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://colab.research.google.com/drive/1KniRcZlIQPEo1fZUDnl_dvOv92_hMf6C?usp=sharing (https://colab.research.google.com/drive/1KniRcZlIQPEo1fZUDnl_dvOv92_hMf6C?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?

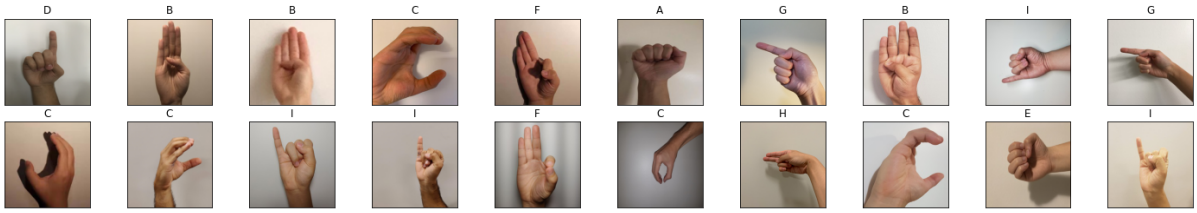
For loading the data, you can use plt.imread as in Lab 1, or any other method that you choose. You may find torchvision.datasets.ImageFolder helpful. (see https://pytorch.org/docs/stable/torchvision/datasets.html?highlight=image%20folder#torchvision.datasets.ImageFolder (https://pytorch.org/docs/stable/torchvision/datasets.html?highlight=image%20folder#torchvisi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

In [2]:
```
# Mount my Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:
```
import numpy as np
import time
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
from torch.utils.data.sampler import SubsetRandomSampler
import torchvision.transforms as transforms
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
from math import floor

use_GPU = True

# Loading Data
def data_loading(batch_size, num_workers):

    # Hand Gesture Image folder location on Google Drive
    hand_img_folder = '/content/drive/My Drive/APS360 Colab Notebooks/Labs/Lab_3b_Gesture_Dataset/'

    # Transform Images to Tensors of pixel size 224 x 224
    transform = transforms.Compose(
            [transforms.ToTensor(),
            transforms.Resize((224, 224))])

    # Three folders within hand_img_folder  # 60% Training, 20% Validation, and 20% Testing images
    # Training:1459 Validation:490 Testing:482 images
    train_set = torchvision.datasets.ImageFolder(hand_img_folder + 'Training', transform=transform)
    val_set = torchvision.datasets.ImageFolder(hand_img_folder + 'Validation', transform=transform)
    test_set = torchvision.datasets.ImageFolder(hand_img_folder + 'Testing', transform=transform)

    training_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, num_workers=num_workers, shuffle=True)
    validation_loader = torch.utils.data.DataLoader(val_set, batch_size=batch_size, num_workers=num_workers, shuffle=True)
    testing_loader = torch.utils.data.DataLoader(test_set, batch_size=batch_size, num_workers=num_workers, shuffle=True)

    return training_loader, validation_loader, testing_loader

# batch_size = 30, num_workers = 1
training_loader, validation_loader, testing_loader = data_loading(30, 1)

# Classes from A-I
classes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']

# obtain one batch of images from Training
dataiter = iter(training_loader)
images, labels = dataiter.next()
images = images.numpy() # convert images to numpy for display
```

```
# plot the images in the batch, along with the corresponding labels
fig = plt.figure(figsize=(25, 4))
for idx in np.arange(20):
    ax = fig.add_subplot(2, 20/2, idx+1, xticks=[], yticks=[])
    plt.imshow(np.transpose(images[idx], (1, 2, 0)))
    ax.set_title(classes[labels[idx]])
```



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 [4]:
```
# Convolutional Neural Network Architecture
class HandGestureClassifier(nn.Module):
    def __init__(self, change_kernel_sizes = [5, 5], name = "HandGesture
s"):
        super(HandGestureClassifier, self).__init__()
        self.name = name # Used for saving different Hyperparameter resu
lts
        self.conv1 = nn.Conv2d(3, 5, change_kernel_sizes[0]) #in_channel
s, out_chanels, kernel_size
        self.pool = nn.MaxPool2d(2, 2) #kernel_size, stride
        self.conv2 = nn.Conv2d(5, 10, change_kernel_sizes[1]) #in_channe
ls, out_chanels, kernel_size

        # Calculations for input size of first fully connected layer
        self.x1 = floor((224-change_kernel_sizes[0]+1)/2) #divide by 2 f
or pooling
        self.x2 = floor((self.x1-change_kernel_sizes[1]+1)/2) #floor for
.5 values

        self.fc1 = nn.Linear(10 * self.x2 * self.x2, 30) #See Calculatio
ns (if needed)
        self.fc2 = nn.Linear(30, 9)

    def forward(self, x):
        z1 = self.pool(F.relu(self.conv1(x))) #layer1CNN
        z2 = self.pool(F.relu(self.conv2(z1))) #layer2CNN
        z2 = z2.view(-1, 10 * self.x2 * self.x2)
        z3 = F.relu(self.fc1(z2)) #layer3NN
        z4 = self.fc2(z3) #layer4NN
        return z4

# Calculations:
# x: 3*224*224 --> conv1: 5*220*220 --> maxpool: 5*110*100 --> conv2: 10
*106*106 --> maxpool: 10*53*53


# 2 Convolutional layers, 2 Pooling layers, 2 Fully connected layers.'

# 1st Convolutional layer: 5 feature maps (5 output channels), 2nd Convo
lutional layer: 10 feature maps (10 output channels).
# I chose 2 Convolutional layers to capture the main 2 features, horizon
tal and vertical edges, and each with
# many feature maps to better the result.

# 2 Pooling layers were used to summarize the max and make each output f
rom Conv layers smaller,
# thus can creater a smaller input to the Fully connected layers.

# The relu activation function was used to add non-linearity and since n
o negative values required.
# Will use the softmax activation function on the final layer with nn.Cr
ossEntropyLoss() for classifying alphabet images.

# The number of hidden units in the fully connected layers are 10*53*53,
30, respectively,
```

```
# because 10*53*53 is outputed from the 2nd Convolutional layer, and 30
  as an arbitrary value.
```

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 [5]:
```
def get_model_name(name, batch_size, learning_rate, epoch):
    path = "model_{0}_bs{1}_lr{2}_epoch{3}".format(name, batch_size, lea
rning_rate, epoch)
    return path
```

In [6]:
```
def train(model, batch_size=150, learning_rate=0.001, num_epochs=10):
    ######################################################################
####
    # Train a classifier on Alphabets A-I
    classes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
    ######################################################################
####
    # Fixed PyTorch random seed for reproducible result
    torch.manual_seed(1000)
    ######################################################################
####
    # Obtain the PyTorch data loader objects to load batches of the data
sets
    training_loader, validation_loader, testing_loader = data_loading(ba
tch_size, 1)
    ######################################################################
####
    # Define the Loss function and optimizer
    # In this case we will use the CrossEntropyLoss
    # which takes percantage likley output.
    # Optimizer will be SGD with Momentum.
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    ######################################################################
####
    # Set up some numpy arrays to store the training/validation accuracy
    train_acc = np.zeros(num_epochs)
    val_acc = np.zeros(num_epochs)
    ######################################################################
####
    # Train the network
    print ("Started Training...")
    # Loop over the data iterator and sample a new batch of training dat
a
    # Get the output from the network, and optimize our loss function.
    start_time = time.time()
    for epoch in range(num_epochs):  # loop over the dataset multiple ti
mes
        print ("Training epoch #" + str(epoch + 1) + " ...")
        for images, labels in iter(training_loader):
            # Got the inputs
            # For GPU
            if torch.cuda.is_available() and use_GPU:
              images = images.cuda()
              labels = labels.cuda()

            # Zero the parameter gradients
            optimizer.zero_grad()                # Clean-up
            # Forward pass, backward pass, and optimize
            outputs = model(images)              # forward pass
            loss = criterion(outputs, labels)    # Compute Loss
            loss.backward()                      # backward pass
            optimizer.step()                     # Update each parameter

