HACETTEPE
ÜNİVERSİTESİ

BBM 418 ASSIGNMENT 3

NAME: Erhan

SURNAME: Kabaoğlu

NUMBER: 21627389

INTRODUCTİON

We have hands image data set and this data set include information about hands which are include information about gender and aspect of hands(for our assignment).

In this assignment, we should classify this images into 8 classes.

We used deep learning approch (CNN architecture) to solve this problem.

This assignment consist of two parts.

First of it, We should build own CNN architecture and select our hyperparameters.

Second of it, We should use pretrained resnet18 architecture and select our hyperparameters to get more accurate results.

Finally compare results of these two parts.

Using different architecture (layer sizes, hidden layer number, activation functions) and different hyperparameters(learning rate, batch-size, epoch number, regularization(drop-out, L2), loss function, optimazer, momentum) is main goal of this assignment, and I will show you experimental results about this things.

IMPLEMENTATİON DETAİLS

Dataset: I implemented HandDataset class which is inherit Dataset class from PyTorch. This class read cvs file for labels from a directory and from h5py file for images from directory into constructor.

Return image and labels given index using transforms into __getitem__ fucntion. Return type is dictionary.

Note: I convert images to .h5 file because read from Google drive is very slow and eliminate GPU performance so I directly get into memory images ones. I give implementation in source code.

Dataloader: I implemented a function named splitTrainValidationTest. This function get paramaters dataset and batch size. I cuted dataset into %60 train set %20 validation set and %20 test set. Using SubsetRandomSampler from pytorch and using numpy library I loaded to data this three set with DataLoader function.

CNN module: I implemented an util class named UNIT which inherit nn.Module from Pytorch. This module just consist of conv layer.

More detail, I created conv2d layer with 3x3 kernal because of 3x3 size is generally give good result. After that, I applied batch normalization to my conv layers to speed up training time. (Batch normalization normalize layers with abstraction by mean and diveded by strandart deviation).

Finally, I applied Relu function to get activation layers (I choosed relu because of computationally less than sigmoid or tanh).

After all, I implemented main modul class named CNNNet2 and CNNNet.

For CNNNet, I used 6 conv layer and 3 fc layer.

More detail, I implemented sequencly like this:

Conv1->conv2->conv3-> pool1->conv4-> conv5-> conv6-> pool1-> fc1-> fc2-> fc3 (Also I applied Relu to fc layers except for last fc layer).

I used max pooling layer to reduce size of conv layer and extract features.

Pooling layer kernal size is 2x2 with stride 2.

For regularization, I defined drop-out fucntion by default 0, and I applied drop-out fc1 and fc2 layers. I will give more detail in experiment part for drop-out.

For CNNNet2, I used 16 conv layers and 4 fc layers.

More detail, I implemented sequencly like this:

Conv1->conv2->conv3-> pool1->conv4-> conv5-> conv6-> pool1->conv7->conv8->conv9-> pool1->conv10-> conv11-> conv12-> pool1->conv13->conv14->conv15->pool1-> fc1-> fc2-> fc3 -> fc4 (Also I applied Relu to fc layers except for last fc layer).

Like CNNNet, I used 2x2 pooling layer with stride 2.

I used drop-out for fc1,fc2 and fc3.

I defined a function named adjust drop-out to change drop-out dynamiclly at train time.

This function apply drop-out according to epoch number and test,validation accuracy. I will give more information in experiment part for drop-out.

Test(validation) Function: I implemented a function named testNN.

This function takes cnn model,criterian and loader parameters.

I used this function to calculate validation loss and validation accuracy into train function.

Function runs with torch.no_grad mode because we don't need to backprop in this function. And network runs eval mode.

Train Fucntion: I implemented a function named train.

This function is actually main function for this assignment.

It takes parameters which are cnn model,criterian,optimizer,epoch number,Path to save results and learning schedular to adjust learning rate dynamicly at train time.

In this function, I trained my cnn network.

Firstly, I applied forward propagation to get output with mini-batch.

Secondly, I calculated loss using loss function.

Thirdly, I calculated gradiants using loss.bacward function.

Pytorch create dynamicly computationaly graph for network, and when we call backward, it calculate all gradiants using chain rule.

Finally, I updated parameters using optimizer step.

SGD optimizer updates parameters like W = W - alpha*dW(alpha=learning rate).

I explained main purpose of this function.

Also,I defined 5 list to keep information about train accuracy,test accuracy,train loss,test loss for one epoch and total train loss for individual mini-batch.

End of the function I saved this information to plot graphs.

Also, I saved best validation result.

Test Function: I implement this function to get test set accuracy.

This function same as Test(validation) function,

except: I didn't calculate loss values in this function and I store prediction and true results to draw confusion matrix.

Plot functions: I implemented 3 plot function to plot graphs.

This functions take parameters from saved information by train function.

It shows plots and save them as image.

Convert H5 Function: I implemented this function to convert images into .h5 file

I used this because getting image from disk or Google drive is very slow.

It iterate all image, resize to 256x256x3 and save into Hand.h5 file.

EXPERIMENTS

Part I

****

Batch size: 16

Learning rate: 0.0005 -> 0.00001 -> 0.000005 -> 0.0000001 (for each 6 epoch)

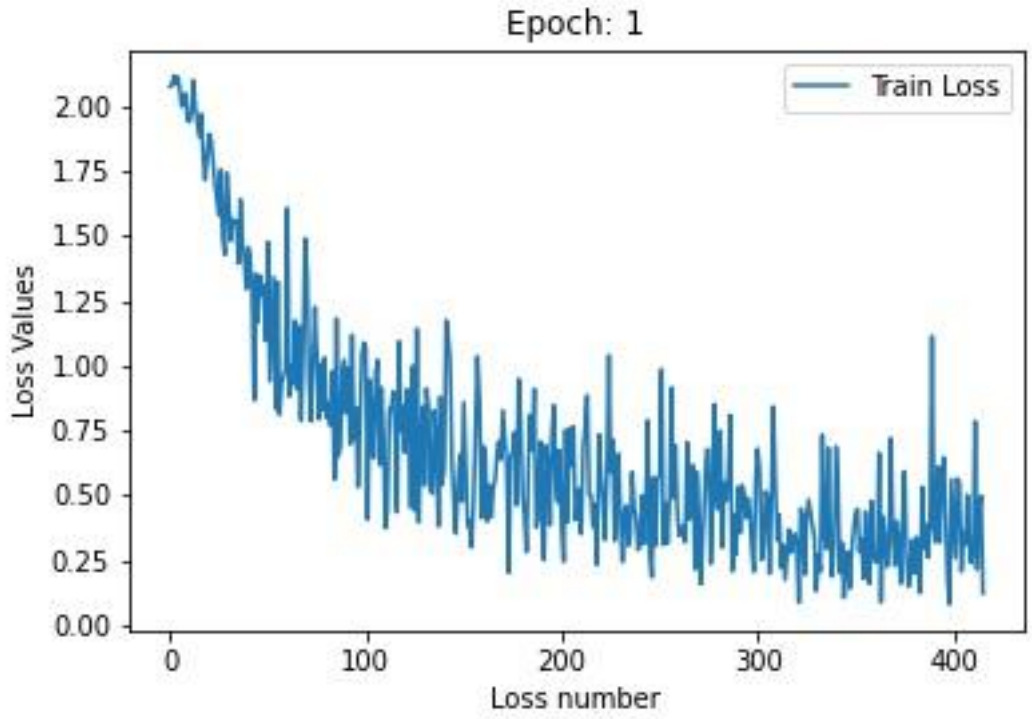
Epoch Number: 30

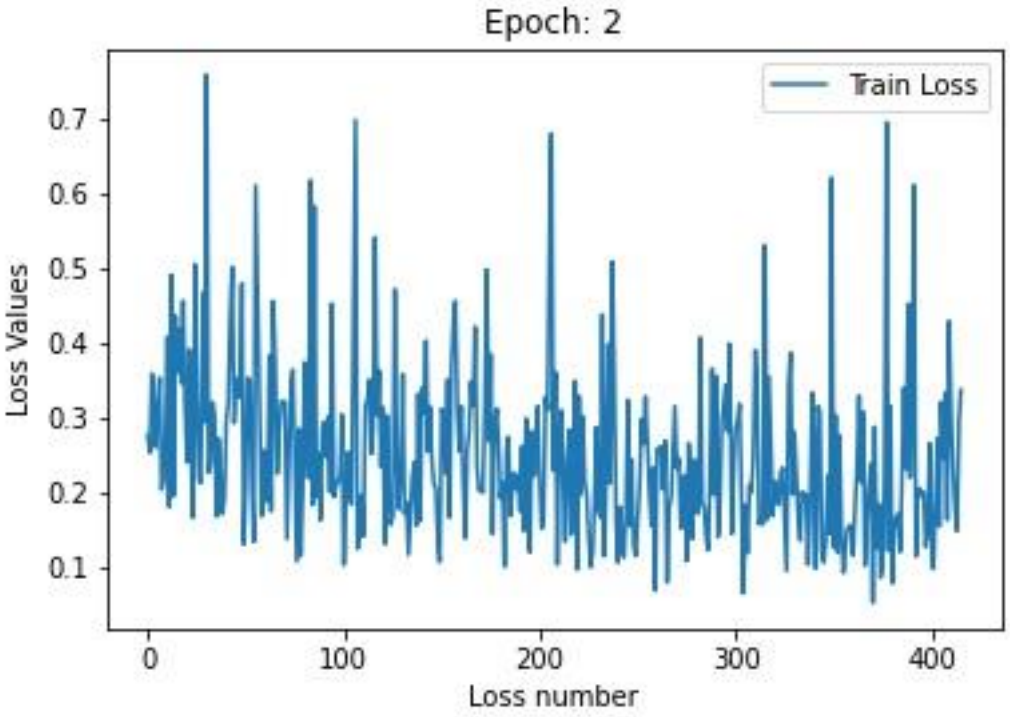
Input size: 1200x1600x32

I used CNNNet architecture.

