PA 5 -Creating CNN model for ASL

Gulam Mohib Parihar
School of Engineering Science
University at Buffalo
Buffalo, New York
gulammoh@buffalo.edu

Nabeel Khan
School of Engineering Science
University at Buffalo
Buffalo, New York
nkhan6@buffalo.edu

Vijaya Rana
School of Engineering Science
University at Buffalo
Buffalo, New York
vrana3@buffalo.edu

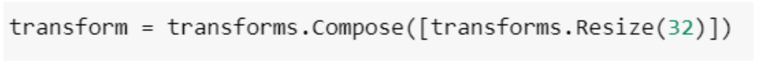
Abstract

ASL word is performed by an ASL signer, based on the acoustic images provided in the dataset, using Convolutional neural network as the classifier

1 Dataset

This dataset includes 10 ASL words performed by 5 subjects. In this dataset, all images are generated by using the short-time Fourier transform (STFT) to calculate a spectrogram as the feature representation of the reflected near-ultrasound waves. This dataset has a training set of 5,000 examples, and a test set of 1,000 examples.

For our model we transformed our images by 32*32 .This reduced our training time a lot .

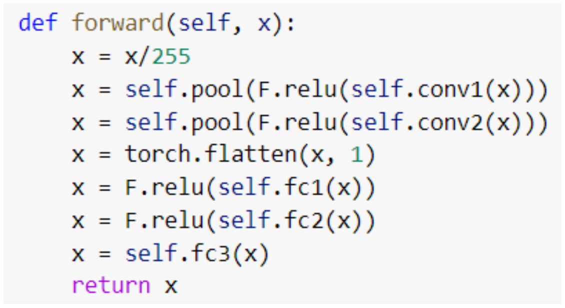


```
transform = transforms.Compose([transforms.Resize(32)])
```

We used a custom image loader, which automatically read images from the pictures folder.

2 Building Neural Network

In this part, we build a Convolutional Neural Network with 2 hidden layers, 2 convolutional layer and a max pool followed by each convolutional layer. As shown below