        # Accuracy
        train_acc[epoch] = get_accuracy(model, training_loader)
```

```
        val_acc[epoch] = get_accuracy(model, validation_loader)

        print(("Epoch {}: Train acc: {} |"+ "Validation acc: {}").format
(
                    epoch + 1,
                    train_acc[epoch],
                    val_acc[epoch]))

        # Save the current model (checkpoint) to a file
        model_path = get_model_name(model.name, batch_size, learning_rat
e, epoch)
        torch.save(model.state_dict(), model_path)

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

    epochs = np.arange(1, num_epochs + 1)

    return train_acc, val_acc, epochs


# Adam optimizer integrates momentum and makes improvement on SGD
# Model = HandGestureClassifier
```

In [7]:
```
def get_accuracy(model, data_loader):
    correct = 0
    total = 0
    for images, labels in data_loader:
        #For GPU
        if torch.cuda.is_available() and use_GPU:
          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
```

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.

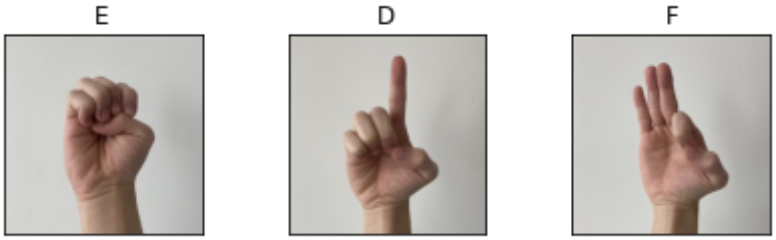
With a large batch size (e.g. the entire small dataset) and learning rate that is not too high, You should be able to obtain a 100% training accuracy on that small dataset relatively quickly (within 200 iterations).

In [11]:
```
# Train Model with small data sets (my own hands) and try to overfit
HandGestureClassifier_check_overfit = HandGestureClassifier()

# For GPU
if torch.cuda.is_available() and use_GPU:
  print("Using GPU")
  HandGestureClassifier_check_overfit.cuda()

train_acc = overfit_testing(HandGestureClassifier_check_overfit, overfit
_loader)
```

```
Using GPU
Started Training (to check Overfit)...
Epoch: 1, Training acc: 0.1111111111111111
Epoch: 2, Training acc: 0.2222222222222222
Epoch: 3, Training acc: 0.3333333333333333
Epoch: 4, Training acc: 0.6296296296296297
Epoch: 5, Training acc: 0.8518518518518519
Epoch: 6, Training acc: 0.8888888888888888
Epoch: 7, Training acc: 1.0
Epoch: 8, Training acc: 1.0
Epoch: 9, Training acc: 1.0
Epoch: 10, Training acc: 1.0
Epoch: 11, Training acc: 1.0
Epoch: 12, Training acc: 1.0
Epoch: 13, Training acc: 1.0
Epoch: 14, Training acc: 1.0
Epoch: 15, Training acc: 1.0
Epoch: 16, Training acc: 1.0
Epoch: 17, Training acc: 1.0
Epoch: 18, Training acc: 1.0
Epoch: 19, Training acc: 1.0
Epoch: 20, Training acc: 1.0
Ended Training (to check Overfit)...
```

In [9]:
```
def overfit_testing(model, loader, batch_size=27, learn_rate = 0.001, num_epochs=20):
    torch.manual_seed(1001)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learn_rate)

    train_acc = np.zeros(num_epochs)

    # training
    print ("Started Training (to check Overfit)...")
    for epoch in range(num_epochs):
        for images, labels in iter(loader):
            # For GPU
            if torch.cuda.is_available() and use_GPU:
              images = images.cuda()
              labels = labels.cuda()

            optimizer.zero_grad()               # Clean-up
            # Forward pass, backward pass, and optimize
            outputs = model(images)             # forward pass
            loss = criterion(outputs, labels)   # Compute Loss
            loss.backward()                     # backward pass
            optimizer.step()                    # Update each parameter

        # Accuracy
        train_acc[epoch] = get_accuracy(model, loader)
        print("Epoch: {}, Training acc: {}".format(epoch + 1, train_acc[epoch]))

    print ("Ended Training (to check Overfit)...")

    return train_acc
```

In [10]:
```
# Loading Data
def overfit_data_loading(batch_size, num_workers):

    # Hand Gesture Image folder location on Google Drive
    hand_img_folder = '/content/drive/My Drive/APS360 Colab Notebooks/Labs/Lab_3b_Gesture_Dataset/'

    # Transform Images to Tensors of pixel size 224 x 224
    transform = transforms.Compose(
            [transforms.ToTensor(),
            transforms.Resize((224, 224))])

    # One folder within hand_img_folder  # 100% Overfit images
    overfit_set = torchvision.datasets.ImageFolder(hand_img_folder + 'Overfitting (My hands)', transform=transform)

    overfit_loader = torch.utils.data.DataLoader(overfit_set, batch_size=batch_size, num_workers=num_workers, shuffle=True)

    return overfit_loader

# batch_size = 3, num_workers = 1
overfit_loader = overfit_data_loading(3, 1)

# Classes from A-I
classes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']

# obtain one batch of images from Training
dataiter = iter(overfit_loader)
images, labels = dataiter.next()
images = images.numpy() # convert images to numpy for display

# plot the images in the batch, along with the corresponding labels
fig = plt.figure(figsize=(25, 4))
for idx in np.arange(3):
    ax = fig.add_subplot(2, 20/2, idx+1, xticks=[], yticks=[])
    plt.imshow(np.transpose(images[idx], (1, 2, 0)))
    ax.set_title(classes[labels[idx]])
```



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 [ ]:
```
# Batch Size
# Changes the size of information the model error will learn from
# Affects how quickly the model can learn, and how accurate
# Too Small --> Less accurate but trains faster
# Too Large --> Can have generalization error and trains slower

# Number of Epoch
# Changes how many times the entire datasets are iterated
# Affects how quickly and accurately the model learns
# Too Small --> Model may not have learned the dataset properly
# Too Large --> Can take too much time

# Learning Rate
# Changes the gradient decent movement (optimizer),
# Affects how fast the model learn
# Too small --> can get stuck
# Too large --> can have suboptimal solution

# Kernel Size
# The size of the kernel (each filter) can affect how the raw data
# image is processed
```

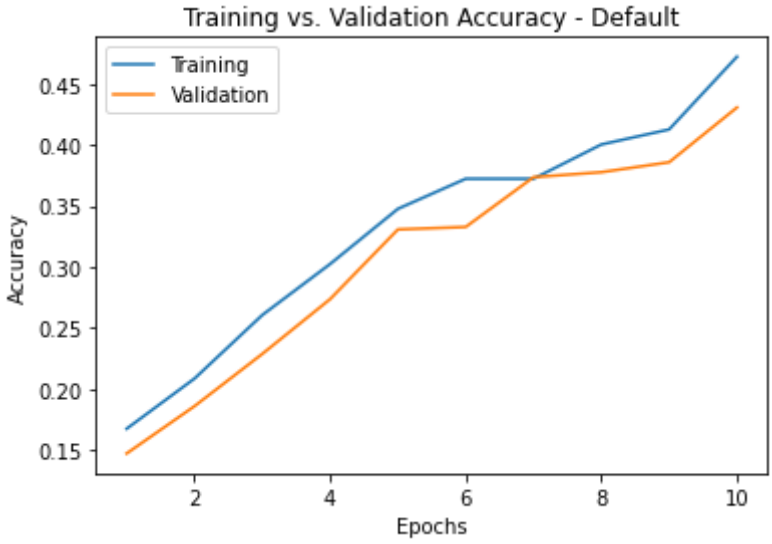
Part (b) - 5 pt

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]:
```
def plot_curves(epochs, train_acc, val_acc, name):
  plt.title("Training vs. Validation Accuracy - {}".format(name))
  plt.plot(epochs, train_acc, label="Training")
  plt.plot(epochs, val_acc, label="Validation")
  plt.xlabel("Epochs")
  plt.ylabel("Accuracy")
  plt.legend(loc='best')
  plt.show()

