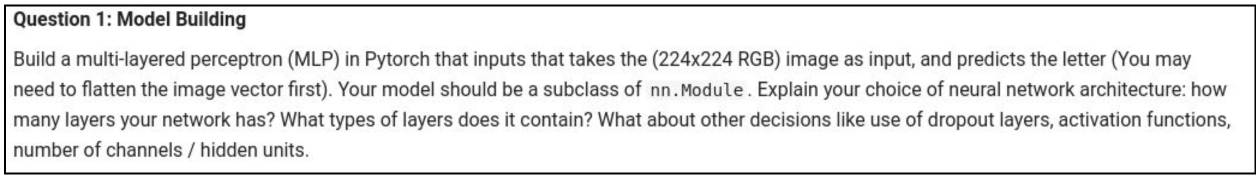
SMAI

Programming Assn

Kushagra Agarwal
2018113012

Link to Google collab Notebook:
https://colab.research.google.com/drive/1FTFckCEAwrG5Ppw2pGS8WmOhN9_weS5l?usp=sharing


Question 1: Model Building

Build a multi-layered perceptron (MLP) in Pytorch that inputs that takes the (224x224 RGB) image as input, and predicts the letter (You may need to flatten the image vector first). Your model should be a subclass of `nn.Module`. Explain your choice of neural network architecture: how many layers your network has? What types of layers does it contain? What about other decisions like use of dropout layers, activation functions, number of channels / hidden units.

Total Number of Layers = 6 (1 Input, 4 Hidden, 1 output)

Input Layer: Number of input nodes = 224 * 224 * 3 = 150528 (as each image is a 224 pixel x 224 pixel with each pixel having 3 values for Red, Green and Blue).

Second layer: 384 nodes

Third Layer: 128 nodes

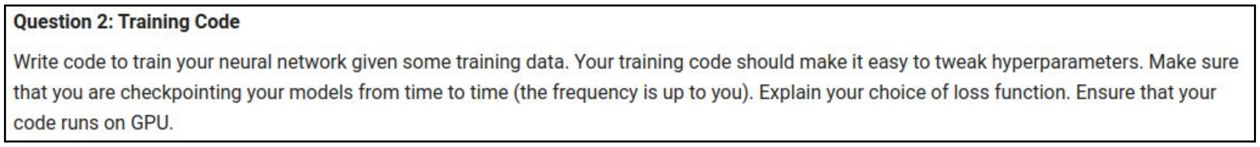
Fourth Layer: 64 nodes

Fifth Layer: 32 nodes

Output Layer: 9 Nodes (Equal to the number of classes 0-8)

All the layers are Linear Layers and I have used ReLu activation (all the outputs are positive hence not an issue).

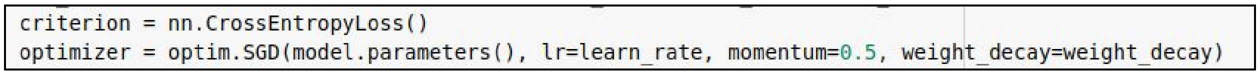
This architecture was used as the input number of nodes was high and to counter the complexity of the data, I used a 6 layer deep model.

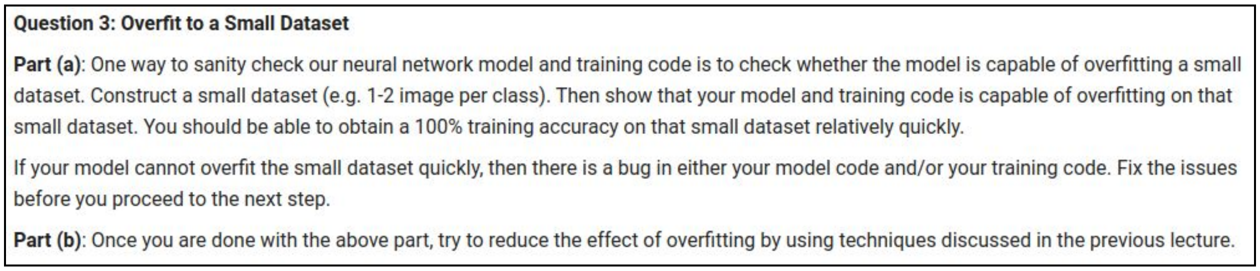

Question 2: Training Code

Write code to train your neural network given some training data. Your training code should make it easy to tweak hyperparameters. Make sure that you are checkpointing your models from time to time (the frequency is up to you). Explain your choice of loss function. Ensure that your code runs on GPU.