In my first experiment, I didn’t resize my image so i used 1200x1600x3 as input.

So I choose my batch size 16 because gpu memory is not enough for bigger batch size and train time take long time for lower batch size.



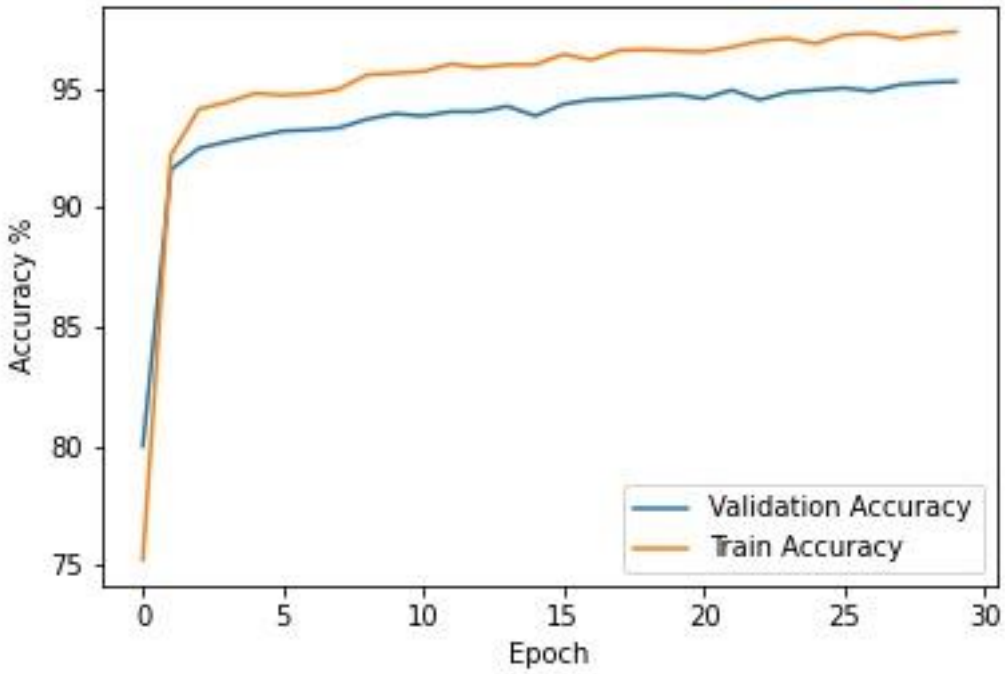


I just want to show for 2 epoch losses, the rest is decrasing slowly and stay same after a while (similar to epoch 2).

I first try learning rate 0.001 but this learning rate not get good result, my loss didn’t decrase.

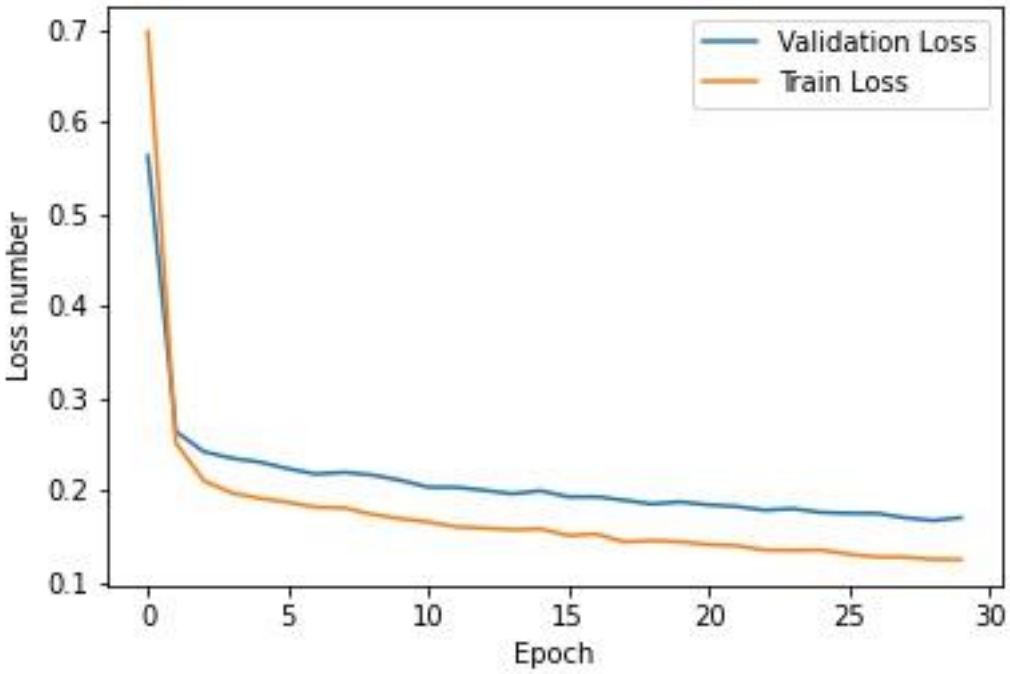
I think this learning rate big to find global optimum because of high resolution(1200x1600x3).

So I try 0.0005 first and decrease learning rate dynamicly at train time.



Here is train and validation accuracy. I reached approximatly 99 train accuracy and 95 validation accuracy.

The difference between train and validation accuracy based on over fitting.



Here is loss values respect to epoch number. It acts parallel validation and train accuracy.

So according to this result I see this experiment affected by over fitting and because of high resolution traning time take very long time and memory space.

I didn’t use drop-out for this experiment.

(I didn’t use .h5 file for this experiment).

****

Batch size: 128

Learning rate: 0.0000001

Epoch Number: 1000

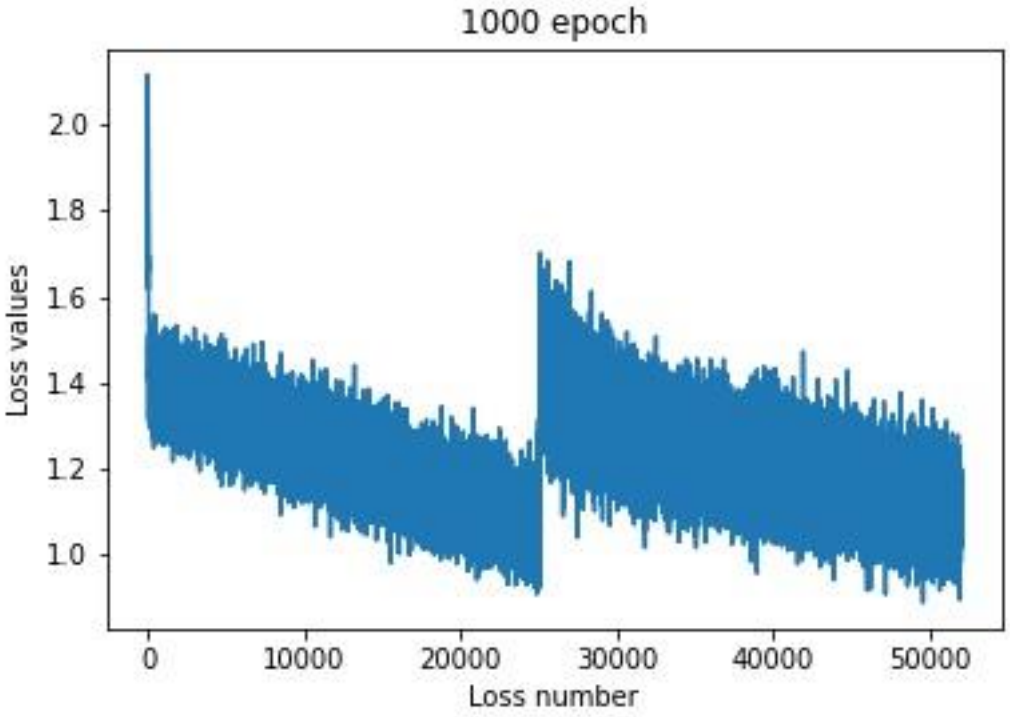
Input size: 32x32x3

I used CNNNet architecture.

In this experiment, I resized images to 32x32x3.

I choosed batch size 128.

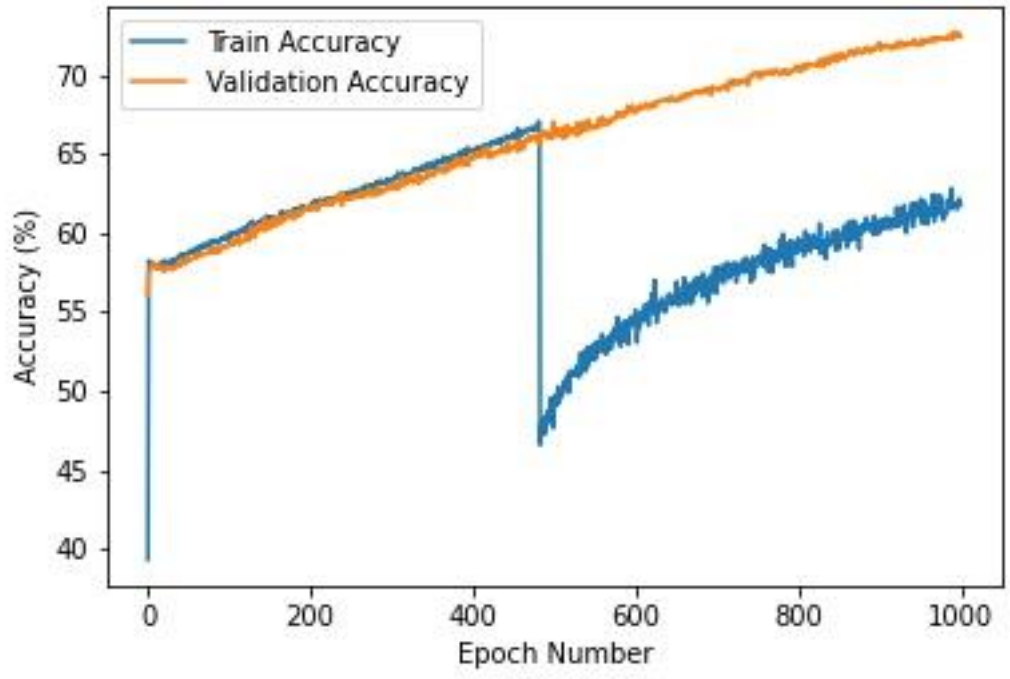
I tried 4 different learning rate shown above.



