part A you will you will gain experience gathering your own data set (specifically images of hand gestures), and understand the challenges involved in the data cleaning process. In Part B 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. Generate and preprocess your own data
2. Load and split data for training, validation and testing
3. Train a Convolutional Neural Network
4. 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

**Submission for Part A:**
Submit a zip file containing your images. Three images each of American Sign Language gestures for letters A - I (total of 27 images). You will be required to clean the images before submitting them. Details are provided under Part A of the handout.

Individual image file names should follow the convention of student-number_Alphabet_file-number.jpg (e.g. 100343434_A_1.jpg).

**Submission for Part B:**
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/1QaOnfna2Cdmm80dPuGNltIIalLuzLzJH/view?usp=sharing (https://drive.google.com/file/d/1QaOnfna2Cdmm80dPuGNltIIalLuzLzJH/view?usp=sharing)

## Part A. Data Collection [10 pt]

So far, we have worked with data sets that have been collected, cleaned, and curated by machine learning researchers and practitioners. Datasets like MNIST and CIFAR are often used as toy examples, both by students and by researchers testing new machine learning models.

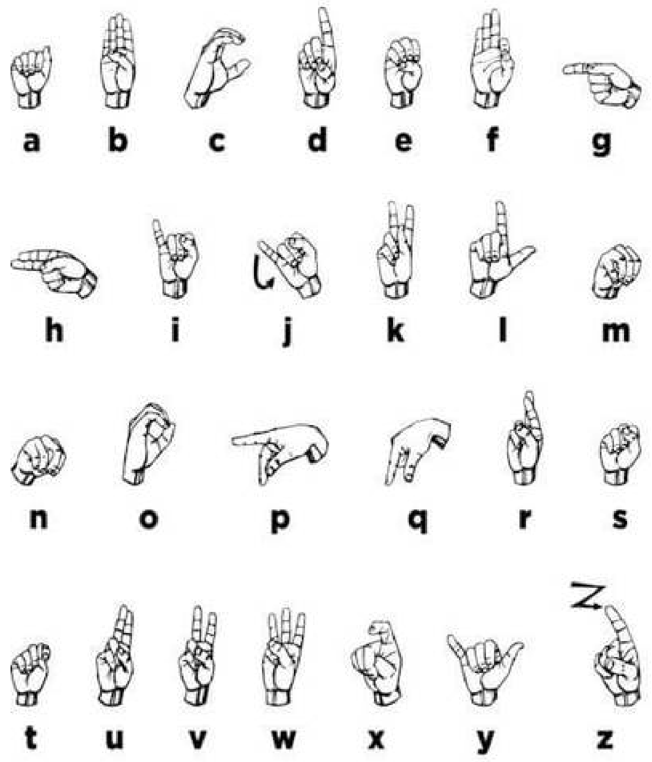
In the real world, getting a clean data set is never that easy. More than half the work in applying machine learning is finding, gathering, cleaning, and formatting your data set.

The purpose of this lab is to help you gain experience gathering your own data set, and understand the challenges involved in the data cleaning process.

### American Sign Language

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.



### Generating Data

We will produce the images required for this lab by ourselves. Each student will collect, clean and submit three images each of Americal Sign Language gestures for letters A - I (total of 27 images) Steps involved in data collection

1. Familiarize yourself with American Sign Language gestures for letters from A - I (9 letters).
2. Take three pictures at slightly different orientation for each letter gesture using your mobile phone.
    - Ensure adequate lighting while you are capturing the images.
    - Use a white wall as your background.
    - Use your right hand to create gestures (for consistency).
    - Keep your right hand fairly apart from your body and any other obstructions.
    - Avoid having shadows on parts of your hand.
3. Transfer the images to your laptop for cleaning.

### Cleaning Data

To simplify the machine learning the task, we will standardize the training images. We will make sure that all our images are of the same size (224 x 224 pixels RGB), and have the hand in the center of the cropped regions.

You may use the following applications to crop and resize your images:

**Mac**

- Use Preview: – Holding down CMD + Shift will keep a square aspect ratio while selecting the hand area. – Resize to 224x224 pixels.

**Windows 10**

- Use Photos app to edit and crop the image and keep the aspect ratio a square.
- Use Paint to resize the image to the final image size of 224x224 pixels.

**Linux**

- You can use GIMP, imagemagick, or other tools of your choosing. You may also use online tools such as http://picresize.com (http://picresize.com) All the above steps are illustrative only. You need not follow these steps but following these will ensure that you produce a good quality dataset. You will be judged based on the quality of the images alone. Please do not edit your photos in any other way. You should not need to change the aspect ratio of your image. You also should not digitally remove the background or shadows—instead, take photos with a white background and minimal shadows.

## Accepted Images

Images will be accepted and graded based on the criteria below

1. The final image should be size 224x224 pixels (RGB).
2. The file format should be a .jpg file.
3. The hand should be approximately centered on the frame.
4. The hand should not be obscured or cut off.
5. The photos follows the ASL gestures posted earlier.
6. The photos were not edited in any other way (e.g. no electronic removal of shadows or background).