Code allows easy tweaking of the hyperparameters.

Training step: Used Cross-Entropy Loss as this is a multi-class classification problem and the correct class can be selected based on Log Loss. I also used the Stochastic Gradient Descent for the optimization.


```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=learn_rate, momentum=0.5, weight_decay=weight_decay)
```



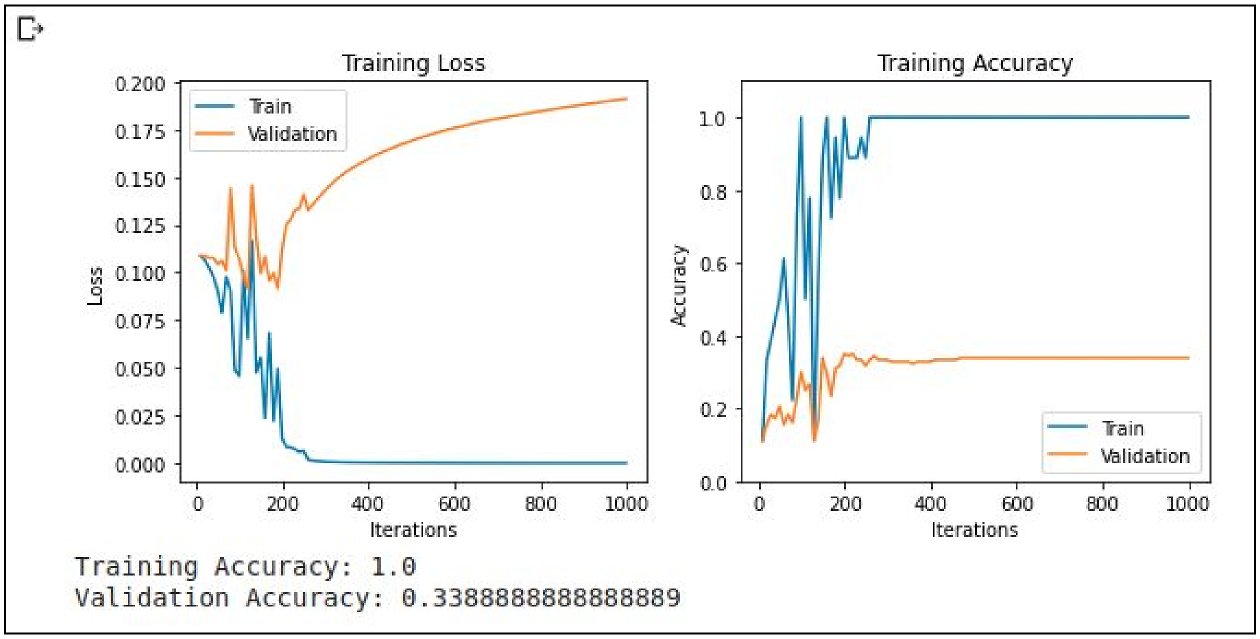
Question 3: Overfit to a Small Dataset

Part (a): One way to sanity check our neural network model and training code is to check whether the model is capable of overfitting a small dataset. Construct a small dataset (e.g. 1-2 image per class). Then show that your model and training code is capable of overfitting on that small dataset. You should be able to obtain a 100% training accuracy on that small dataset relatively quickly.

If your model cannot overfit the small dataset quickly, then there is a bug in either your model code and/or your training code. Fix the issues before you proceed to the next step.

Part (b): Once you are done with the above part, try to reduce the effect of overfitting by using techniques discussed in the previous lecture.

A) With Overfitting we can observe that the Training Accuracy is 100%. But as the model did not understand the data and simply overfit to the training examples, we can see that the validation accuracy is only 33.88%. From the graph, we can see that as the training accuracy increases and goes to 100%, the validation accuracy also reaches a stable 33.88%. Initially, when the model was starting to overfit, the validation loss started to increase, showing that overfitting took place.



```
Training Accuracy: 1.0
Validation Accuracy: 0.3388888888888889
```

