National University of Singapore

EE4705

Human & Robot Interaction

Project 1: Detection of Human Gestures and Behaviors Using Pytorch

Tran Duy Anh - A0200724W

Vigneshwar Venkatasubramanian - A0206234R

Phone Thit Hlaing - A0200121M

Contribution of each teammate

Student A: Phone Thit Hlaing

Student B: Tran Duy Anh

Student C: Vigneshwar Venkatasubramanian

Task 1

1. Gesture Recognition

Gesture recognition is a mathematical interpretation of human motion using machine learning. Gesture recognition includes different aspects such as facial, voice, eye tracking, lip movements and body movements. [1]

Gesture recognition also allows humans to collaborate with robots, working alongside them rather than robots replacing human workers. It provides an interface for humans to interact with robots by establishing effective communication channels between man and robots. As such, robots can relieve humans from carrying out heavy tasks and provide assistance. [2]

2. Fully Connected Neural Network

In this task we will use a fully connected neural network in order to classify the different gestures: left, palm, peace and right.

A fully connected neural network consists of a series of fully connected layers that connect every neuron in one layer to every neuron in the other layer as shown below:

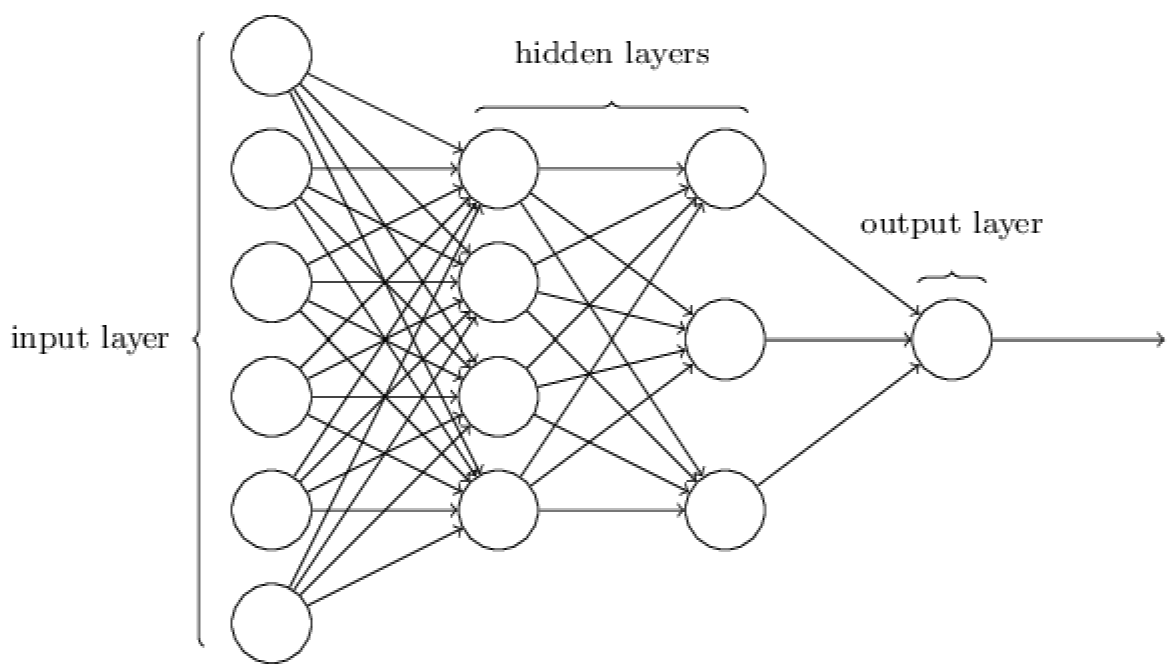

Figure 1: Fully-connected neural network

First, we read the images in the given dataset and process these images to ensure the images are more suited to be trained. The images are resized, colour corrected and indexed to ease the processing.

The images are also stretched into one dimension for the fully connected model to operate. In this task, we decided to use 3 fully connected layers in total, an input layer, a hidden layer and an output layer for the image classification.