## Submission

Submit a zip file containing your images. There should be a total of 27 images (3 for each category)

1. Individual image file names should follow the convention of student-number_Alphabet_file-number.jpg (e.g. 100343434_A_1.jpg)
2. Zip all the images together and name it with the following convention: last-name_student-number.zip (e.g. last-name_100343434.zip).
3. Submit the zipped folder. We will be anonymizing and combining the images that everyone submits. We will announce when the combined data set will be available for download.

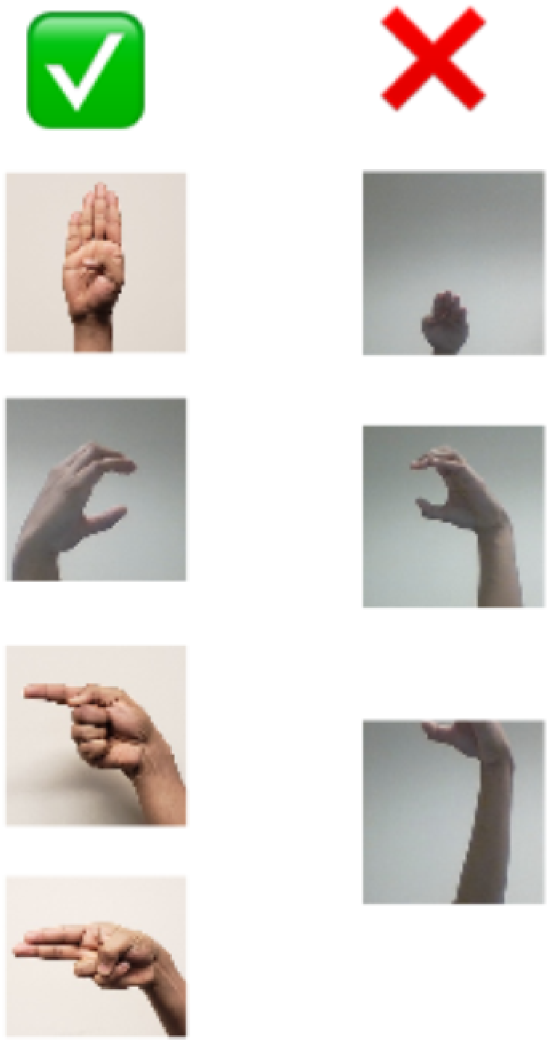


Figure 1: Acceptable Images (left) and Unacceptable Images (right)

## 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 (https://pytorch.org/docs/stable/torchvision/datasets.html?highlight=image%20folder#torchvision.datasets.ImageFolder) )

In [1]:
```
import time
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F

import matplotlib.pyplot as plt
import torch.optim as optim

# Use GPU
use_cuda = True
```

In [2]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

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

In [3]:
```python
import torchvision
import torchvision.transforms as transforms
from torch.utils.data.sampler import SubsetRandomSampler

def get_data_loader (data_path, batch_size=64):
  # Consistent seed
  torch.manual_seed(1)
  np.random.seed(1)

  # Transform PIL images 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)),
       transforms.Resize((224, 224))])

  # Retrieve data using ImageFolder() and the data_path passed in
  # Pass in the transformation
  data = torchvision.datasets.ImageFolder(root=data_path, transform=transform)

  # Randomize indices
  # Use a strict list of indices ensures that there are no intersecting data
  # points between the trainin, validation, and testing set. The shuffle()
  # function will be seeded and is reproducible.
  indices = np.arange(len(data))
  np.random.shuffle(indices)

  # 60%, 20%, 20% split between training, validation, and testing
  train_end = int(len(data) * 0.6)
  val_end = train_end + int(len(data) * 0.2)

  # Generate DataLoader objects
  train_sampler = SubsetRandomSampler(indices[:train_end])
  train_loader = torch.utils.data.DataLoader(data,
                                             batch_size=batch_size,
                                             sampler=train_sampler)

  val_sampler = SubsetRandomSampler(indices[train_end:val_end])
  val_loader = torch.utils.data.DataLoader(data,
                                           batch_size=batch_size,
                                           sampler=val_sampler)

  test_sampler = SubsetRandomSampler(indices[val_end:])
  test_loader = torch.utils.data.DataLoader(data,
                                            batch_size=batch_size,
                                            sampler=test_sampler)

  return train_loader, val_loader, test_loader
```

In [4]:
```python
def get_accuracy (model, data_loader):
  correct = 0
  total = 0

  for images, labels in data_loader:
    #############################################
    #To Enable GPU Usage
    if use_cuda and torch.cuda.is_available():
      images = images.cuda()
      labels = labels.cuda()
    #############################################

    output = model(images)

    # Select index with maximum prediction score
    pred = output.max(1, keepdim=True)[1]
    correct += pred.eq(labels.view_as(pred)).sum().item()
    total += images.shape[0]

  return correct / total
```