Here is loss values for 1000 epoch.

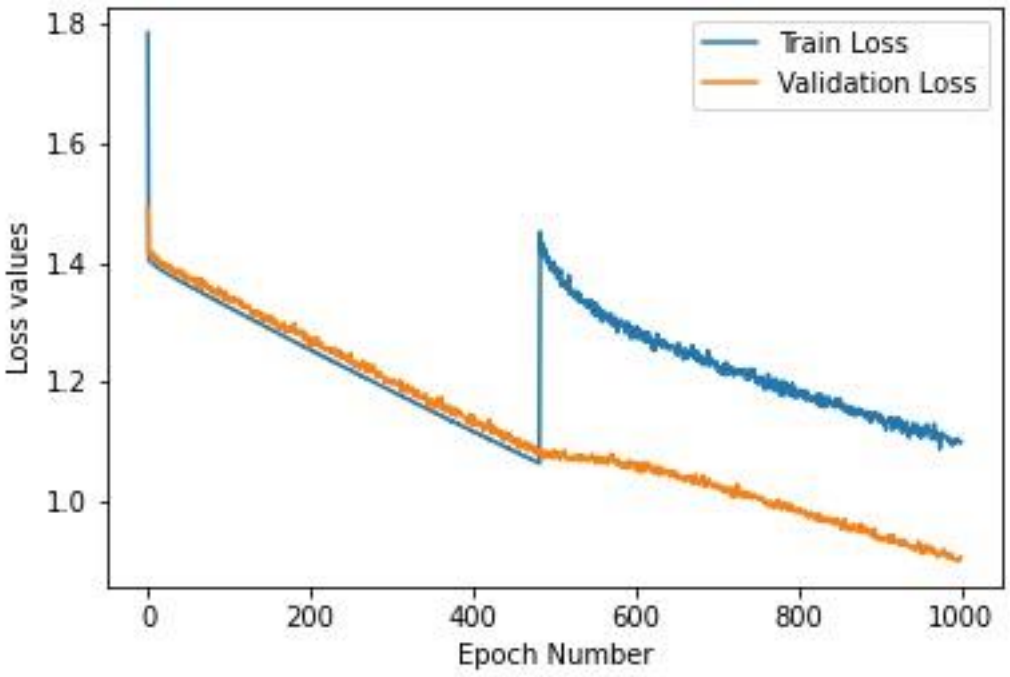
In first epochs learning rate decrase rapidly after that decrase slowly.

As you can see learning rate increase immediately because of adding drop-out.



As you can see when I applied drop-out train accuracy decrease and validation accuracy still increase. This show us validation accuracy don't affected by dropout.

If I try more epoch train accuracy reach to validation accuracy and we can avoid over fitting.



Here is train and validation loss. It acts parallel validation and train accuracy.

We can see drop-out affect.

In this exprement, I avoid overfitting with resizing smaller image(low resolution, low features).

But learning rate is so small and because of this model learn very slowly.

****

Batch size: 64

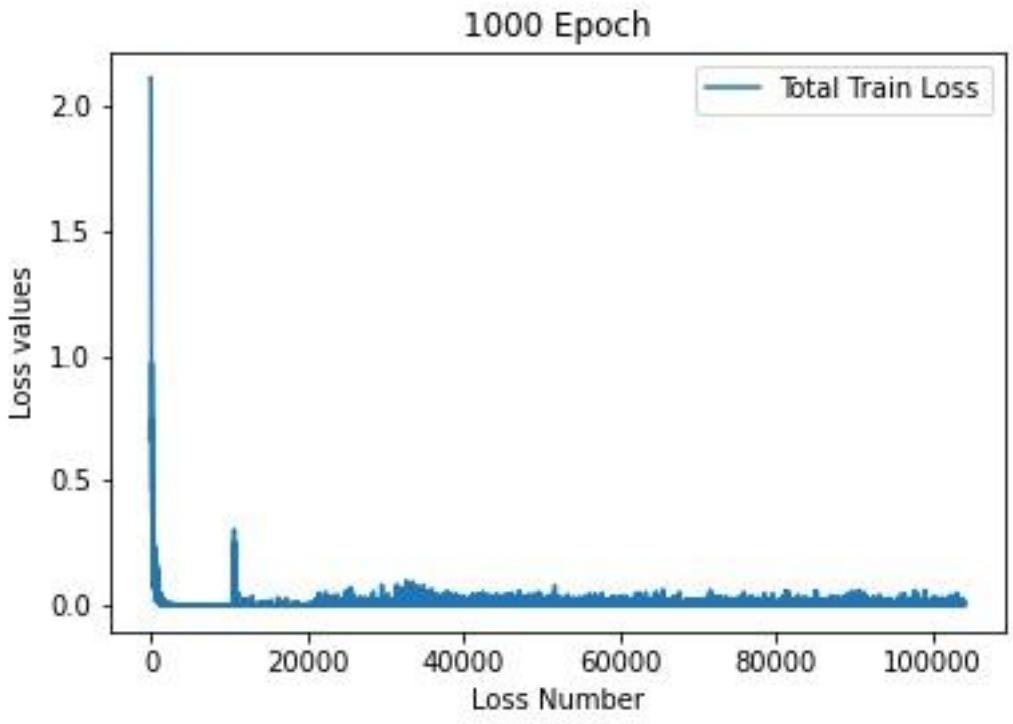
Learning rate: 0.001 -> 0.0001 -> 0.00001 -> 0.000001 (for each 200 epoch)

Epoch Number: 1000

Input size: 32x32x3

I used CNNNet architecture.

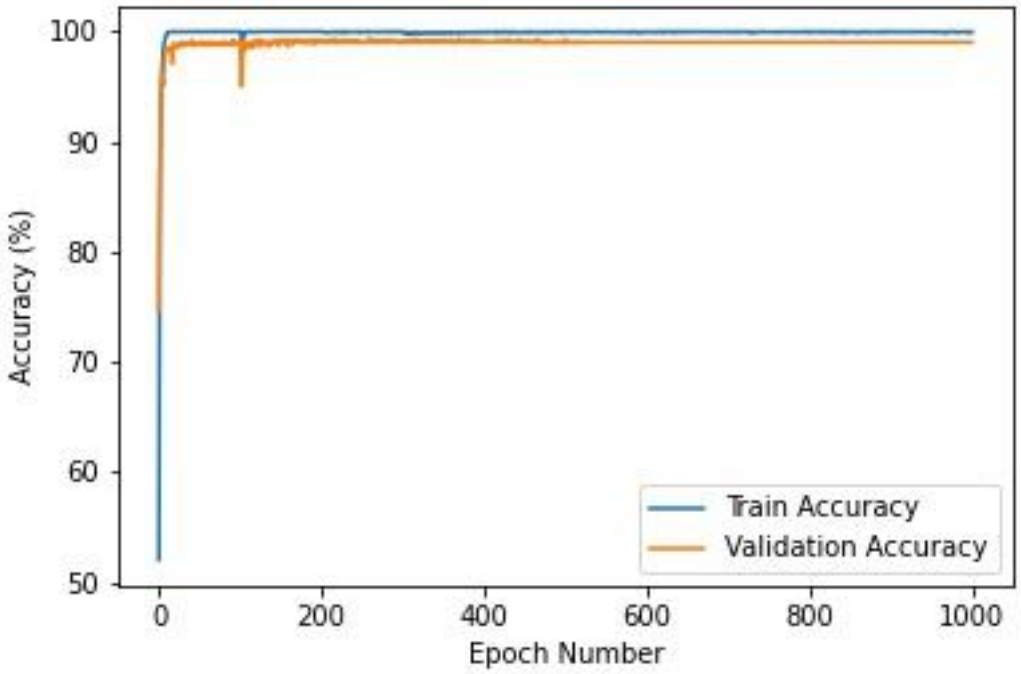
In previous experiment, I used very small learning rate so model trained very slowly. In this exprement I changed batch size 128 to 64 and learning rate 0.000001 to 0.001 and decrase it dynamicly.

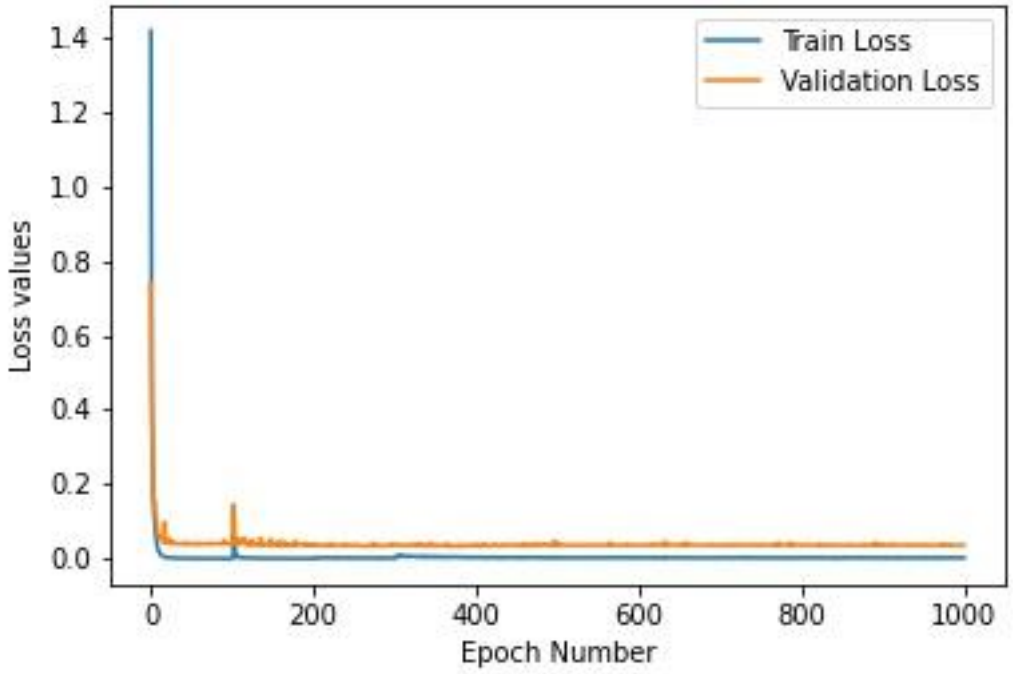


Here is result for loss numbers.