3. Algorithm performance

Below is the result for this model:

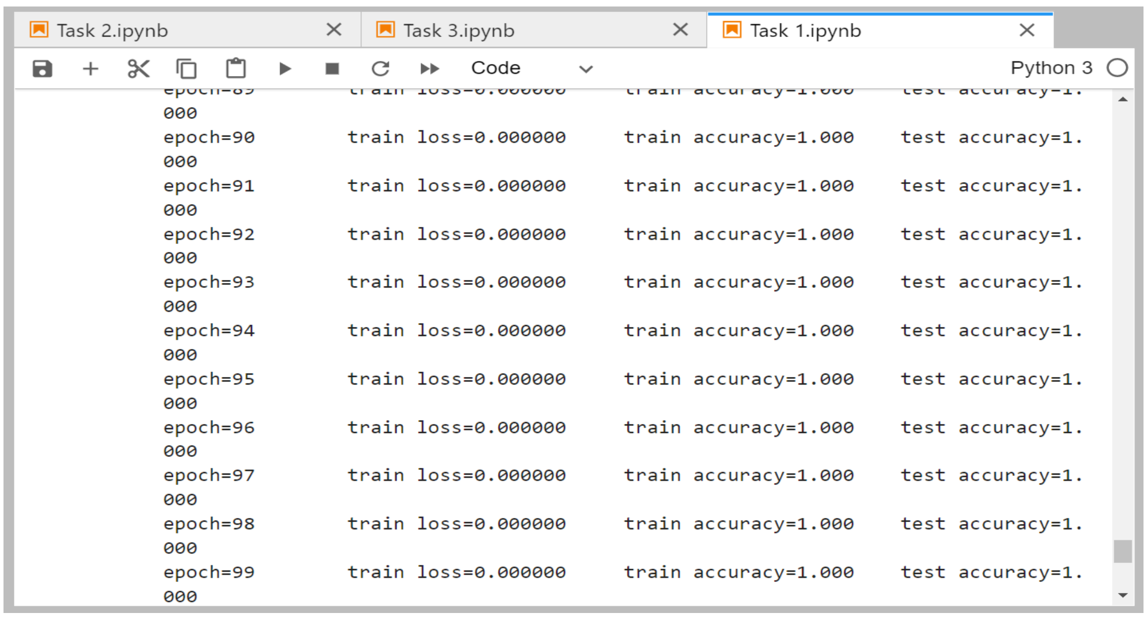

Figure 2: Performance of fully-connected neural network

After a few epochs, the train and test accuracy has already reached 100%. From then on, these two numbers are kept at 100%. This can be because of the lack of variety on the dataset. All the images in the dataset belong to one person in the same place.

Task 2

1. Convolutional Neural Network (CNN)

CNN is a type of neural network that can specifically deal with pixel data[3]. Thus, CNN is superior in image, speech, audio signal input data[4]. In a CNN, input image has to go through various filter layers so that its features can be learned. There are three main types of layers in a typical CNN.

The first type of layer is the Convolutional layer, which is in the heart of CNN. It requires the input to be an image in grayscale or coloured, a filter and a feature map. A filter in this layer is a 2D matrix that travels around the input image, performs dot product to an area of the input image, and saves the result into an output array[4]. Afterwards, the filter shifts by a number of pixels and continues doing the same operation. Such a filter is illustrated in the image below. The output array has another name, called feature map.