## 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?

In [5]:
```
# The CNN follows a similar structure to the CNN used for digit detection in
# LeNet5. I thought that detecting ASL might have similarities with detecting
# written digits, and used a very similar structure to LeNet5.

# The convolutional layers sandwich a max pooling layer. The kernels themselves
# are sized at 5x5, and the layers grow larger down the network. This is to
# allow fast and efficient feature extraction and filtering.

# The classification module is a 3-layer fully-connected neural network with
# 160 + 32 hidden units. This is a deep network that takes in the quickly
# filtered information and classifies it accordingly.

class ASLNet (nn.Module):
  def __init__ (self, kernel=5, input_size=(10 * 53 * 53)):
    super(ASLNet, self).__init__()
    self.kernel = kernel
    self.input_size = input_size

    self.conv1 = nn.Conv2d(3, 6, self.kernel)
    self.pool = nn.MaxPool2d(2, 2)
    self.conv2 = nn.Conv2d(6, 10, self.kernel)

    self.fc1 = nn.Linear(self.input_size, 120)
    self.fc2 = nn.Linear(120, 84)
    self.fc3 = nn.Linear(84, 9)

  def forward (self, x):
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))

    x = x.view(-1, self.input_size)

    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(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 [6]:
```
def train (model,

           data_path=None,
           train_loader=None,
           val_loader=None,

           batch_size=64,
           learning_rate=0.01,
           num_epochs=10):

  # Consistent seed
  torch.manual_seed(1)
  np.random.seed(1)

  # Retrieve data loaders
  # Set loss function and gradient descent optimizer
  if data_path is None:
    train_loader = train_loader
    val_loader = val_loader
  else:
    train_loader, val_loader, test_loader = get_data_loader(data_path=data_path, batch_size=bat
ch_size)

  criterion = nn.CrossEntropyLoss()
  optimizer = optim.Adam(model.parameters(), lr=learning_rate)

  #############################################
  #To Enable GPU Usage
  if use_cuda and torch.cuda.is_available():
    print('CUDA is available!  Training on GPU ...')
    criterion = criterion.cuda()
    model = model.cuda()
  #############################################

  epochs, losses, train_acc, val_acc = [], [], [], []

  # Training
  # Start timing
  start_time = time.time()
  for epoch in range(num_epochs):
    print("Epoch {}/{} ...".format(epoch + 1, num_epochs))

    for images, labels in train_loader:
      #############################################
      #To Enable GPU Usage
      if use_cuda and torch.cuda.is_available():
        images = images.cuda()
        labels = labels.cuda()
      #############################################

      out = model(images)
      loss = criterion(out, labels)
      loss.backward()
      optimizer.step()
      optimizer.zero_grad()

    epochs.append(epoch)
    losses.append(float(loss.item())/batch_size)
    train_acc.append(get_accuracy(model, train_loader))
    val_acc.append(get_accuracy(model, val_loader))

  # Plotting
  plt.title("Training Curve")
  plt.plot(epochs, losses, label="Train")
  plt.xlabel("Epochs")
  plt.ylabel("Loss")
  plt.show()

  plt.title("Training Curve")
  plt.plot(epochs, train_acc, label="Train")
  plt.plot(epochs, val_acc, label="Validation")
plt.xlabel("Epochs")
plt.ylabel("Training Accuracy")
plt.legend(loc='best')
plt.show()

print("Final Training Accuracy: {}".format(train_acc[-1]))
print("Final Validation Accuracy: {}".format(val_acc[-1]))

end_time = time.time()
elapsed_time = end_time - start_time
print("Total time elapsed: {:.2f} seconds".format(elapsed_time))
```

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

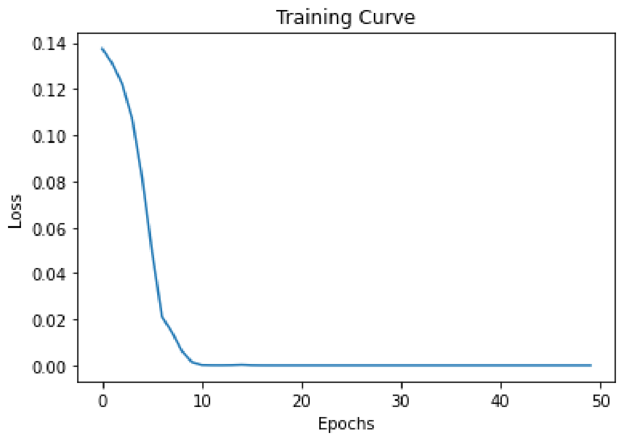
One way to sanity check our neural network model and training code is to check whether the model is capable of “overfitting” or “memorizing” a small dataset. A properly constructed CNN with correct training code should be able to memorize the answers to a small number of images quickly.

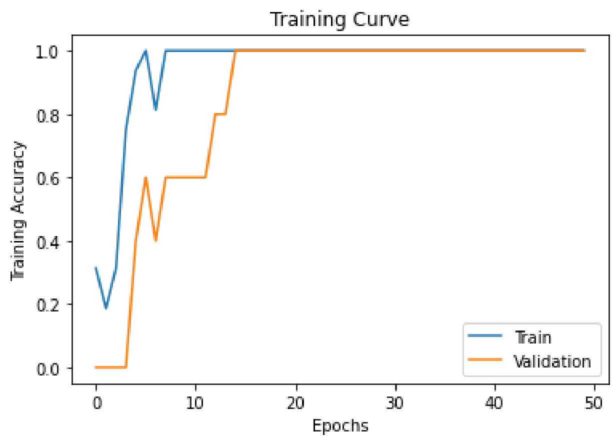
Construct a small dataset (e.g. just the images that you have collected). Then show that your model and training code is capable of memorizing the labels of this small data set.

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 [18]:
```
SmallNet = ASLNet()