As you can see, I immediately reach small loss numbers.

I applied 0.1 drop-out at 100. epoch so you can see this affect from graph.





You can see accuracies and loss numbers for each epoch.

I reachedd %100 train accuracy and %99,4 validation accuracy.

I think this model learned well.

Validation accuracy is very close to train accuracy and there is not over fitting and under fitting.

You can see fluctuations because of drop-out.

****

Batch size: 64

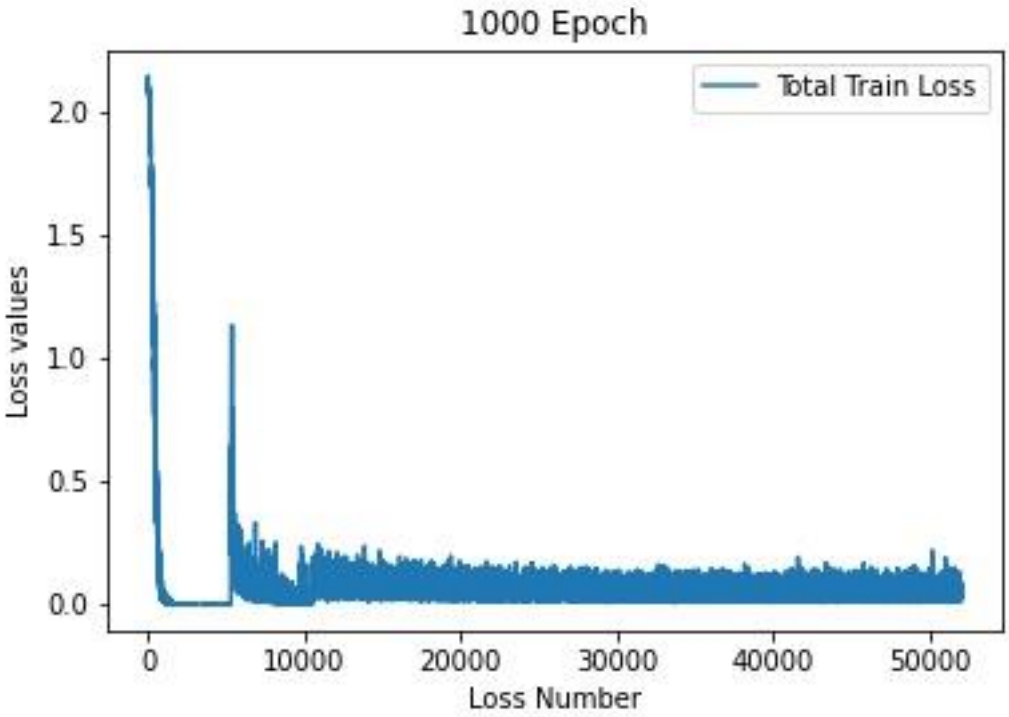
Learning rate: 0.001 -> 0.0001 -> 0.00001 -> 0.000001 (for each 200 epoch)

Epoch Number: 1000

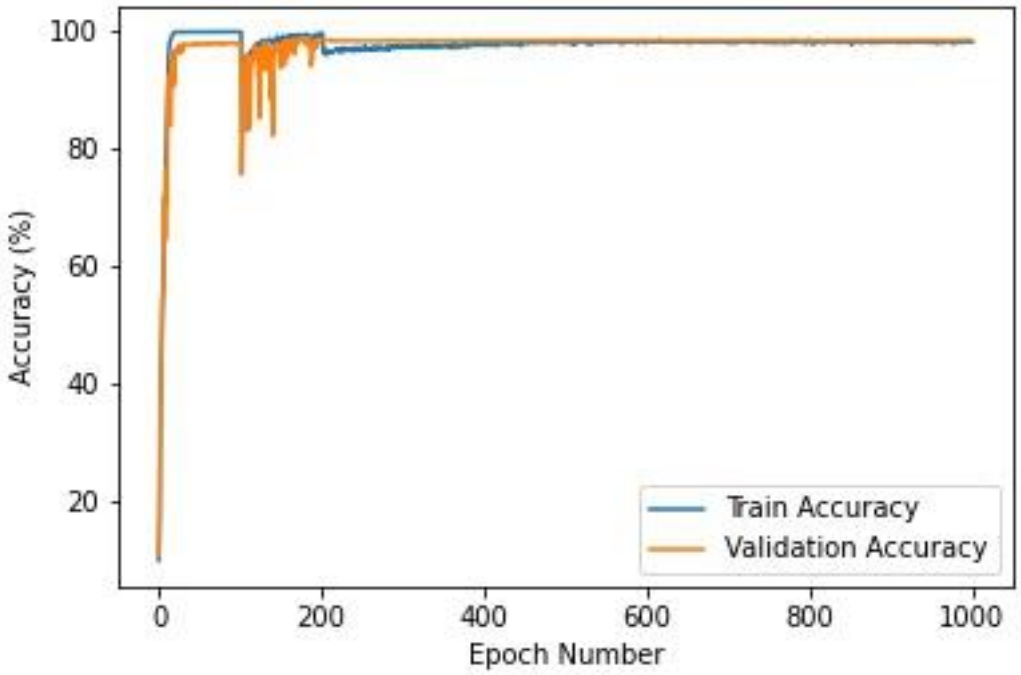
Input size: 32x32x3

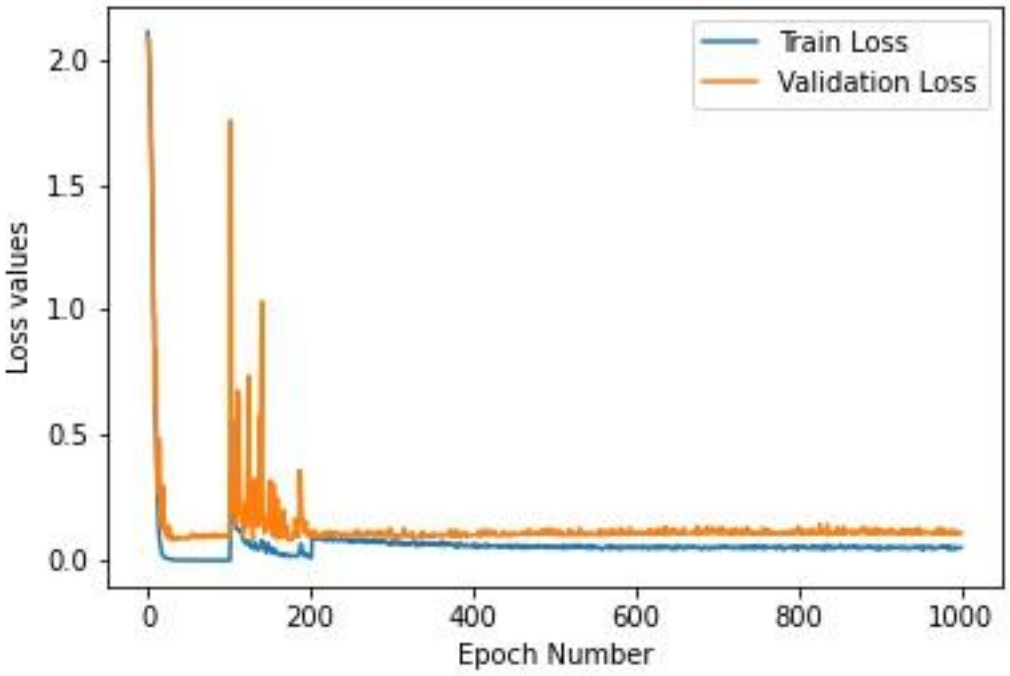
I used CNNNet architecture.

In this experiment, I just changed drop-out factor to gain higher validation accuracy than previous experiment.



As you can see, I applied 0.5 drop out at 100. Epoch and loss value increase, and I applied 0.6 drop-out at 200. Epoch and loss number didn’t decrase as before.





Here is accuracy and loss value results for each epoch.

Before apply drop-out I reach %100 train accuracy but when applied 0.5 and 0.6 drop-out, train accuracy didn't reach %100 because of under fitting.

And validation accuracy and loss stay the same.