  #plt.title("Validation Curve - {}".format(name))
  #plt.plot(epochs, val_acc)
  #plt.xlabel("Epochs")
  #plt.ylabel("Validation Accuracy")
  #plt.show()
```

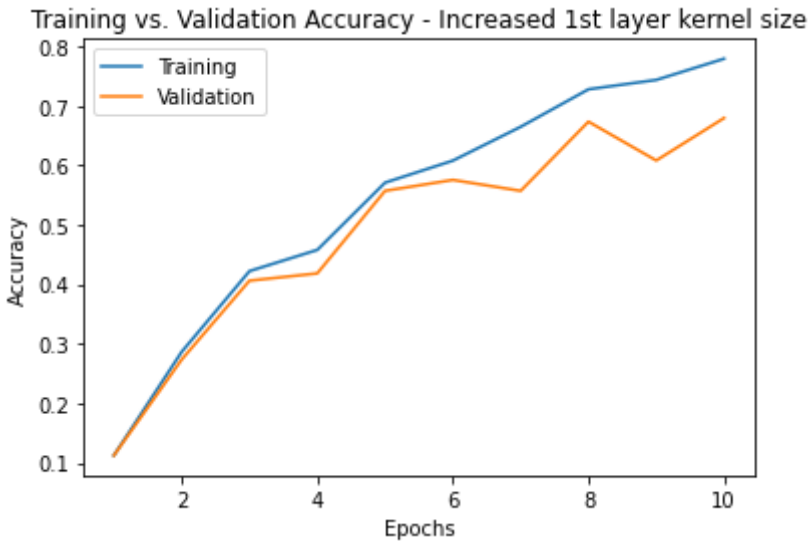
In [14]:
```
# Train Model with Hyperparameters:
# Learning Rate - 0.001
# Batch Size - 150
# Number of Epoch - 10
# Kernel Size - 1st Convolution layer: 5 & 2nd Convolution layer: 5

name = "Default"

HandGestureClassifier_P1 = HandGestureClassifier(change_kernel_sizes = [
5, 5], name = name)

# For GPU
if torch.cuda.is_available() and use_GPU:
  print("Using GPU")
  HandGestureClassifier_P1.cuda()

train_acc, val_acc, epochs = train(HandGestureClassifier_P1, batch_size=
150, learning_rate=0.001, num_epochs=10)

plot_curves(epochs, train_acc, val_acc, name)
```

```
Using GPU
Started Training...
Training epoch #1 ...
Epoch 1: Train acc: 0.16723783413296778 |Validation acc: 0.146938775510
2041
Training epoch #2 ...
Epoch 2: Train acc: 0.2083618917066484 |Validation acc: 0.1857142857142
8572
Training epoch #3 ...
Epoch 3: Train acc: 0.2604523646333105 |Validation acc: 0.2285714285714
2856
Training epoch #4 ...
Epoch 4: Train acc: 0.3022618231665524 |Validation acc: 0.2734693877551
0204
Training epoch #5 ...
Epoch 5: Train acc: 0.3474982864976011 |Validation acc: 0.3306122448979
592
Training epoch #6 ...
Epoch 6: Train acc: 0.3721727210418095 |Validation acc: 0.3326530612244
898
Training epoch #7 ...
Epoch 7: Train acc: 0.3721727210418095 |Validation acc: 0.3734693877551
02
Training epoch #8 ...
Epoch 8: Train acc: 0.4002741603838245 |Validation acc: 0.3775510204081
6324
Training epoch #9 ...
Epoch 9: Train acc: 0.4126113776559287 |Validation acc: 0.3857142857142
8573
Training epoch #10 ...
Epoch 10: Train acc: 0.4722412611377656 |Validation acc: 0.430612244897
9592
Finished Training...
Total time elapsed: 405.78 seconds
```



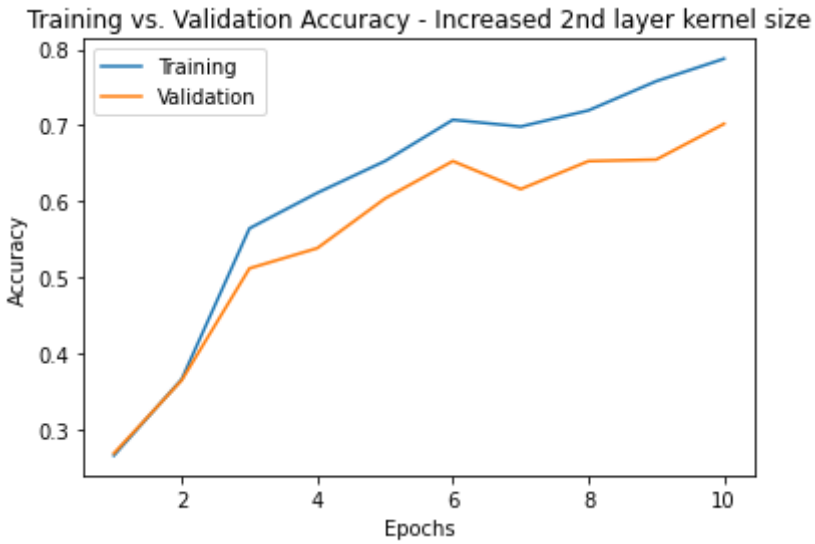
In [ ]:
```
# Here, I tried to increase the kernel size for the 1st layer

# Train Model with Hyperparameters:
# Learning Rate - 0.001
# Batch Size - 150
# Number of Epoch - 10
# Kernel Size - 1st Convolution layer: 10 & 2nd Convolution layer: 5

name = "Increased 1st layer kernel size"

HandGestureClassifier_P2 = HandGestureClassifier(change_kernel_sizes = [
10, 5], name = name)

# For GPU
if torch.cuda.is_available() and use_GPU:
  print("Using GPU")
  HandGestureClassifier_P2.cuda()

train_acc, val_acc, epochs = train(HandGestureClassifier_P2, batch_size=
150, learning_rate=0.001, num_epochs=10)

plot_curves(epochs, train_acc, val_acc, name)
```

```
Started Training...
Training epoch #1 ...
Epoch 1: Train acc: 0.11240575736806031 |Validation acc: 0.112244897959
18367
Training epoch #2 ...
Epoch 2: Train acc: 0.2864976010966415 |Validation acc: 0.2734693877551
0204
Training epoch #3 ...
Epoch 3: Train acc: 0.42220699108978754 |Validation acc: 0.406122448979
5918
Training epoch #4 ...
Epoch 4: Train acc: 0.4578478409869774 |Validation acc: 0.4183673469387
7553
Training epoch #5 ...
Epoch 5: Train acc: 0.570938999314599 |Validation acc: 0.55714285714285
72
Training epoch #6 ...
Epoch 6: Train acc: 0.6079506511309116 |Validation acc: 0.5755102040816
327
Training epoch #7 ...
Epoch 7: Train acc: 0.6648389307745031 |Validation acc: 0.5571428571428
572
Training epoch #8 ...
Epoch 8: Train acc: 0.7278958190541467 |Validation acc: 0.6734693877551
02
Training epoch #9 ...
Epoch 9: Train acc: 0.7436600411240576 |Validation acc: 0.6081632653061
224
Training epoch #10 ...
Epoch 10: Train acc: 0.7793008910212474 |Validation acc: 0.679591836734
6939
Finished Training...
Total time elapsed: 687.28 seconds
```



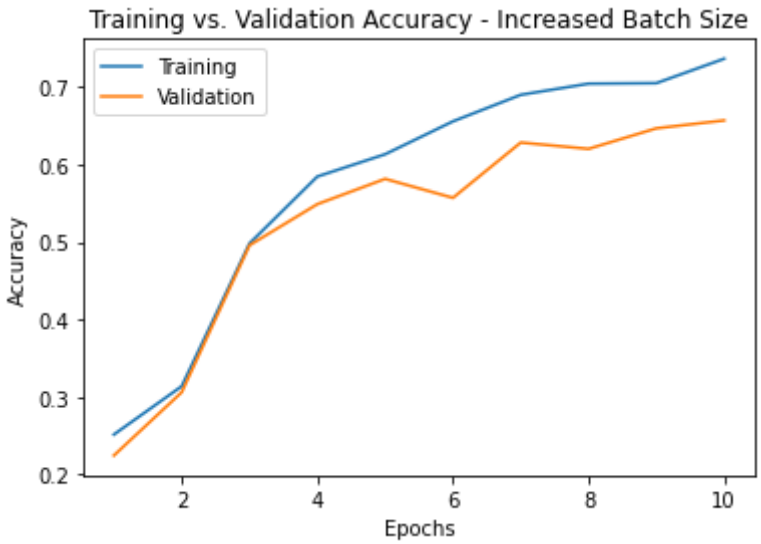
In [ ]:
```
# Here, I will try to increase the kernel size for the 2nd layer.