# Set batch size to 16 (the whole training set)
# Since it is a smaller dataset, train slower
# Set the number of epochs to 50 << 200
train(SmallNet,
      data_path='/content/gdrive/MyDrive/Colab Notebooks/Labs/lee-1004112177',
      batch_size=16,
      learning_rate=0.0025,
      num_epochs=50)
```

```
CUDA is available!  Training on GPU ...
Epoch 1/50 ...
Epoch 2/50 ...
Epoch 3/50 ...
Epoch 4/50 ...
Epoch 5/50 ...
Epoch 6/50 ...
Epoch 7/50 ...
Epoch 8/50 ...
Epoch 9/50 ...
Epoch 10/50 ...
Epoch 11/50 ...
Epoch 12/50 ...
Epoch 13/50 ...
Epoch 14/50 ...
Epoch 15/50 ...
Epoch 16/50 ...
Epoch 17/50 ...
Epoch 18/50 ...
Epoch 19/50 ...
Epoch 20/50 ...
Epoch 21/50 ...
Epoch 22/50 ...
Epoch 23/50 ...
Epoch 24/50 ...
Epoch 25/50 ...
Epoch 26/50 ...
Epoch 27/50 ...
Epoch 28/50 ...
Epoch 29/50 ...
Epoch 30/50 ...
Epoch 31/50 ...
Epoch 32/50 ...
Epoch 33/50 ...
Epoch 34/50 ...
Epoch 35/50 ...
Epoch 36/50 ...
Epoch 37/50 ...
Epoch 38/50 ...
Epoch 39/50 ...
Epoch 40/50 ...
Epoch 41/50 ...
Epoch 42/50 ...
Epoch 43/50 ...
Epoch 44/50 ...
Epoch 45/50 ...
Epoch 46/50 ...
Epoch 47/50 ...
Epoch 48/50 ...
Epoch 49/50 ...
Epoch 50/50 ...
```





```
Final Training Accuracy: 1.0
Final Validation Accuracy: 1.0
Total time elapsed: 10.04 seconds
```

## 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.

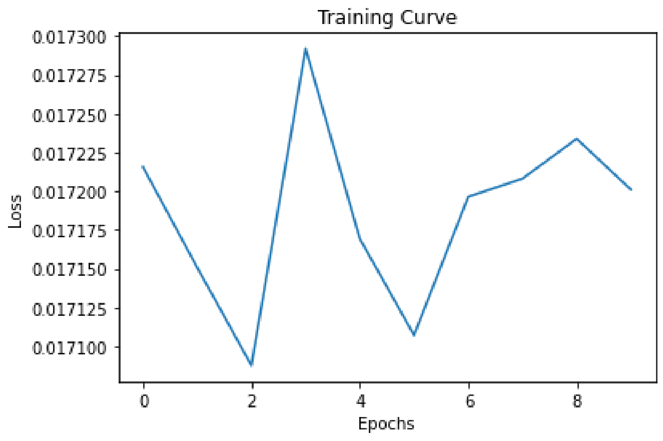
```
In [11]: # In my opinion, the 3 most important hyperparameters that are worth tuning are
         # batch size, learning rate, and size of convolutional layers. Batch size and
         # learning rate are two simple learnable hyperparamters that can vastly affect
         # the training quality and test accuracy. Convolutional layer sizes can control
         # how deep the information that is extracted and filtered out from the input.
```

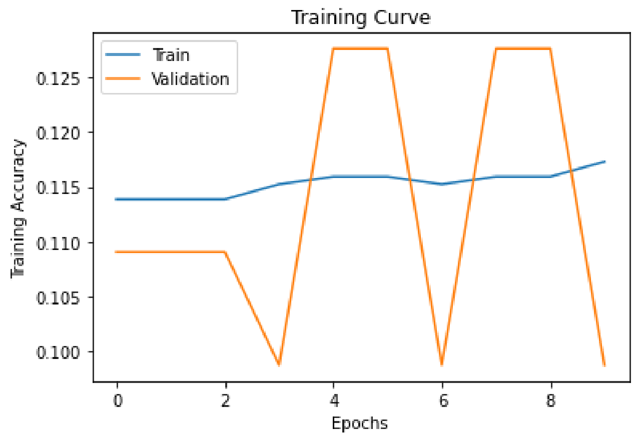
### Part (b) - 5 pt

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 [28]: RealNet = ASLNet()

         print("Training on increased batch size and learning rate ...")
         # Increase batch size
         # Increase learning rate
         train(RealNet,
               data_path='/content/gdrive/MyDrive/Colab Notebooks/Labs/Lab_3b_Gesture_Dataset',
               batch_size=128,
               learning_rate=0.02)
```

```
Training on increased batch size and learning rate ...
CUDA is available!  Training on GPU ...
Epoch 1/10 ...
Epoch 2/10 ...
Epoch 3/10 ...
Epoch 4/10 ...
Epoch 5/10 ...
Epoch 6/10 ...
Epoch 7/10 ...
Epoch 8/10 ...
Epoch 9/10 ...
Epoch 10/10 ...
```