****

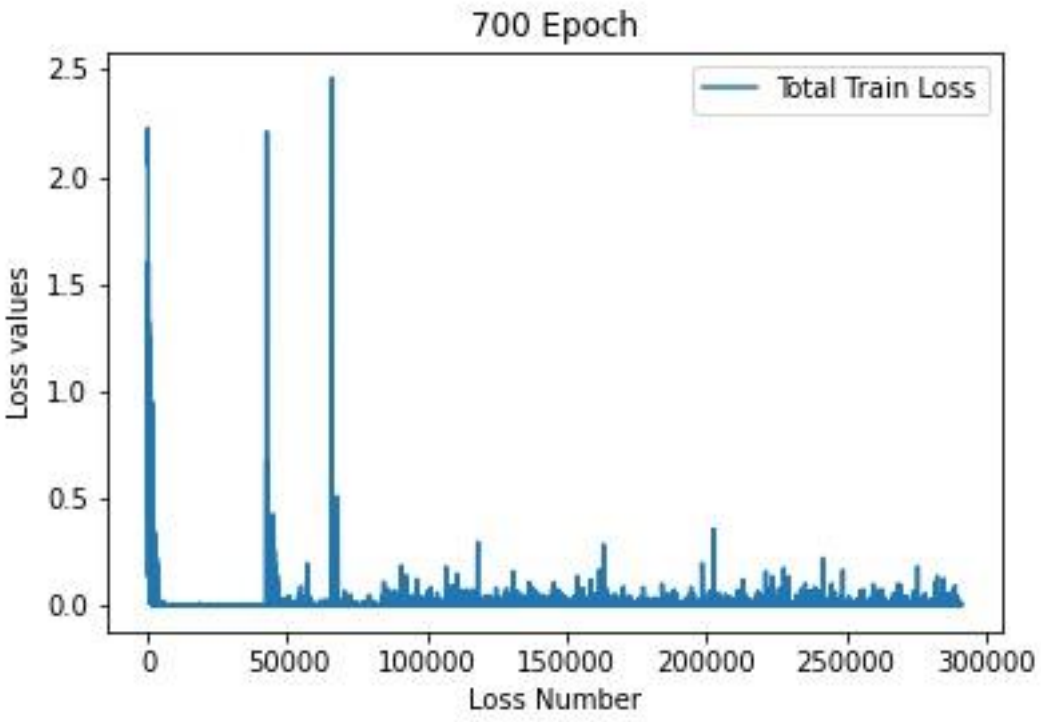
Batch size: 128

Learning rate: 0.001 -> 0.0001 -> 0.00001 (for 200. , 300. and 500. epoch)

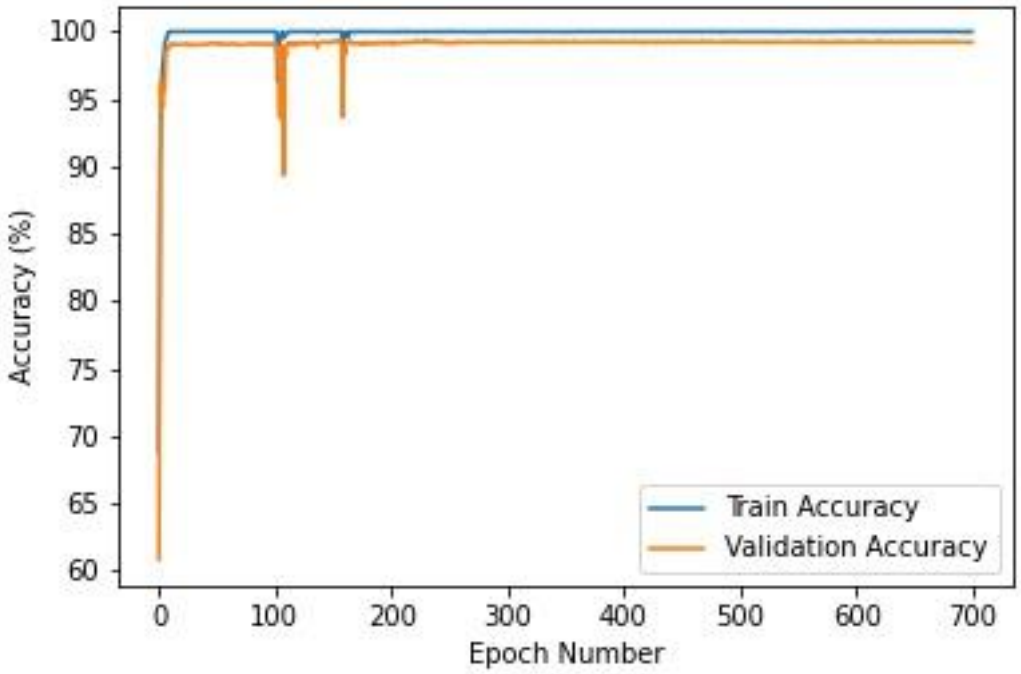
Epoch Number: 700

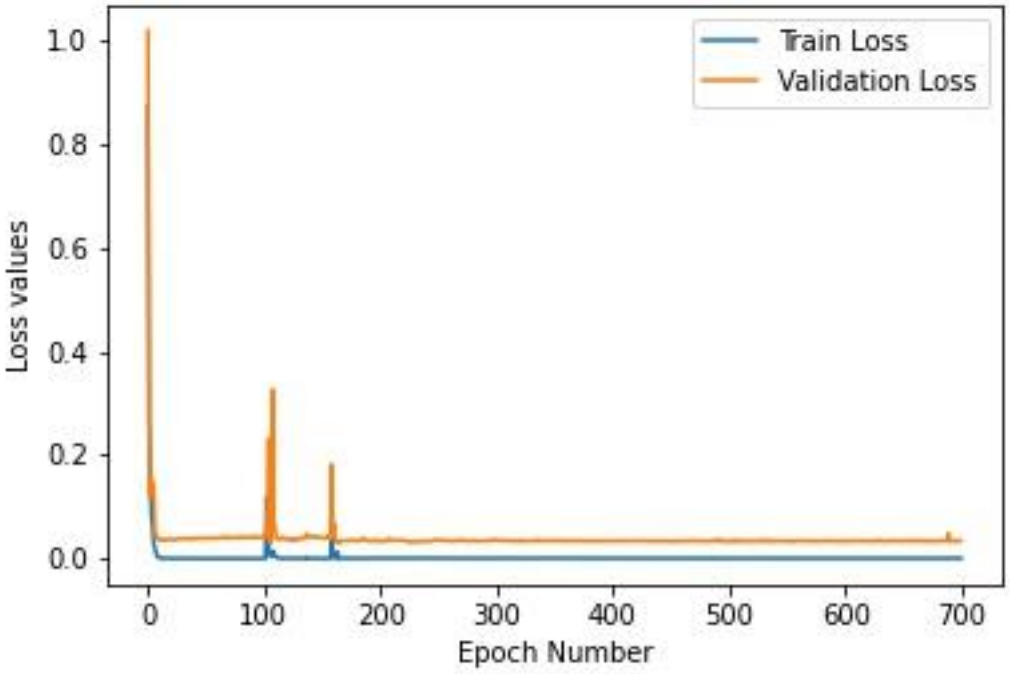
Input size: 32x32x3

In this experiment I used different architecture (CNNNet2).



As you can see, I reached very low loss number and i applied drop-out to model.





Here are train and validation accuracies.

I reached %100 train accuracy and %98 validation accuracy.

This is acceptable but my previous model give more validation accuracy.

So I applied different architectures and hyper parameters.

As you can see, I reached max %99.4 validation accuracy so this is my best architecture therebetween.

I calculated test set accuracy.

```
99.32310469314079
```

```
[16] y_true = np.load('drive/My Drive/conf_true.npy', allow_pickle=True)
     y_pred = np.load('drive/My Drive/conf_pred.npy', allow_pickle=True)
     confusion_matrix(y_true, y_pred)

array([[201,   0,   0,   0,   5,   0,   0,   0],
       [  0, 199,   0,   0,   0,   3,   0,   0],
       [  0,   0, 202,   0,   0,   0,   1,   0],
       [  0,   0,   0, 231,   0,   0,   0,   2],
       [  1,   0,   0,   0, 351,   0,   0,   0],
       [  0,   1,   0,   0,   0, 354,   0,   0],
       [  0,   0,   1,   0,   0,   0, 319,   0],
       [  0,   0,   0,   1,   0,   0,   0, 344]])
```

Here is confusion matrix for this architecture.

When I looked at confusion matrix, I saw our model didn’t classify completely gender type, other classes learned well.

```
classes = ('male dorsal left', 'male dorsal right', 'male palmar left', 'male palmar right',
           'female dorsal left', 'female dorsal right', 'female palmar left', 'female palmar right')
```

My class ranking showed abow.

For example 5. Column and 1. Row value is 5.

This means my model predicted 5 values male dorsal left but actually these are famale dorsal left.

Part II

In this part of assignment, I used pretrained resnet18 architecture.

Many resourches devoloping different architectures and they uses very big data sets and they have maybe very powerfull hardware support, They spend a lot of time to train these architectures and analyze them. For this, we can apply our data sets this pretrained architecture to solve our problem.

Transfer learning actually do this.

We take this architectures and freeze some layers depending on our data sets and adjust our final fc layer for our classification number.

I will show you my different results in resnet18 below.

****

Batch size: 64

Learning rate: 0.001 (and divide by 10 for each 100 epoch)

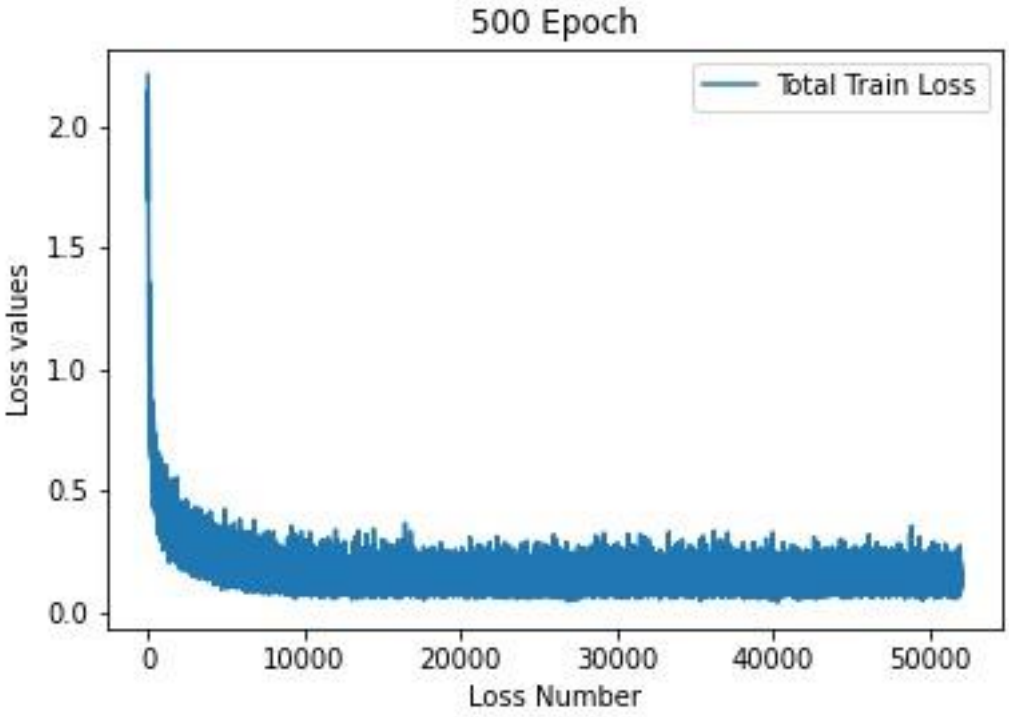
Epoch Number: 500

Input size: 256x256x3

In this experiments, I applied same transform for my dataset.

```
transform = transforms.Compose(
    [transforms.ToPILImage(),
     transforms.Resize(256),
     transforms.CenterCrop(224),
     transforms.ToTensor(),
     transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])
```

When I searched many people uses same transforms for resnet18, so i used them as well.