# Train Model with Hyperparameters:
# Learning Rate - 0.001
# Batch Size - 150
# Number of Epoch - 10
# Kernel Size - 1st Convolution layer: 5 & 2nd Convolution layer: 10

name = "Increased 2nd layer kernel size"

HandGestureClassifier_P3 = HandGestureClassifier(change_kernel_sizes = [
5, 10], name = name)

# For GPU
if torch.cuda.is_available() and use_GPU:
  print("Using GPU")
  HandGestureClassifier_P3.cuda()

train_acc, val_acc, epochs = train(HandGestureClassifier_P3, batch_size=
150, learning_rate=0.001, num_epochs=10)

plot_curves(epochs, train_acc, val_acc, name)
```

```
Started Training...
Training epoch #1 ...
Epoch 1: Train acc: 0.26662097326936257 |Validation acc: 0.269387755102
0408
Training epoch #2 ...
Epoch 2: Train acc: 0.3666895133653187 |Validation acc: 0.3653061224489
796
Training epoch #3 ...
Epoch 3: Train acc: 0.564770390678547 |Validation acc: 0.51224489795918
37
Training epoch #4 ...
Epoch 4: Train acc: 0.6113776559287183 |Validation acc: 0.5387755102040
817
Training epoch #5 ...
Epoch 5: Train acc: 0.6531871144619602 |Validation acc: 0.6040816326530
613
Training epoch #6 ...
Epoch 6: Train acc: 0.7073337902673064 |Validation acc: 0.6530612244897
959
Training epoch #7 ...
Epoch 7: Train acc: 0.6984235777930089 |Validation acc: 0.6163265306122
448
Training epoch #8 ...
Epoch 8: Train acc: 0.7196710075394106 |Validation acc: 0.6530612244897
959
Training epoch #9 ...
Epoch 9: Train acc: 0.7580534612748457 |Validation acc: 0.6551020408163
265
Training epoch #10 ...
Epoch 10: Train acc: 0.7875257025359835 |Validation acc: 0.702040816326
5306
Finished Training...
Total time elapsed: 778.43 seconds
```



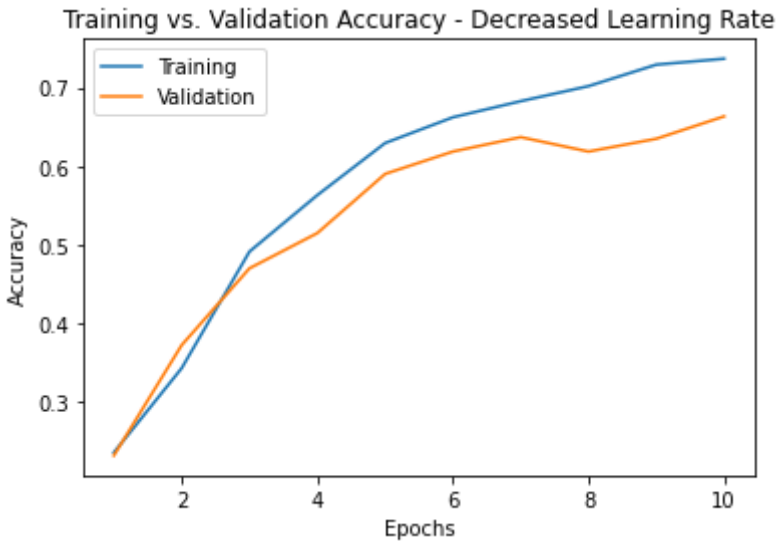
In [ ]:
```
# As it seems increasing the 2nd layer kernel size performed better,
# I will use the same kernel sizes [5, 10] and increase the batch size

# Train Model with Hyperparameters:
# Learning Rate - 0.001
# Batch Size - 250
# Number of Epoch - 10
# Kernel Size - 1st Convolution layer: 5 & 2nd Convolution layer: 10

name = "Increased Batch Size"

HandGestureClassifier_P4 = HandGestureClassifier(change_kernel_sizes = [
5, 10], name = name)

# For GPU
if torch.cuda.is_available() and use_GPU:
  print("Using GPU")
  HandGestureClassifier_P4.cuda()

train_acc, val_acc, epochs = train(HandGestureClassifier_P4, batch_size=
250, learning_rate=0.001, num_epochs=10)

plot_curves(epochs, train_acc, val_acc, name)
```

```
Started Training...
Training epoch #1 ...
Epoch 1: Train acc: 0.25154215215901304 |Validation acc: 0.224489795918
36735
Training epoch #2 ...
Epoch 2: Train acc: 0.3139136394790953 |Validation acc: 0.3061224489795
9184
Training epoch #3 ...
Epoch 3: Train acc: 0.49828649760109667 |Validation acc: 0.495918367346
9388
Training epoch #4 ...
Epoch 4: Train acc: 0.5846470185058259 |Validation acc: 0.5489795918367
347
Training epoch #5 ...
Epoch 5: Train acc: 0.6134338588074023 |Validation acc: 0.5816326530612
245
Training epoch #6 ...
Epoch 6: Train acc: 0.6559287183002056 |Validation acc: 0.5571428571428
572
Training epoch #7 ...
Epoch 7: Train acc: 0.6901987662782728 |Validation acc: 0.6285714285714
286
Training epoch #8 ...
Epoch 8: Train acc: 0.704592186429061 |Validation acc: 0.62040816326530
61
Training epoch #9 ...
Epoch 9: Train acc: 0.7052775873886223 |Validation acc: 0.6469387755102
041
Training epoch #10 ...
Epoch 10: Train acc: 0.7368060315284441 |Validation acc: 0.657142857142
8571
Finished Training...
Total time elapsed: 1050.14 seconds
```



In [ ]:
```
# As it seems increasing batch size made it worse,
# I will use the same kernel sizes [5, 10] and the previous batch size
# and now try to decrease the learning rate

# Train Model with Hyperparameters:
# Learning Rate - 0.0008
# Batch Size - 150
# Number of Epoch - 10
# Kernel Size - 1st Convolution layer: 5 & 2nd Convolution layer: 10

name = "Decreased Learning Rate"

HandGestureClassifier_P5 = HandGestureClassifier(change_kernel_sizes = [
5, 10], name = name)

# For GPU
if torch.cuda.is_available() and use_GPU:
  print("Using GPU")
  HandGestureClassifier_P5.cuda()

train_acc, val_acc, epochs = train(HandGestureClassifier_P5, batch_size=
150, learning_rate=0.0008, num_epochs=10)