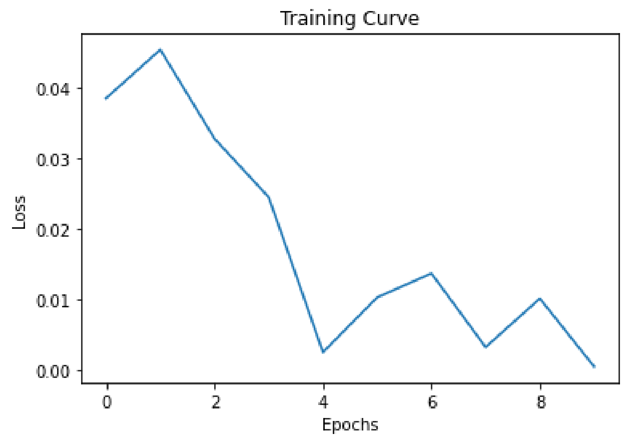


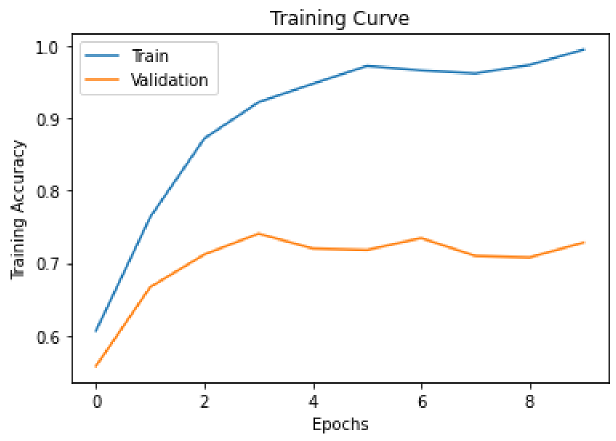


```
Final Training Accuracy: 0.11728395061728394
Final Validation Accuracy: 0.09876543209876543
Total time elapsed: 175.35 seconds
```

In [32]:
```python
RealNet = ASLNet()

print("Training on decreased batch size and learning rate ...")
# Decrease batch size
# Decrease learning rate
train(RealNet,
      data_path='/content/gdrive/MyDrive/Colab Notebooks/Labs/Lab_3b_Gesture_Dataset',
      batch_size=32,
      learning_rate=0.005)
```

```
Training on decreased batch size and learning rate ...
CUDA is available!  Training on GPU ...
Epoch 1/10 ...
Epoch 2/10 ...
Epoch 3/10 ...
Epoch 4/10 ...
Epoch 5/10 ...
Epoch 6/10 ...
Epoch 7/10 ...
Epoch 8/10 ...
Epoch 9/10 ...
Epoch 10/10 ...
```

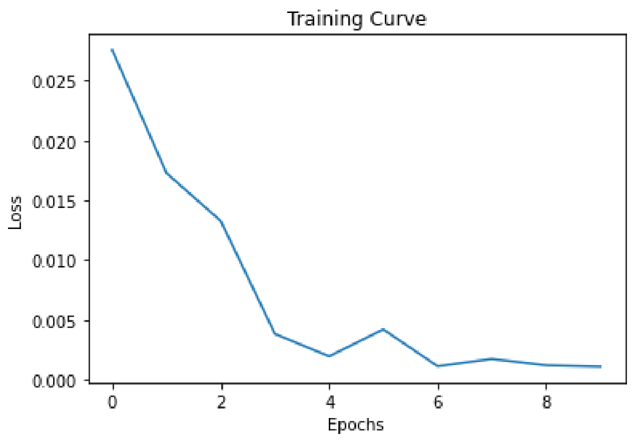


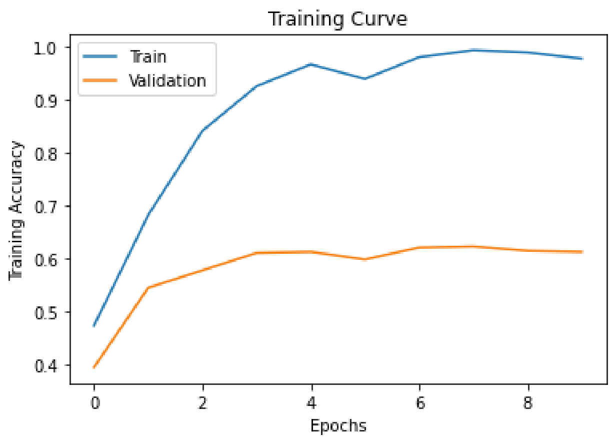


```
Final Training Accuracy: 0.99519890260631
Final Validation Accuracy: 0.7283950617283951
Total time elapsed: 130.81 seconds
```

In [30]:
```
print("Training on increased kernel size ...")
RealNet = ASLNet(kernel=7,
                 input_size=(10 * 51 * 51))
# Increase kernel size
train(RealNet,
      data_path='/content/gdrive/MyDrive/Colab Notebooks/Labs/Lab_3b_Gesture_Dataset')
```

```
Training on increased kernel size ...
CUDA is available!  Training on GPU ...
Epoch 1/10 ...
Epoch 2/10 ...
Epoch 3/10 ...
Epoch 4/10 ...
Epoch 5/10 ...
Epoch 6/10 ...
Epoch 7/10 ...
Epoch 8/10 ...
Epoch 9/10 ...
Epoch 10/10 ...
```