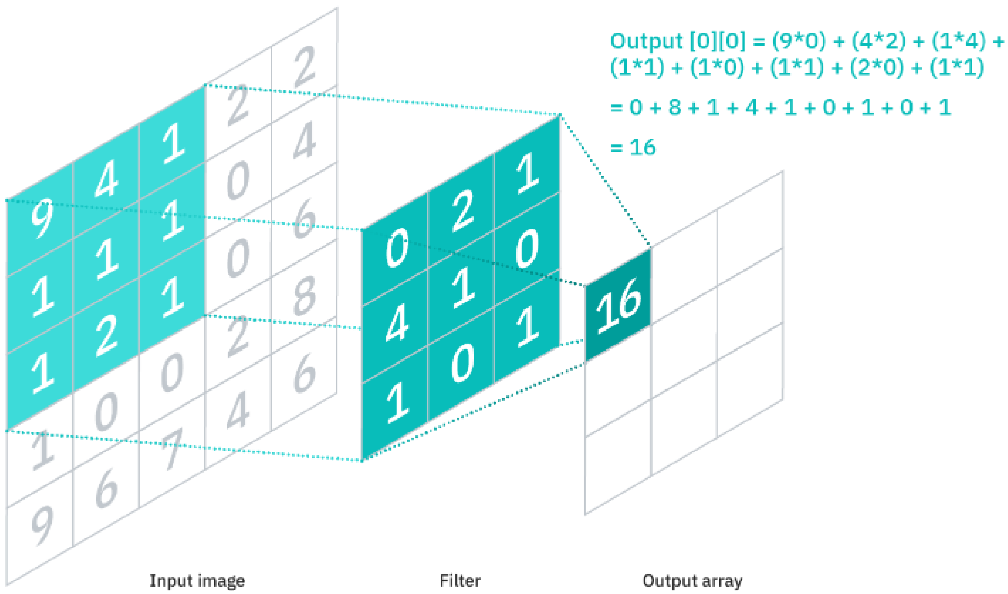

Figure 3: Convolution layer

To set up the convolutional layer, there are a few hyperparameters that need to be known before execution[4].

- Number of filters (F): for example if 5 filters is needed, the feature map will have a depth of 5
- Stride (S): the number of pixel that the filter move over the input image
- Zero-padding(P): a layer of pixel whose value is 0 added around the perimeter of the input image so that the feature map can have the same or bigger size than the input image.

The width and length of output array can be calculated using these formulas below [5]

$$Width = (W-F+2P)/S + 1$$

$$Length = (L-F+2P)/S + 1$$

Where W, L is the width and length of the input array.

Finally, after using a filter, a Rectifier Linear Unit (ReLU) transformation is performed on the feature map[4].

The second type of layer is called the Pooling layer. Its main function is to downsample the input image. This layer also has a filter that goes through the image. However, it doesn't have any weight, and instead it's just a filter function to get the output array[4]. There are two types of such function, which are

- Max pooling: select the pixel in the part of the image that its travels to with maximum value and send that to the output
- Average pooling: calculate the average of the pixel within the part of the image it travels to

The main purpose of the layer is to reduce model complexity and overfitting even though some information might be lost.

The last type of layer is called a fully connected layer. As the name suggests, this layer's node is fully connected to the previous layer's node. Its main function is for classification. Afterwards, a softmax function is applied to the output to get the probability of the input image belonging to which class.[4]

In a typical CNN, a combination of some convolutional layers and a pooling layer is applied at first. After a few of such combinations, a fully connected network is applied at the end to classify the image.

2. **Algorithm performance**

After 200 epochs, below is the results of our CNN.