plot_curves(epochs, train_acc, val_acc, name)
```

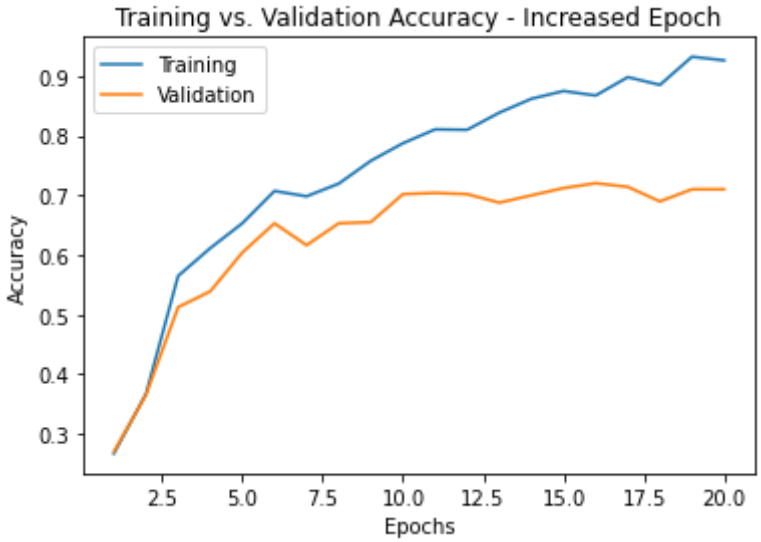
```
Started Training...
Training epoch #1 ...
Epoch 1: Train acc: 0.23440712816997944 |Validation acc: 0.230612244897
95917
Training epoch #2 ...
Epoch 2: Train acc: 0.3420150788211104 |Validation acc: 0.3714285714285
7144
Training epoch #3 ...
Epoch 3: Train acc: 0.49074708704592185 |Validation acc: 0.469387755102
04084
Training epoch #4 ...
Epoch 4: Train acc: 0.5627141877998629 |Validation acc: 0.5142857142857
142
Training epoch #5 ...
Epoch 5: Train acc: 0.6291980808773132 |Validation acc: 0.5897959183673
469
Training epoch #6 ...
Epoch 6: Train acc: 0.6620973269362577 |Validation acc: 0.6183673469387
755
Training epoch #7 ...
Epoch 7: Train acc: 0.682659355723098 |Validation acc: 0.63673469387755
1
Training epoch #8 ...
Epoch 8: Train acc: 0.7018505825908157 |Validation acc: 0.6183673469387
755
Training epoch #9 ...
Epoch 9: Train acc: 0.7292666209732693 |Validation acc: 0.6346938775510
204
Training epoch #10 ...
Epoch 10: Train acc: 0.7368060315284441 |Validation acc: 0.663265306122
4489
Finished Training...
Total time elapsed: 887.80 seconds
```



In [ ]:
```
# I will try to increase the number of epochs

# Train Model with Hyperparameters:
# Learning Rate - 0.001
# Batch Size - 150
# Number of Epoch - 20
# Kernel Size - 1st Convolution layer: 5 & 2nd Convolution layer: 10

name = "Increased Epoch"

HandGestureClassifier_P4 = HandGestureClassifier(change_kernel_sizes = [
5, 10], name = name)

# For GPU
if torch.cuda.is_available() and use_GPU:
  print("Using GPU")
  HandGestureClassifier_P4.cuda()

train_acc, val_acc, epochs = train(HandGestureClassifier_P4, batch_size=
150, learning_rate=0.001, num_epochs=20)

plot_curves(epochs, train_acc, val_acc, name)
```

```
Started Training...
Training epoch #1 ...
Epoch 1: Train acc: 0.26662097326936257 |Validation acc: 0.269387755102
0408
Training epoch #2 ...
Epoch 2: Train acc: 0.3666895133653187 |Validation acc: 0.3653061224489
796
Training epoch #3 ...
Epoch 3: Train acc: 0.564770390678547 |Validation acc: 0.51224489795918
37
Training epoch #4 ...
Epoch 4: Train acc: 0.6113776559287183 |Validation acc: 0.5387755102040
817
Training epoch #5 ...
Epoch 5: Train acc: 0.6531871144619602 |Validation acc: 0.6040816326530
613
Training epoch #6 ...
Epoch 6: Train acc: 0.7073337902673064 |Validation acc: 0.6530612244897
959
Training epoch #7 ...
Epoch 7: Train acc: 0.6984235777930089 |Validation acc: 0.6163265306122
448
Training epoch #8 ...
Epoch 8: Train acc: 0.7196710075394106 |Validation acc: 0.6530612244897
959
Training epoch #9 ...
Epoch 9: Train acc: 0.7580534612748457 |Validation acc: 0.6551020408163
265
Training epoch #10 ...
Epoch 10: Train acc: 0.7875257025359835 |Validation acc: 0.702040816326
5306
Training epoch #11 ...
Epoch 11: Train acc: 0.8108293351610693 |Validation acc: 0.704081632653
0612
Training epoch #12 ...
Epoch 12: Train acc: 0.8101439342015079 |Validation acc: 0.702040816326
5306
Training epoch #13 ...
Epoch 13: Train acc: 0.8389307745030843 |Validation acc: 0.687755102040
8164
Training epoch #14 ...
Epoch 14: Train acc: 0.86223440712817 |Validation acc: 0.7
Training epoch #15 ...
Epoch 15: Train acc: 0.8752570253598355 |Validation acc: 0.712244897959
1837
Training epoch #16 ...
Epoch 16: Train acc: 0.8677176148046607 |Validation acc: 0.720408163265
3061
Training epoch #17 ...
Epoch 17: Train acc: 0.8985606579849211 |Validation acc: 0.714285714285
7143
Training epoch #18 ...
Epoch 18: Train acc: 0.8855380397532556 |Validation acc: 0.689795918367
347
Training epoch #19 ...
Epoch 19: Train acc: 0.9328307059629883 |Validation acc: 0.710204081632
653
```

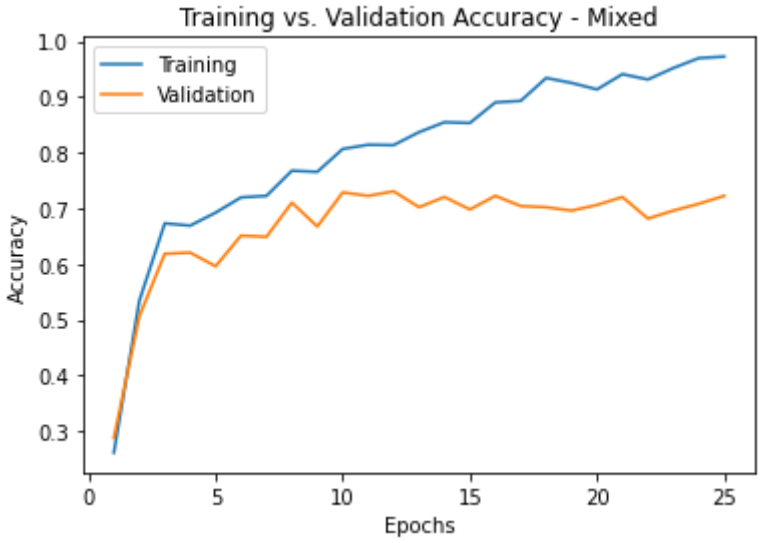
```
Training epoch #20 ...
Epoch 20: Train acc: 0.9266620973269363 |Validation acc: 0.710204081632
653
Finished Training...
Total time elapsed: 1797.42 seconds
```



In [15]:
```
# Since it still seems to overfit,
# I will try to Mix the results

# Train Model with Hyperparameters:
# Learning Rate - 0.0012
# Batch Size - 150
# Number of Epoch - 25
# Kernel Size - 1st Convolution layer: 5 & 2nd Convolution layer: 10

name = "Mixed"

HandGestureClassifier_P7 = HandGestureClassifier(change_kernel_sizes = [
5, 10], name = name)