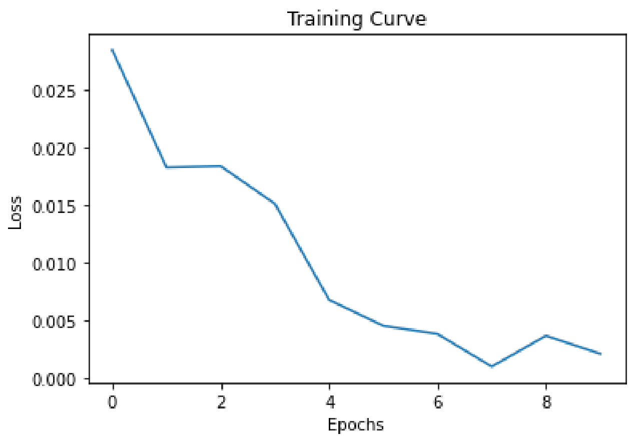


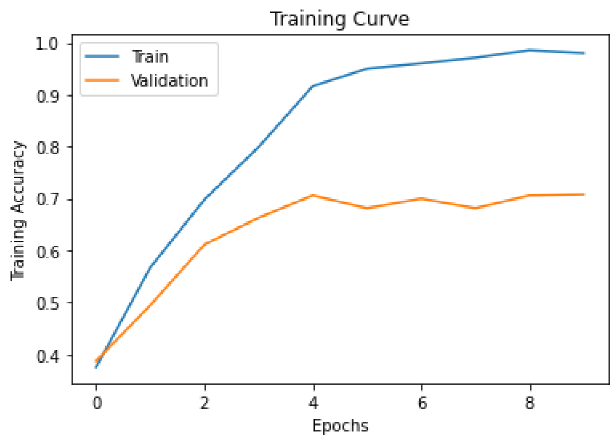


```
Final Training Accuracy: 0.9780521262002744
Final Validation Accuracy: 0.6131687242798354
Total time elapsed: 135.09 seconds
```

```
In [31]: print("Training on decreased kernel size ...")
         RealNet = ASLNet(kernel=3,
                          input_size=(10 * 54 * 54))
         # Decrease kernel size
         train(RealNet,
               data_path='/content/gdrive/MyDrive/Colab Notebooks/Labs/Lab_3b_Gesture_Dataset')
```

```
Training on decreased kernel size ...
CUDA is available!  Training on GPU ...
Epoch 1/10 ...
Epoch 2/10 ...
Epoch 3/10 ...
Epoch 4/10 ...
Epoch 5/10 ...
Epoch 6/10 ...
Epoch 7/10 ...
Epoch 8/10 ...
Epoch 9/10 ...
Epoch 10/10 ...
```





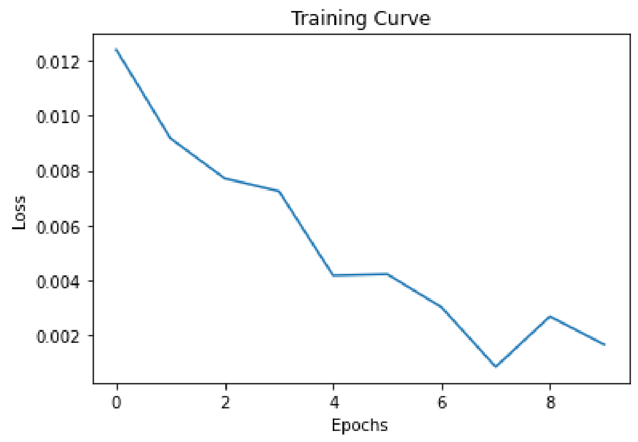
```
Final Training Accuracy: 0.9801097393689986
Final Validation Accuracy: 0.7078189300411523
Total time elapsed: 133.75 seconds
```