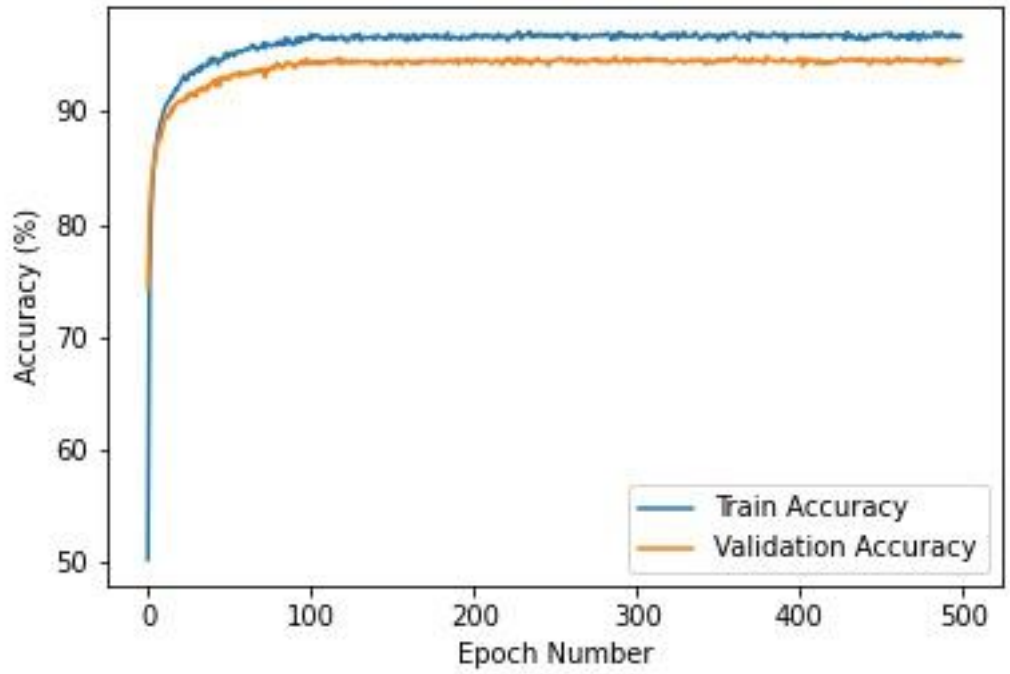


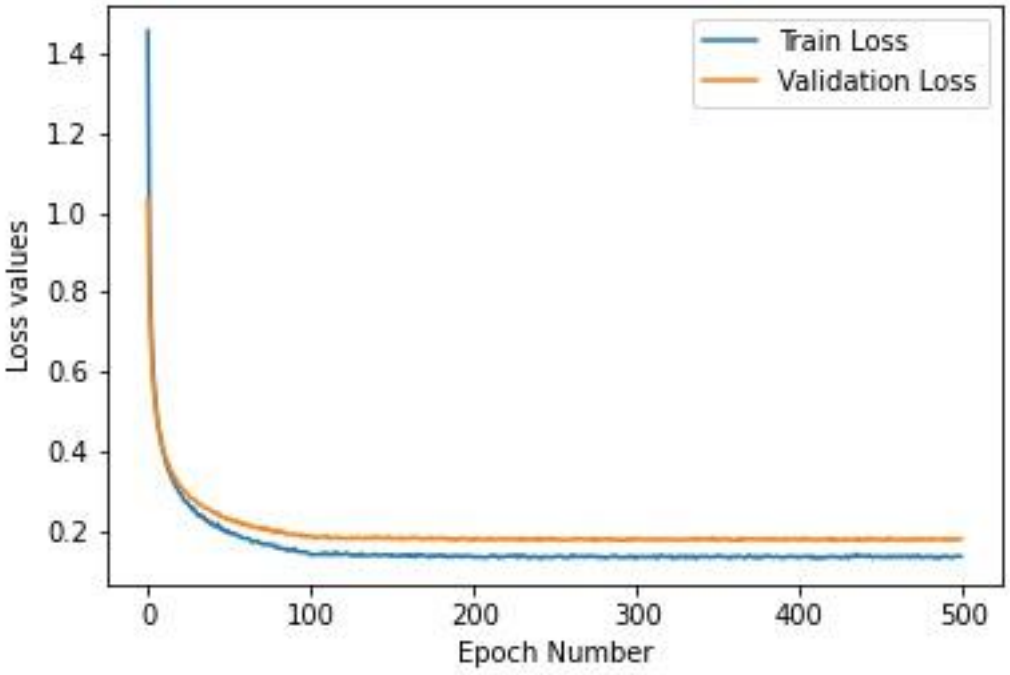
In this experiment, I freezed all layers except final layer and change them 500x1000 to 500x8.

I trained just last fc layer.

As you can see, loss function decrase but i couldn’t get lower loss values.

This architecture suffered from under fitting.





Here are train and validation plots.

As you can see, I couldn't reach %100 train accuracy because of under fitting.

Also there is difference between train and validation accuracy because of over fitting.

****

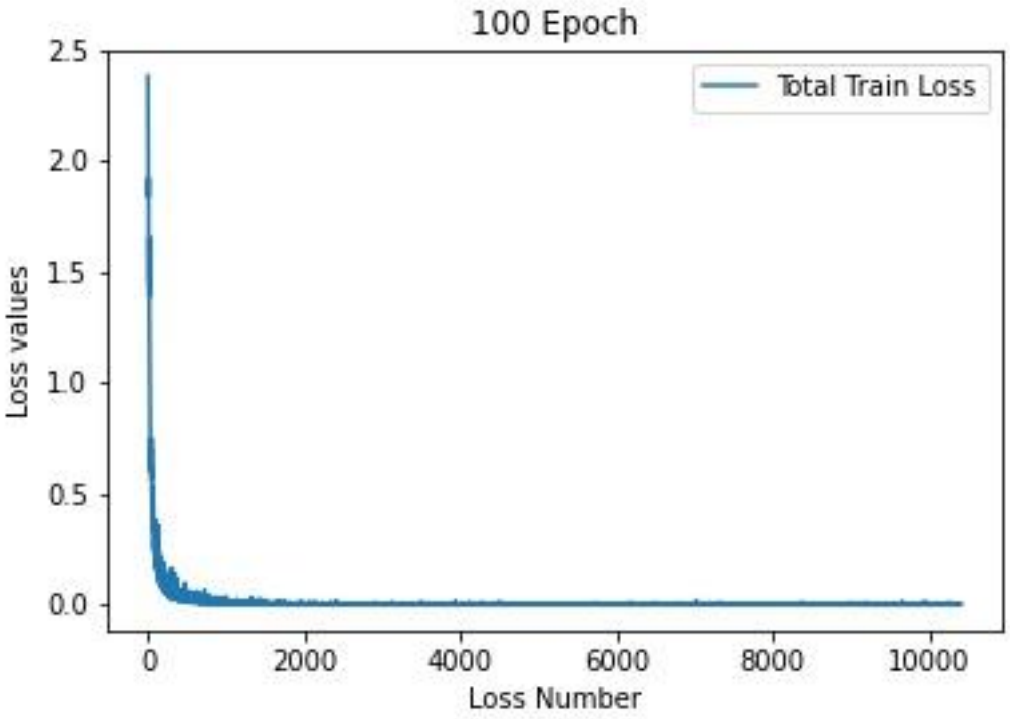
Batch size: 64

Learning rate: 0.001 (and divide by 10 for each 30 epoch)