# For GPU
if torch.cuda.is_available() and use_GPU:
  print("Using GPU")
  HandGestureClassifier_P7.cuda()

train_acc, val_acc, epochs = train(HandGestureClassifier_P7, batch_size=
150, learning_rate=0.0012, num_epochs=25)

plot_curves(epochs, train_acc, val_acc, name)
```

```
Using GPU
Started Training...
Training epoch #1 ...
Epoch 1: Train acc: 0.26113776559287183 |Validation acc: 0.287755102040
81634
Training epoch #2 ...
Epoch 2: Train acc: 0.5339273474982865 |Validation acc: 0.5061224489795
918
Training epoch #3 ...
Epoch 3: Train acc: 0.6730637422892392 |Validation acc: 0.6183673469387
755
Training epoch #4 ...
Epoch 4: Train acc: 0.6689513365318711 |Validation acc: 0.6204081632653
061
Training epoch #5 ...
Epoch 5: Train acc: 0.6922549691569568 |Validation acc: 0.5959183673469
388
Training epoch #6 ...
Epoch 6: Train acc: 0.7196710075394106 |Validation acc: 0.6510204081632
653
Training epoch #7 ...
Epoch 7: Train acc: 0.7224126113776559 |Validation acc: 0.6489795918367
347
Training epoch #8 ...
Epoch 8: Train acc: 0.7676490747087046 |Validation acc: 0.7102040816326
53
Training epoch #9 ...
Epoch 9: Train acc: 0.7655928718300206 |Validation acc: 0.6673469387755
102
Training epoch #10 ...
Epoch 10: Train acc: 0.8067169294037012 |Validation acc: 0.728571428571
4285
Training epoch #11 ...
Epoch 11: Train acc: 0.8142563399588759 |Validation acc: 0.722448979591
8367
Training epoch #12 ...
Epoch 12: Train acc: 0.8135709389993147 |Validation acc: 0.730612244897
9592
Training epoch #13 ...
Epoch 13: Train acc: 0.8368745716244003 |Validation acc: 0.702040816326
5306
Training epoch #14 ...
Epoch 14: Train acc: 0.8546949965729952 |Validation acc: 0.720408163265
3061
Training epoch #15 ...
Epoch 15: Train acc: 0.8533241946538725 |Validation acc: 0.697959183673
4694
Training epoch #16 ...
Epoch 16: Train acc: 0.890335846470185 |Validation acc: 0.7224489795918
367
Training epoch #17 ...
Epoch 17: Train acc: 0.8930774503084304 |Validation acc: 0.704081632653
0612
Training epoch #18 ...
Epoch 18: Train acc: 0.934201507882111 |Validation acc: 0.7020408163265
306
Training epoch #19 ...
```

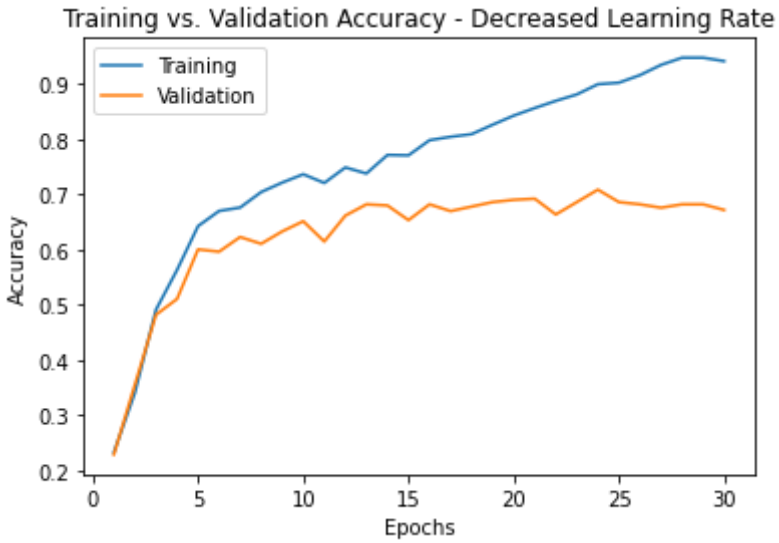
```
Epoch 19: Train acc: 0.9252912954078135 |Validation acc: 0.695918367346
9388
Training epoch #20 ...
Epoch 20: Train acc: 0.9136394790952708 |Validation acc: 0.706122448979
5919
Training epoch #21 ...
Epoch 21: Train acc: 0.9410555174777244 |Validation acc: 0.720408163265
3061
Training epoch #22 ...
Epoch 22: Train acc: 0.9314599040438657 |Validation acc: 0.681632653061
2244
Training epoch #23 ...
Epoch 23: Train acc: 0.9520219328307059 |Validation acc: 0.695918367346
9388
Training epoch #24 ...
Epoch 24: Train acc: 0.969842357779301 |Validation acc: 0.7081632653061
225
Training epoch #25 ...
Epoch 25: Train acc: 0.9725839616175462 |Validation acc: 0.722448979591
8367
Finished Training...
Total time elapsed: 304.38 seconds
```



In [22]:
```
# The training accuracy was the highest, but it still seems to overfit,
# I will try another Mix of the results

# Train Model with Hyperparameters:
# Learning Rate - 0.0008
# Batch Size - 150
# Number of Epoch - 10
# Kernel Size - 1st Convolution layer: 5 & 2nd Convolution layer: 10

name = "Decreased Learning Rate"

HandGestureClassifier_P8 = HandGestureClassifier(change_kernel_sizes = [
5, 10], name = name)

# For GPU
if torch.cuda.is_available() and use_GPU:
  print("Using GPU")
  HandGestureClassifier_P8.cuda()

train_acc, val_acc, epochs = train(HandGestureClassifier_P8, batch_size=
150, learning_rate=0.0008, num_epochs=30)

plot_curves(epochs, train_acc, val_acc, name)
```

```
Using GPU
Started Training...
Training epoch #1 ...
Epoch 1: Train acc: 0.23235092529129542 |Validation acc: 0.228571428571
42856
Training epoch #2 ...
Epoch 2: Train acc: 0.3406442769019877 |Validation acc: 0.3551020408163
265
Training epoch #3 ...
Epoch 3: Train acc: 0.49074708704592185 |Validation acc: 0.481632653061
2245
Training epoch #4 ...
Epoch 4: Train acc: 0.5627141877998629 |Validation acc: 0.5102040816326
531
Training epoch #5 ...
Epoch 5: Train acc: 0.6422206991089787 |Validation acc: 0.6
Training epoch #6 ...
Epoch 6: Train acc: 0.6696367374914325 |Validation acc: 0.5959183673469
388
Training epoch #7 ...
Epoch 7: Train acc: 0.6758053461274846 |Validation acc: 0.6224489795918
368
Training epoch #8 ...
Epoch 8: Train acc: 0.7039067854694997 |Validation acc: 0.6102040816326
53
Training epoch #9 ...
Epoch 9: Train acc: 0.7210418094585332 |Validation acc: 0.6326530612244
898
Training epoch #10 ...
Epoch 10: Train acc: 0.7361206305688828 |Validation acc: 0.651020408163
2653
Training epoch #11 ...
Epoch 11: Train acc: 0.7203564084989719 |Validation acc: 0.614285714285
7143
Training epoch #12 ...
Epoch 12: Train acc: 0.748457847840987 |Validation acc: 0.6612244897959
184
Training epoch #13 ...
Epoch 13: Train acc: 0.7374914324880055 |Validation acc: 0.681632653061
2244
Training epoch #14 ...
Epoch 14: Train acc: 0.7710760795065114 |Validation acc: 0.679591836734
6939
Training epoch #15 ...
Epoch 15: Train acc: 0.77039067854695 |Validation acc: 0.65306122448979
59
Training epoch #16 ...
Epoch 16: Train acc: 0.7978067169294037 |Validation acc: 0.681632653061
2244
Training epoch #17 ...
Epoch 17: Train acc: 0.8039753255654558 |Validation acc: 0.669387755102
0408
Training epoch #18 ...
Epoch 18: Train acc: 0.8087731322823852 |Validation acc: 0.677551020408
1633
Training epoch #19 ...
Epoch 19: Train acc: 0.8259081562714188 |Validation acc: 0.685714285714
```

```
2857
Training epoch #20 ...
Epoch 20: Train acc: 0.8423577793008911 |Validation acc: 0.689795918367
347
Training epoch #21 ...
Epoch 21: Train acc: 0.8560657984921178 |Validation acc: 0.691836734693
8775
Training epoch #22 ...
Epoch 22: Train acc: 0.8690884167237835 |Validation acc: 0.663265306122
4489
Training epoch #23 ...
Epoch 23: Train acc: 0.8807402330363262 |Validation acc: 0.685714285714
2857
Training epoch #24 ...
Epoch 24: Train acc: 0.8992460589444825 |Validation acc: 0.708163265306
1225
Training epoch #25 ...
Epoch 25: Train acc: 0.9019876627827279 |Validation acc: 0.685714285714
2857
Training epoch #26 ...
Epoch 26: Train acc: 0.9156956819739548 |Validation acc: 0.681632653061
2244
Training epoch #27 ...
Epoch 27: Train acc: 0.934201507882111 |Validation acc: 0.6755102040816
326
Training epoch #28 ...
Epoch 28: Train acc: 0.9472241261137766 |Validation acc: 0.681632653061
2244
Training epoch #29 ...
Epoch 29: Train acc: 0.9472241261137766 |Validation acc: 0.681632653061
2244
Training epoch #30 ...
Epoch 30: Train acc: 0.9410555174777244 |Validation acc: 0.671428571428
5714
Finished Training...
Total time elapsed: 367.93 seconds
```



Part (c) - 2 pt

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

In [ ]:
```
# The best model from the 7 above is the model HandGestureClassifier_P7.
# This is kind of obvious because I made this model by tweaking from the
# best (experimental) results from previous models.