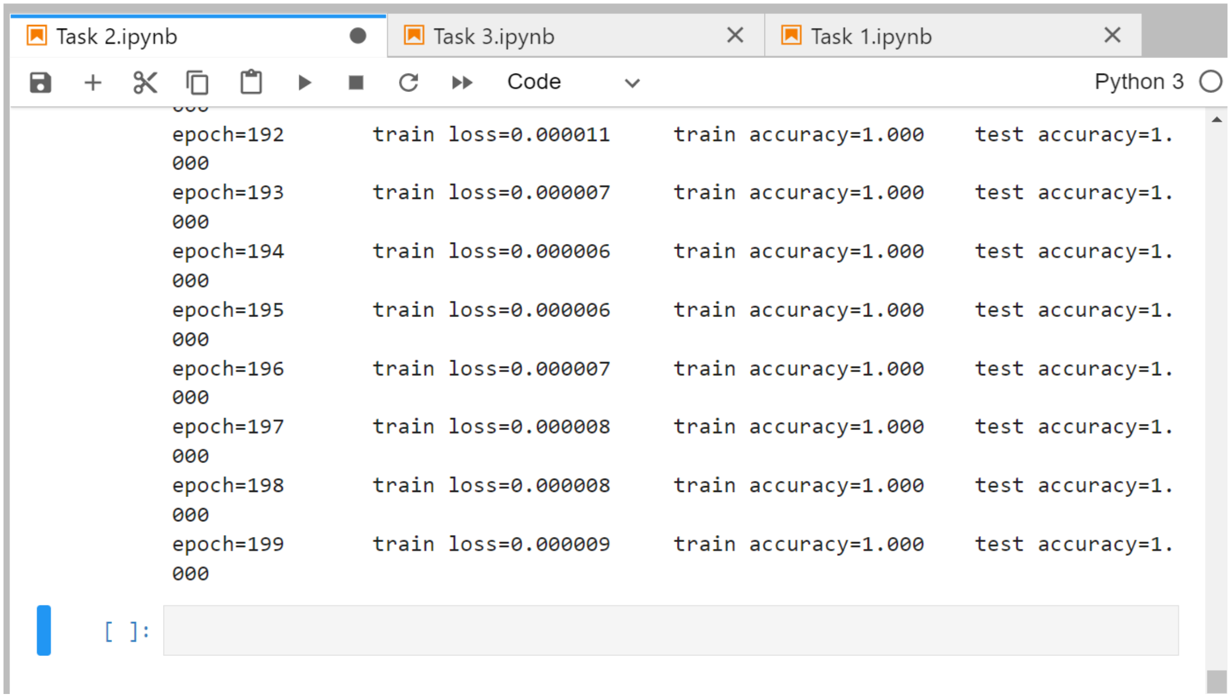

Figure 4: Test result of CNN

As seen from the image, the error in both training and test is approximately 0, which is considered perfect. However, when looking at the dataset given by the EE4705 team, we realized that all the images there show just one hand with the same background. In other words, the dataset lacks variety.

To compare between CNN and Fully-connected models' performance in task 1 and task 2, a bigger dataset with more variety is needed, which has been done in Task 3. The current dataset's performance can't give us the comparison between the 2 models since trend of the train and test accuracy is roughly the same.

Task 3

Literature search

While we initially wanted to take pictures and videos of ourselves posing in different gestures and body language for our dataset, we found this to be counterproductive as there are already existing datasets online which comprise hundreds of thousands of images. Moreover, the images in those datasets vary in lighting, resolution, aspect ratio and the subjects of the images vary in skin tone, body size and other factors which we would not be able to compete with. As such, we thought it would be a fools' errand to come up with our own data set when there are other more comprehensive datasets out there. Hence, we used images from existing datasets like those in the table below, and transformed them to provide a diverse range of image data to train our model.

Human3.6M	Cambridge hand gesture dataset
ChaLearn LAP IsoGD	MultiModal Gesture dataset
BIGHands dataset	QMUL under Grouynd Re-Identification(GRID)
CAMPUS-Human dataset for human re-identaification	Market-1501

The performance of a deep learning model is largely dependent on the size and challenge posed by the training and testing dataset. Computer vision approaches eliminate the need for wearable devices by using large amounts of data to train systems that can generalize previously unencountered scenarios.[7] Large amounts of labeled image data can replace complicated computational pipelines by a single, end-to-end trainable neural network. Images within the dataset, however, must contain different qualities of images taken in various indoor and outdoor environments. This produces a challenging dataset for deep learning.

A large dataset is required to effectively perform deep-learning tasks. However, there are techniques to increase the size of the dataset. For example, in [6], the authors were attempting Appearance based pedestrians' head pose and body orientation estimation using deep learning, for which there was no large dataset available for pedestrian orientation classification.

Technique 1: Collect images from various related existing datasets, such as VIPeR, CAVIAR4REID, ETHZ and Market-1501. In [6], they searched datasets for person re-identification or person reid, which is the problem of matching people across disjoint camera views in multi-camera systems, like intelligent CCTV systems.

Technique 2: Data augmentation via flipping. As some transformed images can be applied to multiple datasets, data augmentation works. For example, as the team in [6] needed to identify the orientation of the people in each image, simply flipping and rotating the images can provide a new data point in the opposite class for deep learning. A body image oriented at 90 degrees is flipped and then added to the class of images with orientation of 270 degrees.

Technique 3: Data augmentation via RGB channels. Each RGB channel from a specific image is treated as a separate image, which can increase the dataset by three-fold.