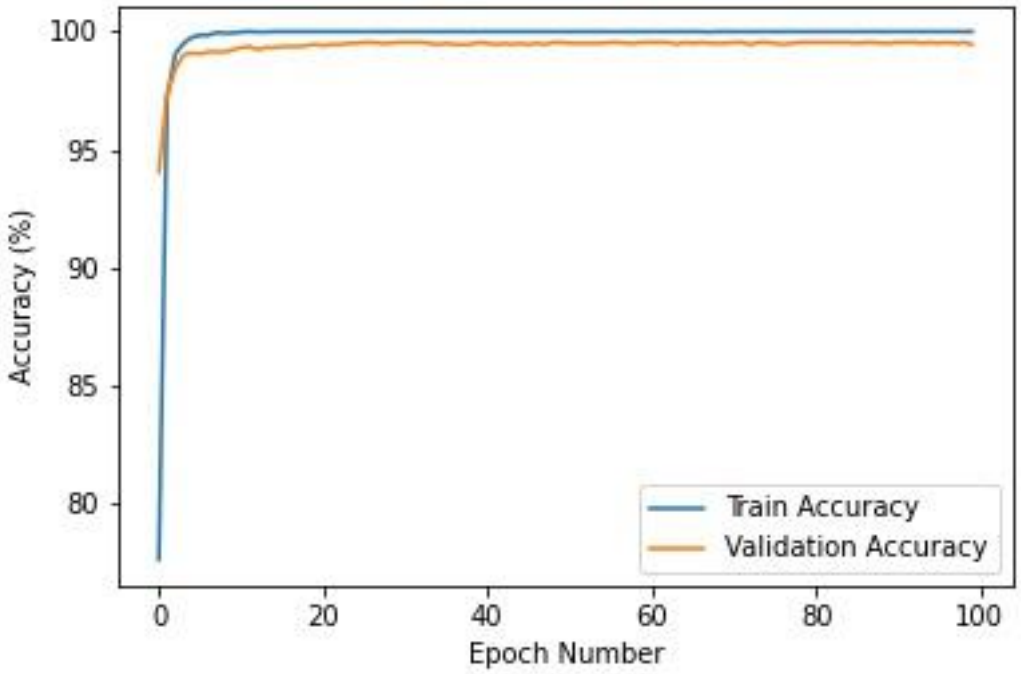
Epoch Number: 100

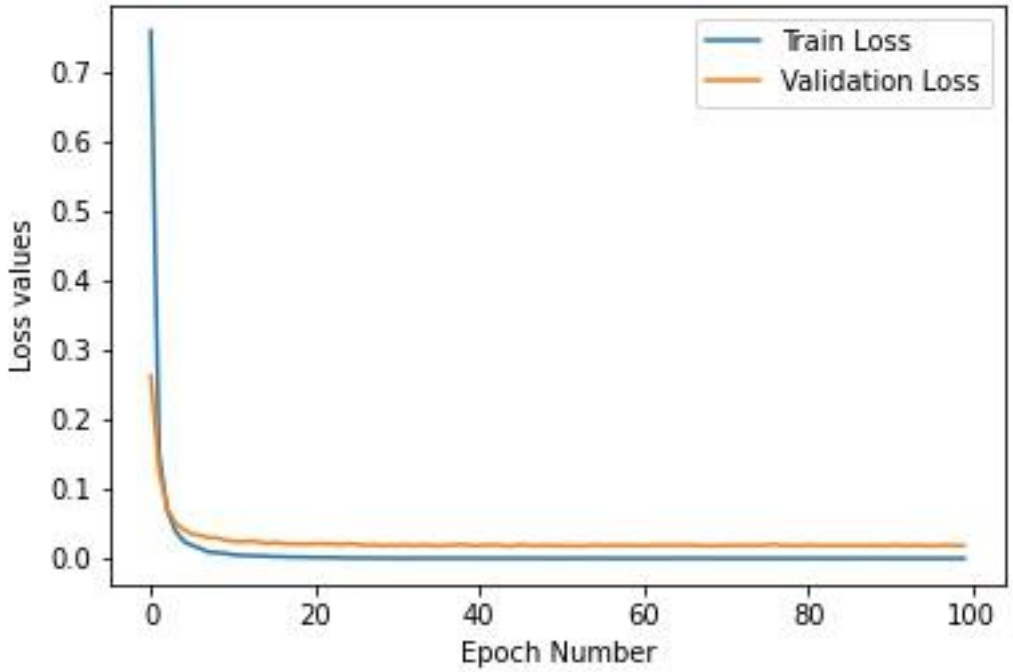
Input size: 256x256x3

In this exprement, I trained fc layer and last sequantial block and freezed rest.



As you can see, loss values more decrased than previous experiment.





Here are train and validation plots.

I reached %100 train accuracy and %99.5 validation accuracy.

This architecture trained well. (No suffering over fitting or under fitting)

****

Batch size: 64

Learning rate: 0.001 (and divide by 10 for each 30 epoch)

Epoch Number: 100

Input size: 256x256x3

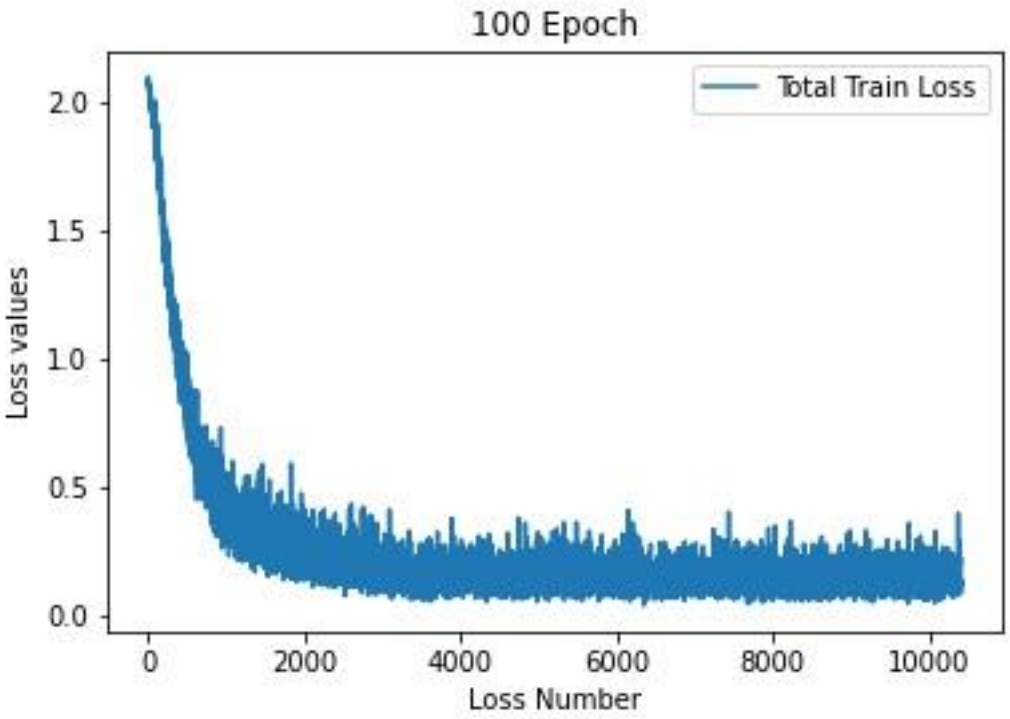
In this experiment, I added more fc layers to top of the architecture.

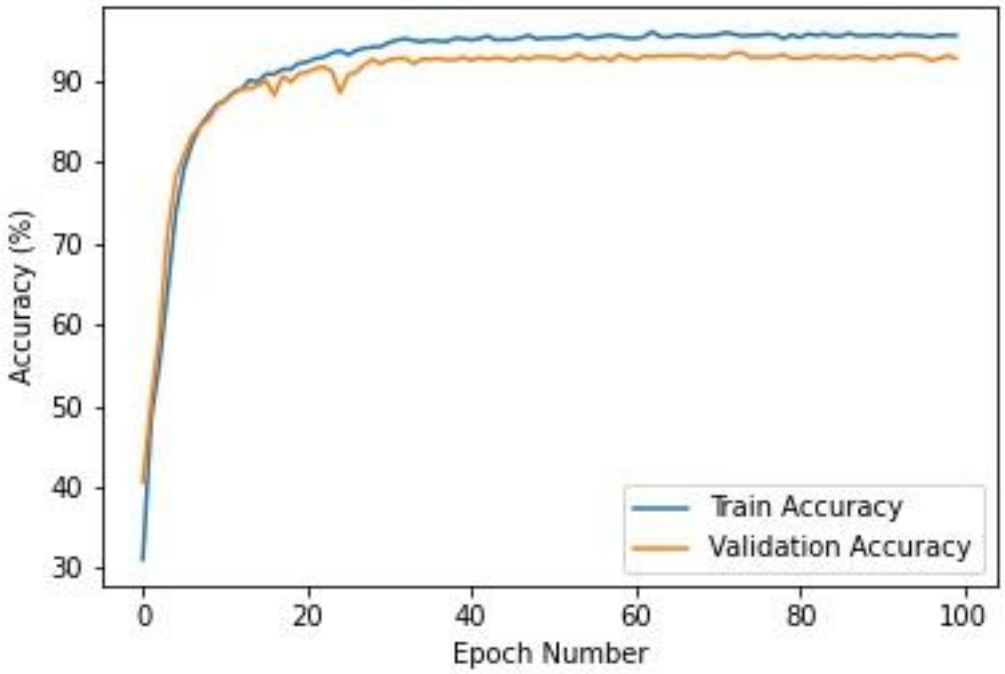
Here is detail:

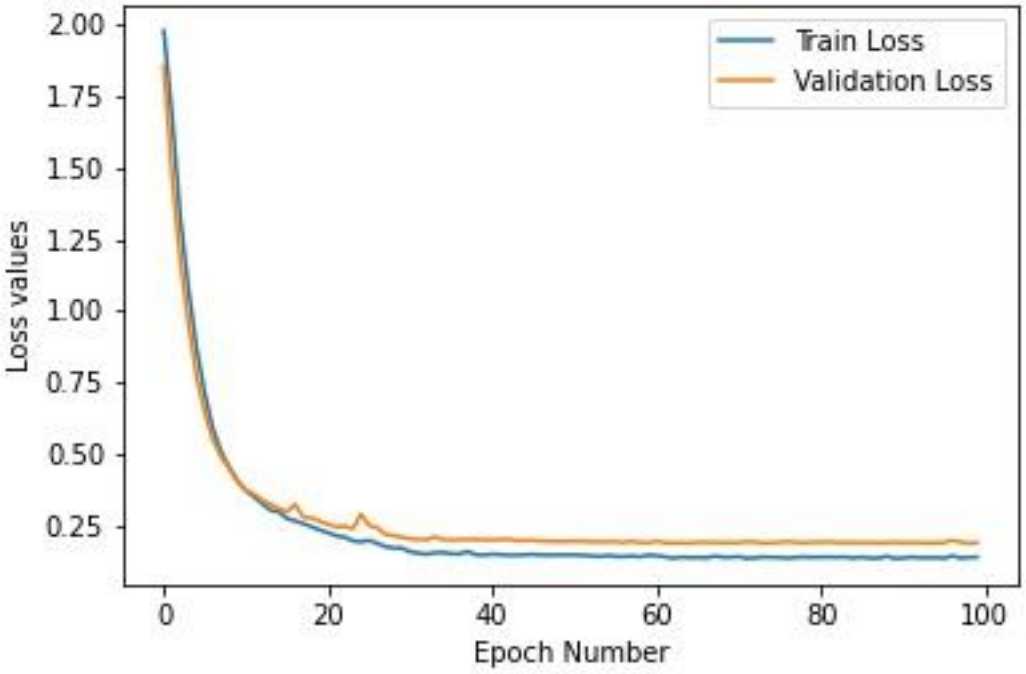
From 500x1000

To 500x500 -> relu -> 500x64 -> relu -> 64x8.