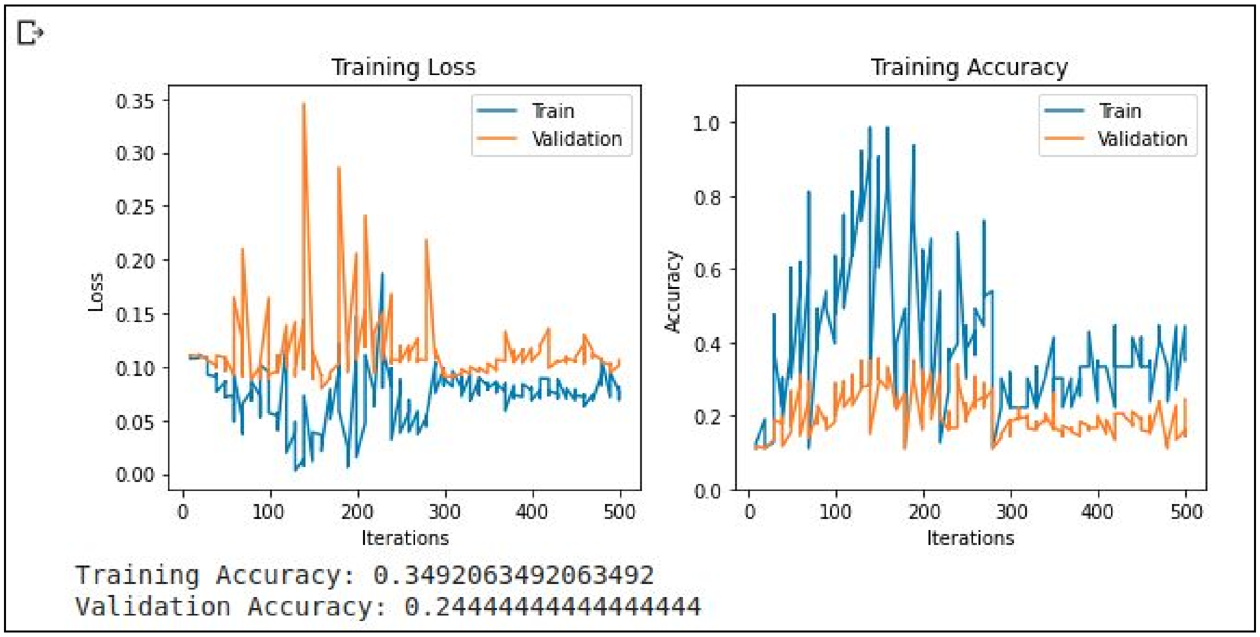
B) To reduce overfitting the following methods were employed:

i) Data Augmentation

ii) Weight Decay

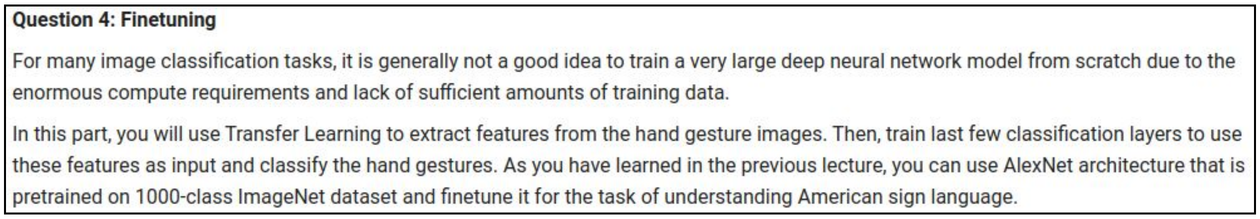
iii) Dropout Regularisation with 40% Dropout rate in the layers

iv) Reduced Number of Iterations from 1000 to 500



```
Training Accuracy: 0.3492063492063492
Validation Accuracy: 0.24444444444444444
```

Training accuracy dropped to 34.92% and the Validation accuracy also dropped to 24.44%.