# This model is the best because it has
# the highest training accuracy
# and the highest validation accuracy.
# Although it seems to have overfitting,
# this model had the least overfitting
```

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 [25]:
```
get_accuracy(HandGestureClassifier_P7, testing_loader)
```

Out[25]:
```
0.7282157676348547
```

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 [ ]:
```
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):

In [ ]:
```
# img = ... a PyTorch tensor with shape [N,3,224,224] containing hand im
ages ...
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 [71]:
```
import torchvision.models
alexnet = torchvision.models.alexnet(pretrained=True)

classes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']

# Loading Data
def data_loading_alex(batch_size, num_workers):
  # batch_size = 1, num_workers = 1
  training_loader, validation_loader, testing_loader = data_loading(batc
h_size, 1)

  # Hand Gesture Image folder location on Google Drive
  hand_img_folder = '/content/drive/My Drive/APS360 Colab Notebooks/Lab
s/Lab_3b_Gesture_Dataset/'

  i=0
  for images, labels in iter(training_loader):
    features = alexnet.features(images)
    features_tensor = (torch.from_numpy(features.detach().numpy())).sque
eze(0)

    torch.save(features_tensor, hand_img_folder + "Training-Alex/" + str
(classes[labels]) + "/" + str(i) + ".tensor")
    i+=1

  i=0
  for images, labels in iter(validation_loader):
    features = alexnet.features(images)
    features_tensor = (torch.from_numpy(features.detach().numpy())).sque
eze(0)

    torch.save(features_tensor, hand_img_folder + "Validation-Alex/" + s
tr(classes[labels]) + "/" + str(i) + ".tensor")
    i+=1

  i=0
  for images, labels in iter(testing_loader):
    features = alexnet.features(images)
    features_tensor = (torch.from_numpy(features.detach().numpy())).sque
eze(0)

    torch.save(features_tensor, hand_img_folder + "Testing-Alex/" + str(
classes[labels]) + "/" + str(i) + ".tensor")
    i+=1
```

In [32]:
```
data_loading_alex(1, 1)
```

In [72]:
```
hand_img_folder = '/content/drive/My Drive/APS360 Colab Notebooks/Labs/Lab_3b_Gesture_Dataset/'


train_feature_set = torchvision.datasets.DatasetFolder(hand_img_folder + 'Training-Alex', loader=torch.load, extensions=('.tensor'))
val_feature_set = torchvision.datasets.DatasetFolder(hand_img_folder + 'Validation-Alex', loader=torch.load, extensions=('.tensor'))
test_feature_set = torchvision.datasets.DatasetFolder(hand_img_folder + 'Testing-Alex', loader=torch.load, extensions=('.tensor'))

# batch_size = 30, num_workers = 1
training_feature_loader = torch.utils.data.DataLoader(train_feature_set, batch_size=32, num_workers=1, shuffle=True)
validation_feature_loader = torch.utils.data.DataLoader(val_feature_set, batch_size=32, num_workers=1, shuffle=True)
testing_feature_loader = torch.utils.data.DataLoader(test_feature_set, batch_size=32, num_workers=1, shuffle=True)
```

In [73]:
```
# obtain one batch of data from Training_feature for verification
dataiter = iter(training_feature_loader)
features, labels = dataiter.next()
print(features.shape)
print(labels.shape)
```

```
torch.Size([32, 256, 6, 6])
torch.Size([32])
```

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

In [83]:
```
class HandGestureClassifier_AlexNet(nn.Module):
    def __init__(self, name = "HandGestures"):
        super(HandGestureClassifier_AlexNet, self).__init__()
        self.name = name # Used for saving different Hyperparameter resu
lts
        self.conv1 = nn.Conv2d(256, 128, 3)
        self.pool = nn.MaxPool2d(2, 2)

        # Calculations for input size of first fully connected layer
        # input=6*6, Kernel Size=3*3
        self.x1 = floor((6-3+1)/2)

        self.fc1 = nn.Linear(128 * self.x1 * self.x1, 32)
        self.fc2 = nn.Linear(32, 9)

    def forward(self, features):
        z1 = self.pool(F.relu(self.conv1(features))) #layer1CNN
        z1 = z1.view(-1, 128 * self.x1 * self.x1)
        z2 = F.relu(self.fc1(z1)) #layer2NN
        z3 = self.fc2(z2) #layer3NN
        prob = F.softmax(z3, dim=1)
        return prob
```

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.

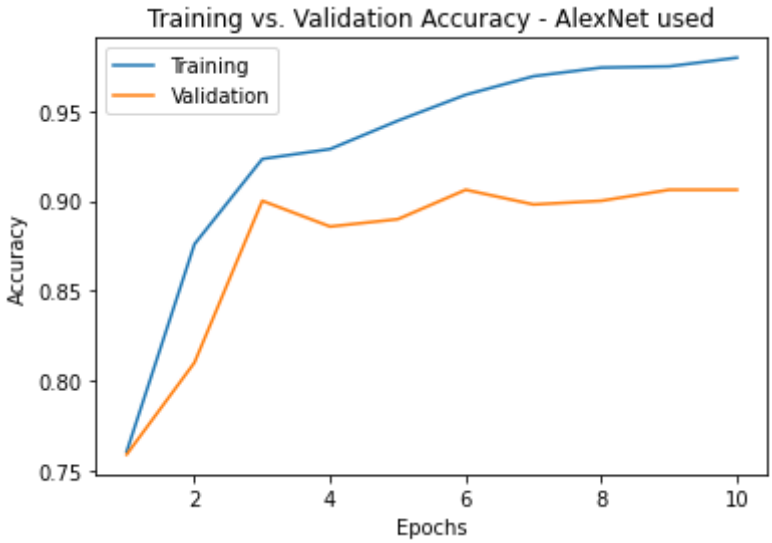
In [ ]:
```
tensor = torch.from_numpy(tensor.detach().numpy())

# This was already used when saving the files to directory!
```

In [81]:
```
def train_Alex(model, batch_size=32, learning_rate=0.001, num_epochs=10
):
    ###################################################################
####
    # Train a classifier on Alphabets A-I
    classes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
    ###################################################################
####
    # Fixed PyTorch random seed for reproducible result
    torch.manual_seed(1002)
    ###################################################################
####
    # Obtain the AlexNet data loader objects to load batches of the data
sets
    train_feature_set = torchvision.datasets.DatasetFolder(hand_img_fold
er + 'Training-Alex', loader=torch.load, extensions=('.tensor'))
    val_feature_set = torchvision.datasets.DatasetFolder(hand_img_folder
+ 'Validation-Alex', loader=torch.load, extensions=('.tensor'))

    # batch_size = 32, num_workers = 1
    training_feature_loader = torch.utils.data.DataLoader(train_feature_
set, batch_size=32, num_workers=1, shuffle=True)
    validation_feature_loader = torch.utils.data.DataLoader(val_feature_
set, batch_size=32, num_workers=1, shuffle=True)
    ###################################################################
####
    # Define the Loss function and optimizer
    # In this case we will use the CrossEntropyLoss
    # which takes percantage likley output.
    # Optimizer will be SGD with Momentum.
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    ###################################################################
####
    # Set up some numpy arrays to store the training/validation accuracy
    train_acc = np.zeros(num_epochs)
    val_acc = np.zeros(num_epochs)
    ###################################################################
####
    # Train the network
    print ("Started Training...")
    # Loop over the data iterator and sample a new batch of training dat
a
    # Get the output from the network, and optimize our loss function.
    start_time = time.time()
    for epoch in range(num_epochs):  # loop over the dataset multiple ti
mes
        print ("Training epoch #" + str(epoch + 1) + " ...")
        for images, labels in iter(training_feature_loader):
            # Got the inputs
            # For GPU
            if torch.cuda.is_available() and use_GPU:
              images = images.cuda()
              labels = labels.cuda()