Here are results:







As you can see, Train accuracy couldn’t reach %100 accuracy and validation accuracy stay approxiametly %93.

So this model sufferd model under fitting but i am not sure for this comment maybe i can reach %100 accuracy with more training or get more data.

****

Batch size: 64

Learning rate: 0.001 (and divide by 10 for each 30 epoch)

Epoch Number: 100

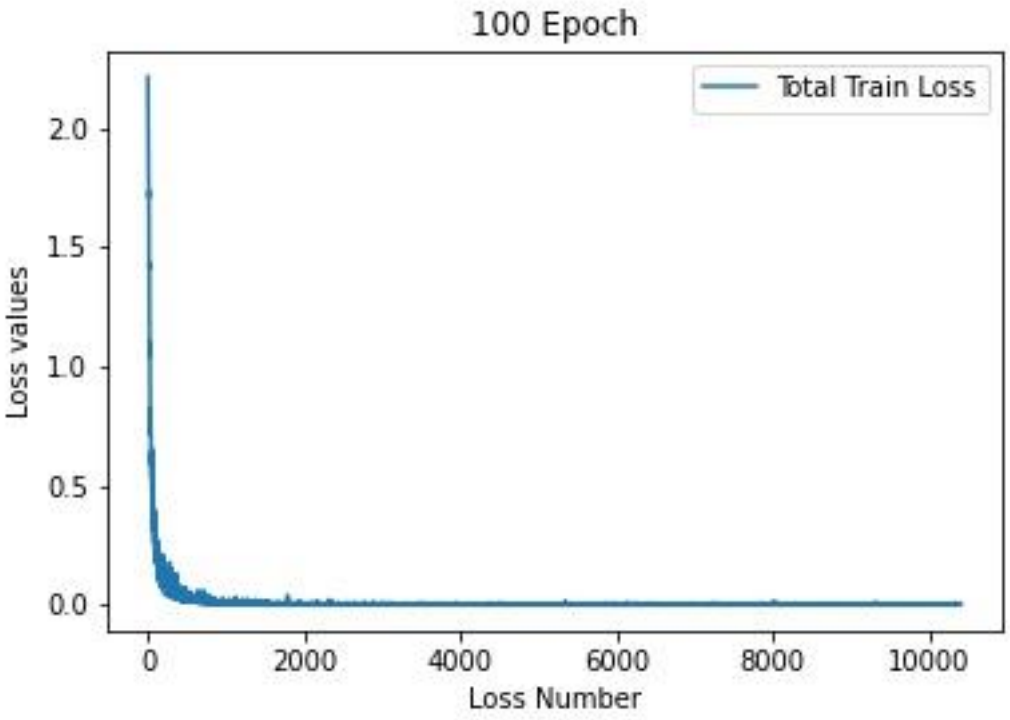
Input size: 256x256x3

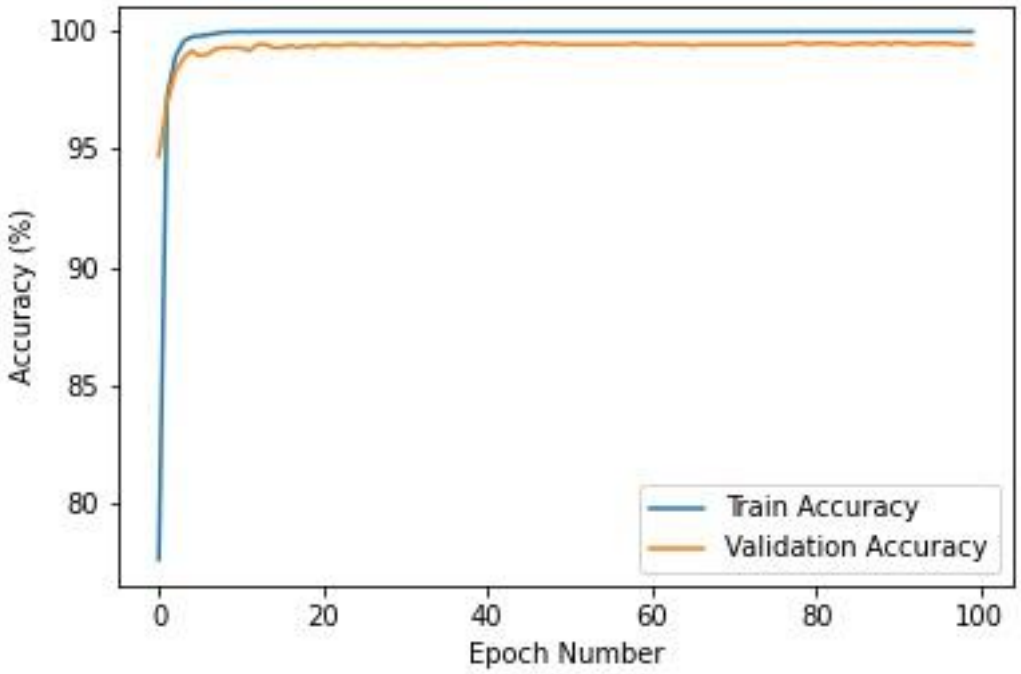
I saw adding more additional fc layer didn't work well (Of course this is my idea).

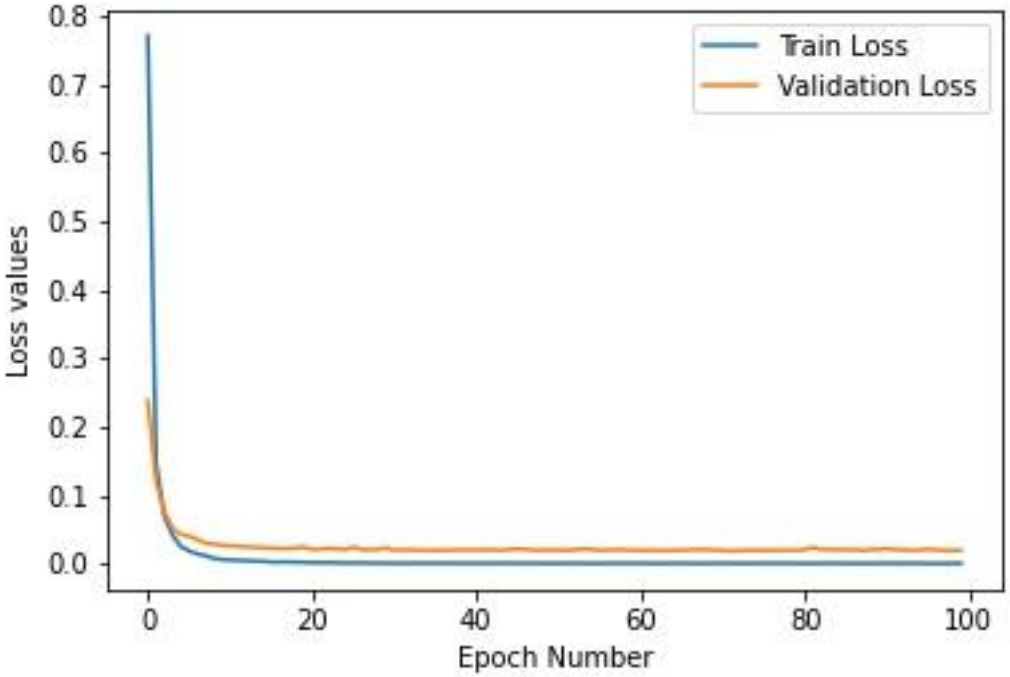
So I freezed less conv layer from 2. Experiment to see what is happening.

I trained fc layer and last 2 sequantial layer.

Here are results:







This experiment gives very close result with 2. Experiment.

I reached %100 train accuracy and %99.4 validation accuracy.

This model trained well and gives good results.

I did different experiments and I reached max %99.5 validation accuracy.

For this experiment, I calculated test set accuracy.

```
99.50361010830325
```

As you can see above, I reached %99.5 train accuracy.

```
array([[200,   0,   0,   0,   6,   0,   0,   0],
       [  0, 199,   0,   0,   0,   3,   0,   0],
       [  0,   0, 203,   0,   0,   0,   0,   0],
       [  0,   0,   0, 231,   0,   0,   0,   2],
       [  0,   0,   0,   0, 352,   0,   0,   0],
       [  0,   0,   0,   0,   0, 355,   0,   0],
       [  0,   0,   0,   0,   0,   0, 320,   0],
       [  0,   0,   0,   0,   0,   0,   0, 345]])
```

I calculated confusion matrix fort his experiment.

I just failed to classify gender type like part I.

Other classes learned well.

So I can say freezing more and less layer (conv or fc) depending on data set can improve our performance and normalizing image before training help to reach gradiant global minumum.

I can observe transfer learning is powerful technic to train different data set.

I reached more accurate test result in part2.

I think the reason of this, Pretrained architectures are more complex architecture and they traind with huge data a long time.

So If we have less data or less computational power, we can use this pre trained architecture to get better results.

Also this architectures have been tried so many times.

CONCLUSİON

I tried many different architectures.

I can observe that, This is a process that consist of choose hyper parameters and architecture, encoding them and getting results.

This process is a cycle.

We have to repeat this cycle until get good results.

I didn’t apply softmax function because CrossEntropy class apply itself.

I can apply batch normalization to fc layers to improve training time.

Also I can use Adam optimizer to improve training time.