Algorithm Performance:

```
epoch=30    train loss=2.080710    train accuracy=0.259    test accuracy=0.214
epoch=31    train loss=2.060065    train accuracy=0.268    test accuracy=0.230
epoch=32    train loss=2.059229    train accuracy=0.269    test accuracy=0.223
epoch=33    train loss=2.043662    train accuracy=0.274    test accuracy=0.221
epoch=34    train loss=2.018576    train accuracy=0.279    test accuracy=0.230
epoch=35    train loss=2.012052    train accuracy=0.286    test accuracy=0.223
epoch=36    train loss=2.004555    train accuracy=0.287    test accuracy=0.228
epoch=37    train loss=1.986778    train accuracy=0.289    test accuracy=0.236
epoch=38    train loss=1.973111    train accuracy=0.298    test accuracy=0.223
epoch=39    train loss=1.954801    train accuracy=0.303    test accuracy=0.241
epoch=40    train loss=1.942707    train accuracy=0.310    test accuracy=0.260
epoch=41    train loss=1.934935    train accuracy=0.311    test accuracy=0.224
epoch=42    train loss=1.911970    train accuracy=0.315    test accuracy=0.240
```

Figure 5: Final algorithm performance at epoch=42 for Task 3

The test accuracy achieved at epoch==42 is around ~0.240. I believe that the performance of this model could have been fast-tracked if I had had a GPU to work with, as it is much faster for these sort of computations.

Explanation of Network:

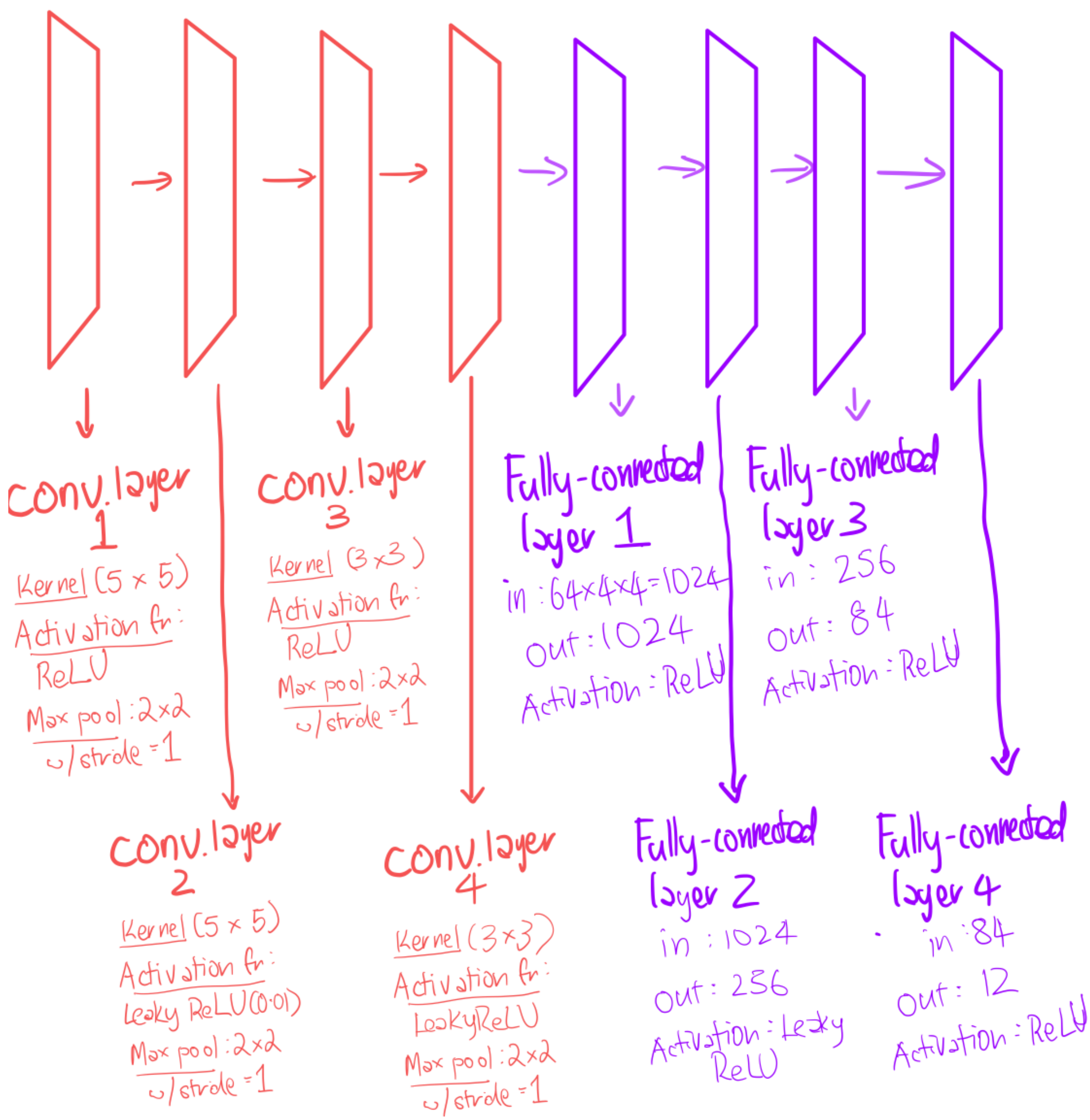

Figure 6: Our neural network diagram

Our neural network consists of 4 convolutional layers and 5 fully connected layers. We used both ReLU and LeakyReLU activation functions to provide complexity to our networks.

S/N	Layer Name	Computation/Explanation	Size = [batch_size, channels, height, width]
1	Input Layer (0th layer)	Initial image sizes from the datasets varied in sizes, thus we had to come up with a standardized size to use for the .resize() function. This was chosen to be (height =100, width=100)	[8,3,100,100]
2	After 1st convolutional layer	(Input - filter)/stride + 1 = (100-5)/1 + 1 = 96 → 96x96; Channel is changed to 6	[8,6,96,96]
3	After 1st max pooling	Since we use a 2x2 max-pool with a stride=2, the size is halved in both height and width.	[8,6,48,48]
4	After 2nd convolutional layer	(Input - filter)/stride + 1 = (48-5)/1 + 1 = 44 → 44x44; Channel is changed to 6	[8,6,44,44]
5	After 2nd max pooling	Since we use a 2x2 max-pool with a stride=2, the size is halved in both height and width.	[8,6,22,22]
6	After 3rd convolutional layer	(Input - filter)/stride + 1 = (22-3)/1 + 1 = 20 → 20x20;	[8,6,20,20]
7	After 3rd max pooling	Since we use a 2x2 max-pool with a stride=2, the size is halved in both height and width.	[8,6,10,10]
8	After 4th convolutional layer	(Input - filter)/stride + 1 = (10-3)/1 + 1 = 8 → 8x8;	[8,6,8,8]
9	After 4th max pooling	Since we use a 2x2 max-pool with a stride=2, the size is halved in both height and width.	[8,6,4,4] → passed to fully connected layers

Table 1: Data progression as it passes through our network

Problems encountered:

1. Overfitting Part 1

```
epoch=62        train loss=0.001194     train accuracy=1.000     test accuracy=0.224
epoch=63        train loss=0.001084     train accuracy=1.000     test accuracy=0.218
epoch=64        train loss=0.001001     train accuracy=1.000     test accuracy=0.224
epoch=65        train loss=0.000936     train accuracy=1.000     test accuracy=0.218
epoch=66        train loss=0.000859     train accuracy=1.000     test accuracy=0.221
epoch=67        train loss=0.000805     train accuracy=1.000     test accuracy=0.214
epoch=68        train loss=0.000766     train accuracy=1.000     test accuracy=0.214
epoch=69        train loss=0.000727     train accuracy=1.000     test accuracy=0.205
epoch=70        train loss=0.000689     train accuracy=1.000     test accuracy=0.211

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [51], in <cell line: 3>()
```

Figure 7: Initial performance metrics

Initially, we had test data around 128 images per class for 12 classes. At this point, we reused the same convolutional neural network from Task 2 for this task, but with our custom dataset. While we had very good performance on training data (accuracy ==1.00) we also had very poor performance on testing data (maximum accuracy==0.225), especially when compared to the test accuracies of Task 1 and Task 2. Since my dataset size was much bigger in Task 3 than for Task 1 and Task 2, I decreased my epoch size to 100 from 200. By doing this, I hoped to lessen the strain on my CPU. As a result, even though it took up to 20 seconds to run through each epoch, results remained poor with little change.