            # Zero the parameter gradients
```

```
            optimizer.zero_grad()               # Clean-up
            # Forward pass, backward pass, and optimize
            outputs = model(images)             # forward pass
            loss = criterion(outputs, labels)   # Compute Loss
            loss.backward()                     # backward pass
            optimizer.step()                    # Update each parameter

        # Accuracy
        train_acc[epoch] = get_accuracy(model, training_feature_loader)
        val_acc[epoch] = get_accuracy(model, validation_feature_loader)

        print(("Epoch {}: Train acc: {} |"+ "Validation acc: {}").format
(
                    epoch + 1,
                    train_acc[epoch],
                    val_acc[epoch]))

        # Save the current model (checkpoint) to a file
        model_path = get_model_name(model.name, batch_size, learning_rat
e, epoch)
        torch.save(model.state_dict(), model_path)

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

    epochs = np.arange(1, num_epochs + 1)

    return train_acc, val_acc, epochs
```

In [84]:
```
name = "AlexNet used"

HandGestureClassifier_Alex = HandGestureClassifier_AlexNet()

if torch.cuda.is_available() and use_GPU:
  HandGestureClassifier_Alex.cuda()

train_acc, val_acc, epochs = train_Alex(HandGestureClassifier_Alex, 32)

plot_curves(epochs, train_acc, val_acc, name)
```

```
Started Training...
Training epoch #1 ...
Epoch 1: Train acc: 0.7607950651130911 |Validation acc: 0.7591836734693
878
Training epoch #2 ...
Epoch 2: Train acc: 0.8759424263193969 |Validation acc: 0.8102040816326
53
Training epoch #3 ...
Epoch 3: Train acc: 0.9232350925291295 |Validation acc: 0.9
Training epoch #4 ...
Epoch 4: Train acc: 0.9287183002056203 |Validation acc: 0.8857142857142
857
Training epoch #5 ...
Epoch 5: Train acc: 0.9444825222755312 |Validation acc: 0.8897959183673
47
Training epoch #6 ...
Epoch 6: Train acc: 0.9588759424263193 |Validation acc: 0.9061224489795
918
Training epoch #7 ...
Epoch 7: Train acc: 0.9691569568197396 |Validation acc: 0.8979591836734
694
Training epoch #8 ...
Epoch 8: Train acc: 0.973954763536669 |Validation acc: 0.9
Training epoch #9 ...
Epoch 9: Train acc: 0.9746401644962303 |Validation acc: 0.9061224489795
918
Training epoch #10 ...
Epoch 10: Train acc: 0.9794379712131597 |Validation acc: 0.906122448979
5918
Finished Training...
Total time elapsed: 41.05 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 [ ]:
```
# The test accuracy of the best model with using transfer learning was m
uch better
# than from the model without transfer learning in Part3(d).
# 0.929 > 0.728
# Also, it is obvious that there is much less overfitting than before
```

In [85]:
```
get_accuracy(HandGestureClassifier_Alex, testing_feature_loader)
```

Out[85]:
```
0.9294605809128631
```

In [86]:
```
# From Part3(d)
get_accuracy(HandGestureClassifier_P7, testing_loader)
```

Out[86]:
```
0.7282157676348547
```

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.

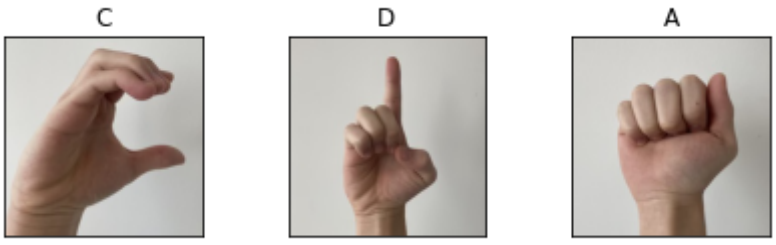
Using the best transfer learning model developed in Part 4. Report the test accuracy on your sample images and how it compares to the test accuracy obtained in Part 4(d)? How well did your model do for the different hand guestures? Provide an explanation for why you think your model performed the way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

In [ ]:
```
# The test accuracy on the testing sample images (my own hands)
# was very high to be 0.9788,
# which is higher than the test accuracy from part4(d).

# The model using transfer learning (model from part4)
# did very well on different hand gestures,
# as seen from the test accuracy of both the large set of given testing
 images
# and also the small set of given (my own hand) testing images

# I think the model performed better on my own hands because
# there is less derivation from the given images of my own hands
# (because it is only my hand)
# compared to the given images of many students that has higher derivati
on
# i.e. my hands my have less error than the collection of many students
# because I can check that there are no "wrong" hand gestures given by m
e

# Also, there are only 27 images of my hand
# compared to 482 images (20% from all images) to test on.
# Thus, it is easier to make less error.
```

In [121]:
```
get_accuracy(HandGestureClassifier_Alex, testingAlex_set_loader)
```

Out[121]:
```
0.9787525702535984
```

In [111]:
```
# Loading Data
def testingAlex_data_loading(batch_size, num_workers):

    # Hand Gesture Image folder location on Google Drive
    hand_img_folder = '/content/drive/My Drive/APS360 Colab Notebooks/Labs/Lab_3b_Gesture_Dataset/'

    # Transform Images to Tensors of pixel size 224 x 224
    transform = transforms.Compose(
            [transforms.ToTensor(),
            transforms.Resize((224, 224))])

    # One folder within hand_img_folder  # 100% Overfit images
    testingAlex_set = torchvision.datasets.ImageFolder(hand_img_folder + 'Overfitting (My hands)', transform=transform)

    testingAlex_loader = torch.utils.data.DataLoader(testingAlex_set, batch_size=batch_size, num_workers=num_workers, shuffle=True)

    return testingAlex_loader

# batch_size = 3, num_workers = 1
testingAlex_loader = testingAlex_data_loading(3, 1)

# Classes from A-I
classes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']

# obtain one batch of images from Training
dataiter = iter(testingAlex_loader)
images, labels = dataiter.next()
images = images.numpy() # convert images to numpy for display

# plot the images in the batch, along with the corresponding labels
fig = plt.figure(figsize=(25, 4))
for idx in np.arange(3):
    ax = fig.add_subplot(2, 20/2, idx+1, xticks=[], yticks=[])
    plt.imshow(np.transpose(images[idx], (1, 2, 0)))
    ax.set_title(classes[labels[idx]])
```



In [116]:
```
import torchvision.models
alexnet = torchvision.models.alexnet(pretrained=True)

classes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']

# Loading Data
def my_data_loading_alex(batch_size, num_workers):
  # batch_size = 1, num_workers = 1
  testingAlex_loader = testingAlex_data_loading(batch_size, 1)

  # Hand Gesture Image folder location on Google Drive
  hand_img_folder = '/content/drive/My Drive/APS360 Colab Notebooks/Lab
s/Lab_3b_Gesture_Dataset/'

  i=0
  for images, labels in iter(testingAlex_loader):
    features = alexnet.features(images)
    features_tensor = (torch.from_numpy(features.detach().numpy())).sque
eze(0)

    torch.save(features_tensor, hand_img_folder + "My-Alex/" + str(class
es[labels]) + "/" + str(i) + ".tensor")
    i+=1
```

In [117]:
```
my_data_loading_alex(1, 1)
```

In [119]:
```
hand_img_folder = '/content/drive/My Drive/APS360 Colab Notebooks/Labs/L
ab_3b_Gesture_Dataset/'

testingAlex_set = torchvision.datasets.DatasetFolder(hand_img_folder +
'Training-Alex', loader=torch.load, extensions=('.tensor'))

# batch_size = 30, num_workers = 1
testingAlex_set_loader = torch.utils.data.DataLoader(train_feature_set,
batch_size=32, num_workers=1, shuffle=True)
```