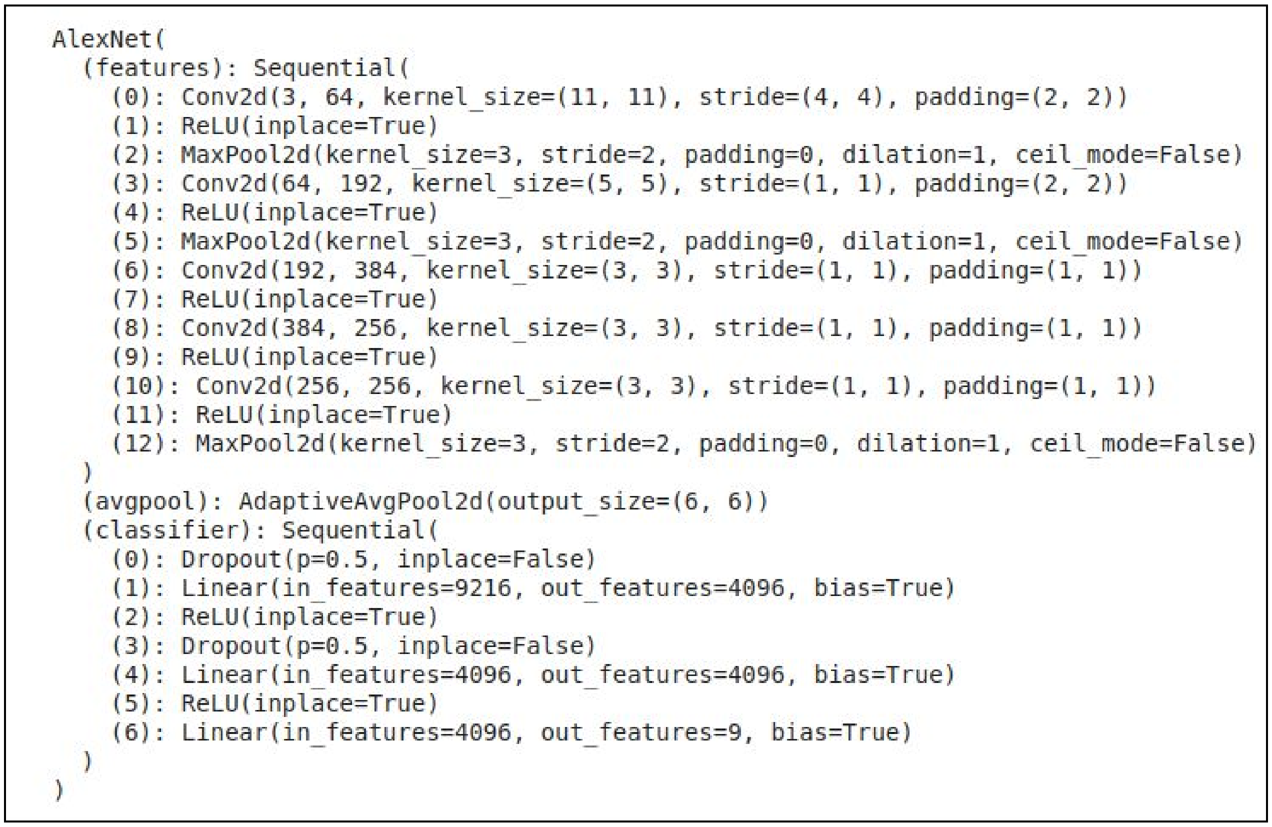


Question 4: Finetuning

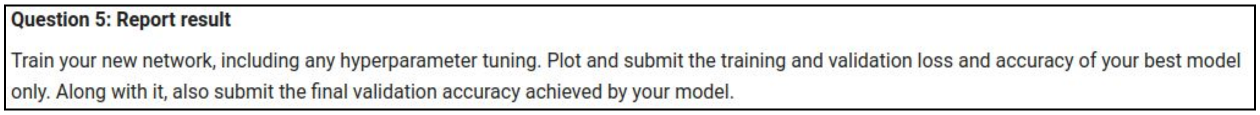
For many image classification tasks, it is generally not a good idea to train a very large deep neural network model from scratch due to the enormous compute requirements and lack of sufficient amounts of training data.

In this part, you will use Transfer Learning to extract features from the hand gesture images. Then, train last few classification layers to use these features as input and classify the hand gestures. As you have learned in the previous lecture, you can use AlexNet architecture that is pretrained on 1000-class ImageNet dataset and finetune it for the task of understanding American sign language.