Initially, batch size was set at 8,which was higher than previous tasks at 4 as I read that higher batch sizes will make the model run faster and be more generalized[13]. Since increase in data size will lead to stronger performance[13], we also increased data size for each class to at least 250+ images per class. Another thing we did was to change the training to testing ratio from 80% training data to 75% training data. We did this to combat the issue that we perceived as overfitting.

2. Overfitting part 2

```
epoch=22        train loss=0.077689     train accuracy=0.991    test accuracy=0.218
epoch=23        train loss=0.053397     train accuracy=0.996    test accuracy=0.218
epoch=24        train loss=0.171220     train accuracy=0.946    test accuracy=0.214
epoch=25        train loss=0.116957     train accuracy=0.971    test accuracy=0.203
epoch=26        train loss=0.088218     train accuracy=0.977    test accuracy=0.214
epoch=27        train loss=0.056084     train accuracy=0.987    test accuracy=0.226
epoch=28        train loss=0.023873     train accuracy=0.997    test accuracy=0.225
epoch=29        train loss=0.019898     train accuracy=0.997    test accuracy=0.215
epoch=30        train loss=0.018915     train accuracy=0.998    test accuracy=0.213
epoch=31        train loss=0.162970     train accuracy=0.948    test accuracy=0.199

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [58], in <cell line: 3>()
```

Figure 8: Despite trying to reduce overfitting, our initial model was not bearing results

After making all the changes in Overfitting part 1, results did not seem to change much. Although only 30 epochs have passed in this iteration of the code, testing accuracy is still around 0.2, whereas training accuracy is almost perfect, at 0.95. This means that the model has learned particulars that help it perform better in my training data that are not applicable to the larger data population, which results in subpar performance. The model is still overfit! After referring to [12], we realized there were a few other changes to be made, though.

We then increased the learning rate to 0.01 after referring [11], weight decay to 0.001 after referring to [10], and batch size to 16 after referring to [13].

```
epoch=2 train loss=2.480477     train accuracy=0.090    test accuracy=0.098
epoch=3 train loss=2.481259     train accuracy=0.093    test accuracy=0.098
epoch=4 train loss=2.480475     train accuracy=0.089    test accuracy=0.116
epoch=5 train loss=2.480226     train accuracy=0.096    test accuracy=0.098
epoch=6 train loss=2.481130     train accuracy=0.088    test accuracy=0.098
epoch=7 train loss=2.480720     train accuracy=0.087    test accuracy=0.116
epoch=8 train loss=2.480838     train accuracy=0.091    test accuracy=0.084
epoch=9 train loss=2.481220     train accuracy=0.093    test accuracy=0.116
epoch=10        train loss=2.480986     train accuracy=0.088    test accuracy=0.098

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [76], in <cell line: 3>()
```

Figure 9: Results of Overfitting part 2

3. **Reconfiguring fully connected layers**

```
# initialize first (and only) set of FC => RELU layers
self.fc1 = Linear(in_features=16*5*5, out_features=84)
self.relu3 = ReLU()
self.fc2 = Linear(in_features=84, out_features=num_classes)
```

Figure 10: Parameters for fully connected layer

Initially, the parameters of the first fully connected layer, fc1, was in_features ==400, out_features == 256, while the parameters of the second fully connected layer, fc2, was in_features==256, out_features==12. However, we found that this did not converge to a desirable accuracy in adequate time. Thus, we changed the out_feature of fc1and in_feature of fc2 to 84.