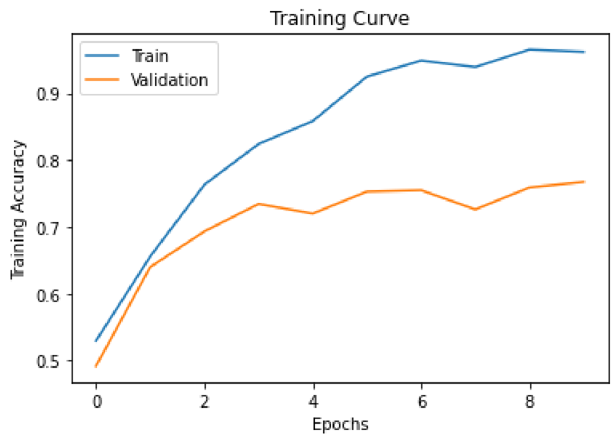
## Part (c) - 2 pt

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

In [33]:
```
# The model with decreased batch size and learning rate had the highest
# validation accuracy amongst the 4 models, which is the accuracy usually used
# to determine hyperparameter quality. Both the training and validation curves
# are smooth and non-erratic, which is to be expected of a low learning rate.

# I've decided to tweak some of the numbers a little bit to yield a better
# validation accuracy.

BestModel = ASLNet()

print("Training on decreased batch size and learning rate ...")
# Decrease batch size
# Decrease learning rate
train(BestModel,
      data_path='/content/gdrive/MyDrive/Colab Notebooks/Labs/Lab_3b_Gesture_Dataset',
      batch_size=128,
      learning_rate=0.0025)
```

```
Training on decreased batch size and learning rate ...
CUDA is available!  Training on GPU ...
Epoch 1/10 ...
Epoch 2/10 ...
Epoch 3/10 ...
Epoch 4/10 ...
Epoch 5/10 ...
Epoch 6/10 ...
Epoch 7/10 ...
Epoch 8/10 ...
Epoch 9/10 ...
Epoch 10/10 ...
```





```
Final Training Accuracy: 0.9622770919067215
Final Validation Accuracy: 0.7674897119341564
Total time elapsed: 137.29 seconds
```

## 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 [34]:
```
train_loader, val_loader, test_loader = get_data_loader(
    data_path='/content/gdrive/MyDrive/Colab Notebooks/Labs/Lab_3b_Gesture_Dataset')

print("Test accuracy: {}".format(get_accuracy(BestModel, test_loader)))
```

```
Test accuracy: 0.7741273100616016
```

## 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 [22]:
```
import torchvision.models
alexnet = torchvision.models.alexnet(pretrained=True)
```

```
Downloading: "https://download.pytorch.org/models/alexnet-owt-4df8aa71.pth" to /root/.cache/t
orch/hub/checkpoints/alexnet-owt-4df8aa71.pth
```

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):

In [35]:
```
# img = ... a PyTorch tensor with shape [N,3,224,224] containing hand images ...
#features = alexnet.features(img)
```

**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 [53]:
```
# Note that this code block has already been run once, and does not need to be
# run again.

import os

label_name = (
  'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I'
)

def save_features (data_path, data_loader):
  # Number of iterations
  n = 0

  for images, labels in data_loader:
    features = alexnet.features(images)
    # Conver to Tensor for PyTorch save() function
    f_tensor = torch.from_numpy(features.detach().numpy())

    # Generate folder name
    folder_name = data_path + "/" + str(label_name[labels])

    # Generate in case it is missing
    if not os.path.isdir(folder_name):
      os.mkdir(folder_name)

    # Make sure to squeeze to clean up the Tensor
    torch.save(f_tensor.squeeze(0), folder_name + "/" + str(n) + ".tensor")

    n += 1

# Retrieve DataLoaders with batch size 1
train_loader, val_loader, test_loader = get_data_loader(
    data_path='/content/gdrive/MyDrive/Colab Notebooks/Labs/Lab_3b_Gesture_Dataset',
    batch_size=1)

# Save the features
print("Saving features (training) ...")
save_features('/content/gdrive/MyDrive/Colab Notebooks/Labs/Lab_3b_Features/train', train_loade
r)
print("Saving features (validation) ...")
save_features('/content/gdrive/MyDrive/Colab Notebooks/Labs/Lab_3b_Features/val', val_loader)
print("Saving features (testing) ...")
save_features('/content/gdrive/MyDrive/Colab Notebooks/Labs/Lab_3b_Features/test', test_loader)
```

```
Saving features (training) ...
Saving features (validation) ...
Saving features (testing) ...
```

## 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 [54]:
```
# features = ... load precomputed alexnet.features(img) ...
#output = model(features)
#prob = F.softmax(output)
```

```
In [77]: # Since the input dimensions are 256 x 6 x 6, the convolutional kernels don't
         # need to be large. I used 2 convolutional layers with 2x2 kernels. These
         # filters are then fed into a 2-layer fully-connected network that performs the
         # final classification. It has 50 hidden units to account for the fact that it
         # does not have a third layer. I found through trial and error that having
         # 3 layers resulted in overfitting and poor validation accuracy.

         class ASL_AlexNet(nn.Module):
           def __init__ (self):
             super(ASL_AlexNet, self).__init__()

             self.conv1 = nn.Conv2d(256, 50, 2)
             self.conv2 = nn.Conv2d(50, 25, 2)

             self.fc1 = nn.Linear(25 * 4 * 4, 50)
             self.fc2 = nn.Linear(50, 9)

           def forward (self, x):
             x = F.relu(self.conv1(x))
             x = F.relu(self.conv2(x))

             x = x.view(-1, 25 * 4 * 4)

             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.

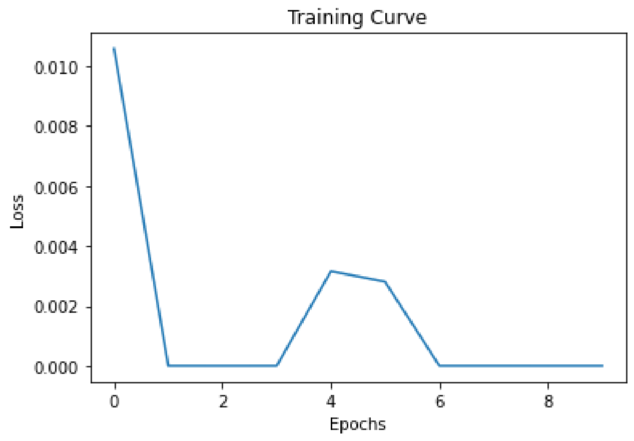
Note: Depending on how you are caching (saving) your AlexNet features, PyTorch might still be tracking updates to the **AlexNet weights**, which we are not tuning. One workaround is to convert your AlexNet feature tensor into a numpy array, and then back into a PyTorch tensor.

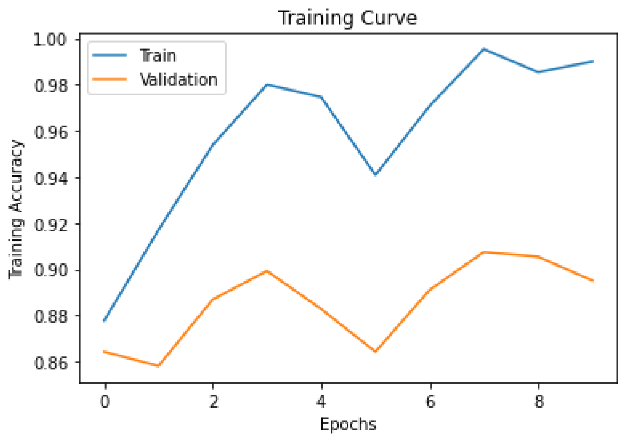
```
In [56]: #tensor = torch.from_numpy(tensor.detach().numpy())
```

In [82]:

```python
# Load the features back in
train_features = torchvision.datasets.DatasetFolder(
    '/content/gdrive/MyDrive/Colab Notebooks/Labs/Lab_3b_Features/train',
    loader=torch.load,
    extensions=('.tensor'))
val_features = torchvision.datasets.DatasetFolder(
    '/content/gdrive/MyDrive/Colab Notebooks/Labs/Lab_3b_Features/val',
    loader=torch.load,
    extensions=('.tensor'))
test_features = torchvision.datasets.DatasetFolder(
    '/content/gdrive/MyDrive/Colab Notebooks/Labs/Lab_3b_Features/test',
    loader=torch.load,
    extensions=('.tensor'))

# Generate DataLoader objects
train_f_loader = torch.utils.data.DataLoader(train_features,
                                             shuffle=True,
                                             drop_last=True)
val_f_loader = torch.utils.data.DataLoader(val_features,
                                           shuffle=True,
                                           drop_last=True)
test_f_loader = torch.utils.data.DataLoader(test_features,
                                            shuffle=True,
                                            drop_last=True)

# Create the new network
AdvancedNet = ASL_AlexNet()

if use_cuda and torch.cuda.is_available():
  AdvancedNet = AdvancedNet.cuda()

train(AdvancedNet,
      train_loader=train_f_loader,
      val_loader=val_f_loader,
      batch_size=128,
      learning_rate=0.0005)
```

```
CUDA is available!  Training on GPU ...
Epoch 1/10 ...
Epoch 2/10 ...
Epoch 3/10 ...
Epoch 4/10 ...
Epoch 5/10 ...
Epoch 6/10 ...
Epoch 7/10 ...
Epoch 8/10 ...
Epoch 9/10 ...
Epoch 10/10 ...
```





```
Final Training Accuracy: 0.99
Final Validation Accuracy: 0.8950617283950617
Total time elapsed: 77.66 seconds
```

### 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 [83]:
```
# The test accuracy is a higher than the test accuracy from Part 3(d).
# AlexNet, in particular, is very good at extracting important features from
# images and visual information.

if use_cuda and torch.cuda.is_available():
  AdvancedNet = AdvancedNet.cuda()

print("Test accuracy: {}".format(get_accuracy(AdvancedNet, test_f_loader)))
```

```
Test accuracy: 0.8993839835728953
```

## 5. Additional Testing [5 pt]

As a final step in testing we will be revisiting the sample images that you had collected and submitted at the start of this lab. These sample images should be untouched and will be used to demonstrate how well your model works at identifying your hand guestures.

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 [88]:
```
# The model performed quite well. I believe that the performance was slightly
# worse than the accuracy in Part 4(d), due to the fact that the testing
# dataset for Part 4(d)'s accuracy came from the same overall dataset as the
# training and validation set. This means that the images were doctored and
# edited similarly, and included hands of the same people.

# This test dataset, however, included completely different environments,
# lighting, and people. This would bring a lot more unobserved variance and
# information, attributing to the slightly worse test accuracy.

train_loader, val_loader, test_loader = get_data_loader(
    data_path='/content/gdrive/MyDrive/Colab Notebooks/Labs/lee-1004112177',
    batch_size=1)

save_features('/content/gdrive/MyDrive/Colab Notebooks/Labs/lee-1004112177', train_loader)
submitted_features = torchvision.datasets.DatasetFolder(
    '/content/gdrive/MyDrive/Colab Notebooks/Labs/lee-1004112177',
    loader=torch.load,
    extensions=('.tensor'))

test_f_loader = torch.utils.data.DataLoader(submitted_features,
                                            drop_last=True)

print("Test accuracy: {}".format(get_accuracy(AdvancedNet, test_f_loader)))
```

```
Test accuracy: 0.875
```