Using a Pre-trained AlexNet model to employ Transfer Learning. The last layer of the model was changed to 9 nodes as the output has 9 classes. Used Cross-Entropy and Stochastic Gradient Descent. AlexNet is a pre-trained model on the Image-1000 dataset and such models are good to use as these models learn the features for an image and then we can retrain the last layers for our specific purpose.



```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=9, bias=True)
  )
)
```



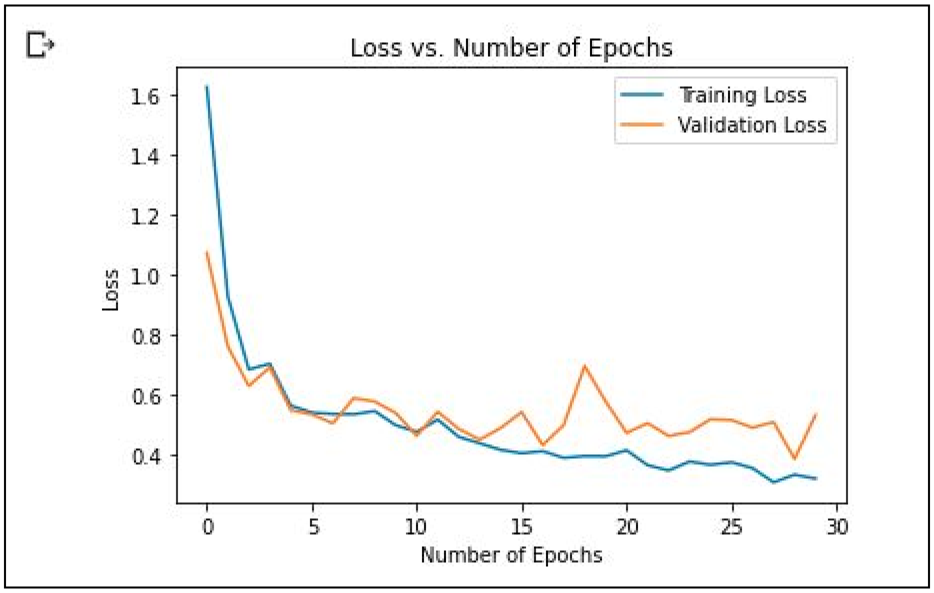
Question 5: Report result

Train your new network, including any hyperparameter tuning. Plot and submit the training and validation loss and accuracy of your best model only. Along with it, also submit the final validation accuracy achieved by your model.

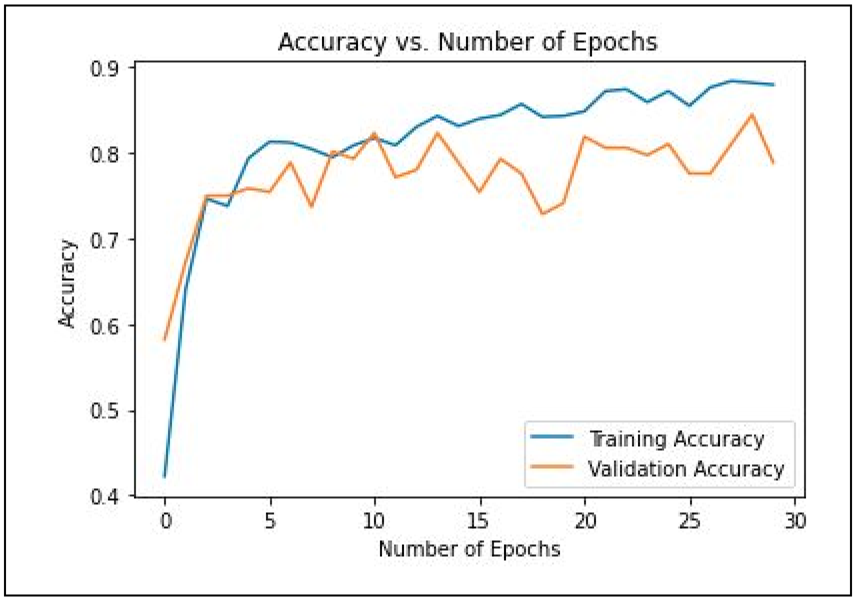
Both the Training Accuracy and the Validation Accuracy increased well for the first few epochs and then nearly settled with more epochs. Similarly, high losses are observed initially but then they decrease and gradually become a small value.

The final validation accuracy for the best model is 84.4828%, which is much higher than the accuracy for the Multi-Layer Perceptron which was 17.2222%. Therefore, we can conclude that pre-trained models which are a complex combination of Linear, Polynomial, and Convolutional Neural Layers trained on Image-based Datasets are preferred.

Plot for Losses for Training and Validation:



Plot for Accuracy for Training and Validation:


Accuracy vs. Number of Epochs
Training Accuracy
Validation Accuracy
Accuracy
Number of Epochs