4. **Change of neural network entirely**

In the end, we came to the conclusion that the CNN we used for task 2 was much too low end for task 3. After referring to multiple literature such as [9][10][11], we realized that our model can be increased in complexity by adding more layers. Thus, we added a few more layers, which was shown initially in Figure 6. After adding 2 more convolutional layers and 2 more fully connected layers, we realized that the complexity better fit the project. However, we realized now that the problem is underfitting, rather than overfitting as our training accuracy was low now as well!

```
epoch=67    train loss=2.246788    train accuracy=0.221    test accuracy=0.167
epoch=68    train loss=2.242902    train accuracy=0.222    test accuracy=0.185
epoch=69    train loss=2.243480    train accuracy=0.225    test accuracy=0.167
epoch=70    train loss=2.239975    train accuracy=0.225    test accuracy=0.173
epoch=71    train loss=2.236735    train accuracy=0.225    test accuracy=0.168
epoch=72    train loss=2.236321    train accuracy=0.221    test accuracy=0.182
epoch=73    train loss=2.231464    train accuracy=0.227    test accuracy=0.187
```

Figure 11: Problem went from overfitting to underfitting

```
In [5]: #classes: applauding, blowing bubbles, brushing teeth,
        #drinking, holding umbrella, jumping, phoning, reading,
        #running, textinf, waving, writing
        model = CNNModel(num_classes=12)
        if torch.cuda.is_available():
            model = model.cuda()
            print("cuda activated")
        optimizer = torch.optim.Adam(model.parameters(), lr=0.00001, weight_decay=0.0001)
        loss_func = CrossEntropyLoss()
```

Figure 12: Experimenting with learning rates and weight decays

We also experimented with a multitude of different learning rates (0.0000001, 0.000001, 0.0001, 0.001, 0.1) and a multitude of weight_decays (0.000000001, 0.00000001, 0.0000001, 0.00000, 0.0001, 0.001) and tried all different combinations in between. We eventually settled on a learning rate of 0.0001 and a weight_decay of 0001.

What we could have done better

One thing we could have done to improve our model is reduce the number of classes. Some of the classes we used included objects, like holding an umbrella, texting on a phone, calling someone on the phone. However, these images were much more challenging for our model as it included both body gestures, as well as the recognition of the object itself. The amount of data to train the model for these types of classes was not enough compared to what our computer CPUs could handle.

Another thing we could have done is data augmentation. As seen in our literature search, we could have separated the color channels, flipped the images, rotated the images to make sure we had increased and varied data. However, our CPUs could barely handle the data as it is and eventually decided against it as increasing the data size would only increase the time length of the epochs.

References:

[1] P. Mahajan, “Fully connected vs Convolutional Neural Networks,” *Medium*, 23-Oct-2020. [Online]. Available: https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5

[2] H. Liu and L. Wang, “Gesture recognition for human-robot collaboration: A Review,” *International Journal of Industrial Ergonomics*, 06-Mar-2017. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0169814117300690.

[3]T. T. Contributor, “What is Convolutional Neural Network? - definition from whatis.com,” *SearchEnterpriseAI*, 26-Apr-2018. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network.

[4]By: IBM Cloud Education, “What are convolutional neural networks?,” *IBM*. [Online]. Available: https://www.ibm.com/cloud/learn/convolutional-neural-networks.

[5]“Pytorch tutorial 14 - Convolutional Neural Network (CNN),” *YouTube*, 07-Feb-2020. [Online]. Available: https://www.youtube.com/watch?v=pDdP0TFzsoQ.

[6] M. Raza, Z. Chen, S.-U. Rehman, P. Wang, and P. Bao, “Appearance based pedestrians’ head pose and body orientation estimation using Deep Learning,” *Neurocomputing*, vol. 272, pp. 647–659, 2018.

[7] J. Materzynska, G. Berger, I. Bax, and R. Memisevic, “The jester dataset: A large-scale video dataset of human gestures,” *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019.

[8] Dataset used: http://vision.stanford.edu/Datasets/40actions.html

[9] https://pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html

[10] https://towardsdatascience.com/this-thing-called-weight-decay-a7cd4bcfccab

[11]https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0

[12]
https://towardsdatascience.com/deep-learning-3-more-on-cnns-handling-overfitting-2bd5d99abe5d

[13]
https://stats.stackexchange.com/questions/352036/what-should-i-do-when-my-neural-network-doesnt-learn