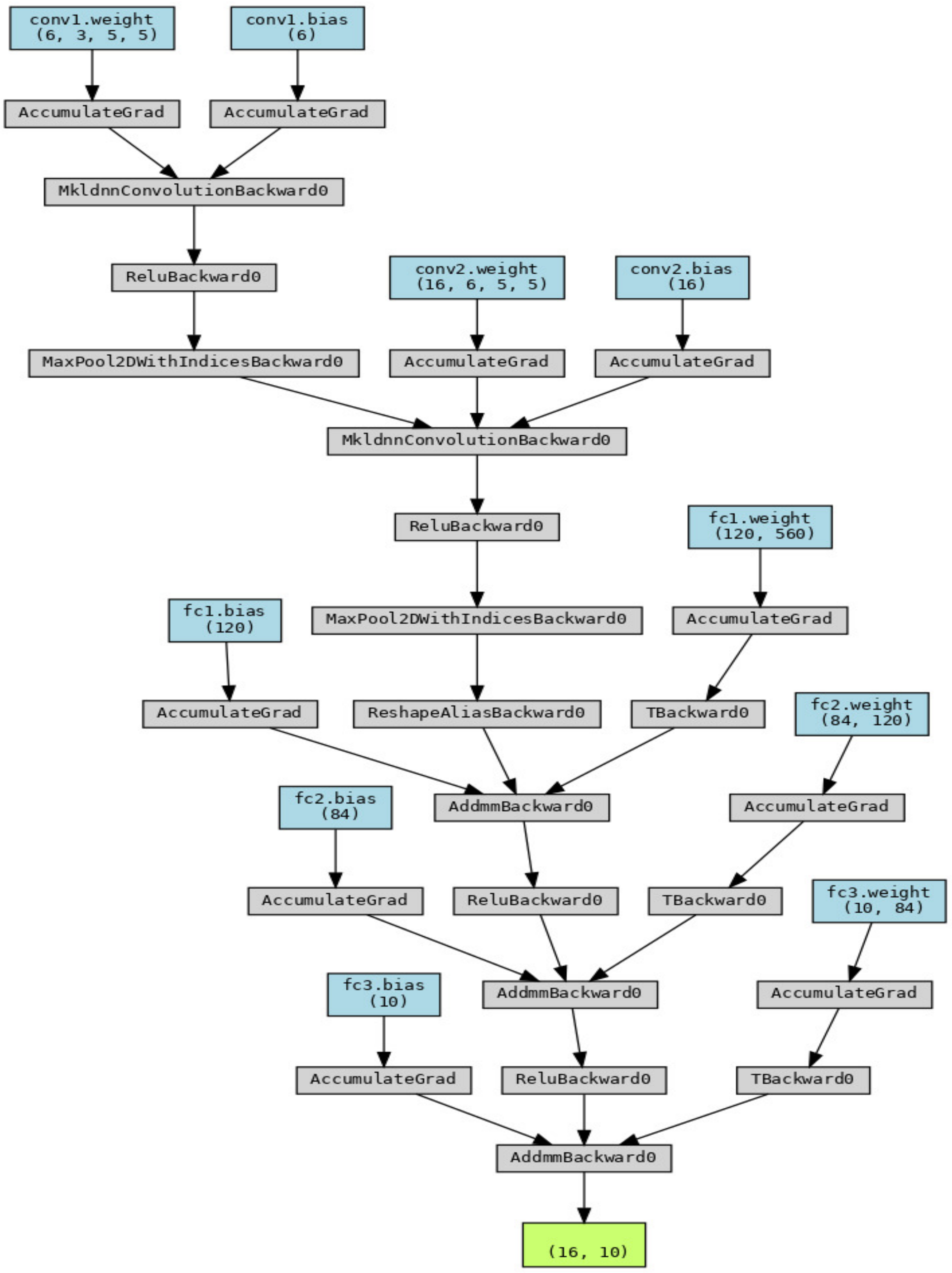


```
def forward(self, x):
    x = x/255
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = torch.flatten(x, 1)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

The dimensions of all the layers used are shown below

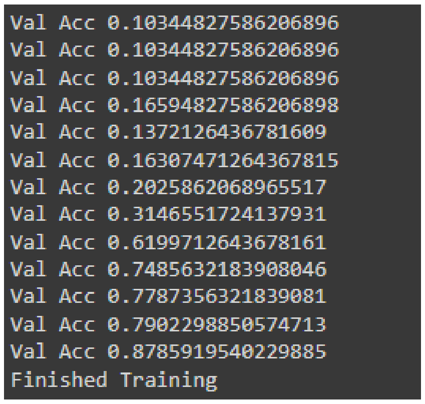
```
self.conv1 = nn.Conv2d(3, 6, 5)
self.pool = nn.MaxPool2d(2, 2)
self.conv2 = nn.Conv2d(6, 16, 5)
self.fc1 = nn.Linear(560, 120)
self.fc2 = nn.Linear(120, 84)
self.fc3 = nn.Linear(84, 10)
```

Stucture of the neural net

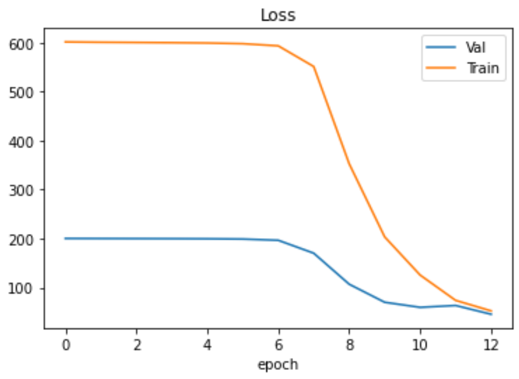


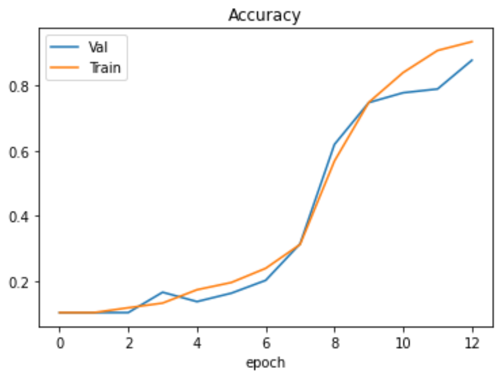
2.1 Training the Neural Network

On training our aforementioned neural network we were able to achieve validation accuracy of 88% in 13 epochs. We used early stopping, and stopped training once we had >85% validation accuracy.



```
Val Acc 0.10344827586206896
Val Acc 0.10344827586206896
Val Acc 0.10344827586206896
Val Acc 0.16594827586206898
Val Acc 0.1372126436781609
Val Acc 0.16307471264367815
Val Acc 0.2025862068965517
Val Acc 0.3146551724137931
Val Acc 0.6199712643678161
Val Acc 0.7485632183908046
Val Acc 0.7787356321839081
Val Acc 0.7902298850574713
Val Acc 0.8785919540229885
Finished Training
```





From the above graphs we can see that training and validation loss is decreasing while accuracy is increasing.

2.2 Testing the Neural Network

In this part, we test the neural network on the test set and we get test **accuracy** of **80.5%**.

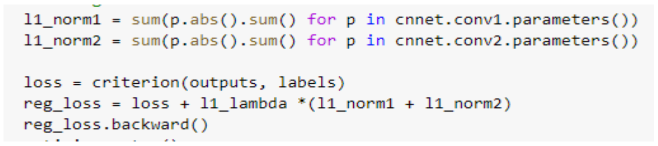
```
0.805316091954023
```

This can be further improved by regularization.

2.3 L1 Regularization

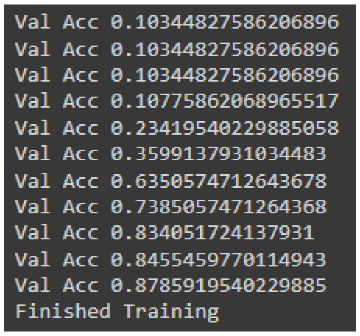
Experimental setup: Here, we added L2 regularization in the loss function by adding the l2 norms of the conv2D layers to the model.

We used early stopping, and stopped training once we had >85% validation accuracy.
We used `l1_lambda = 0.001`



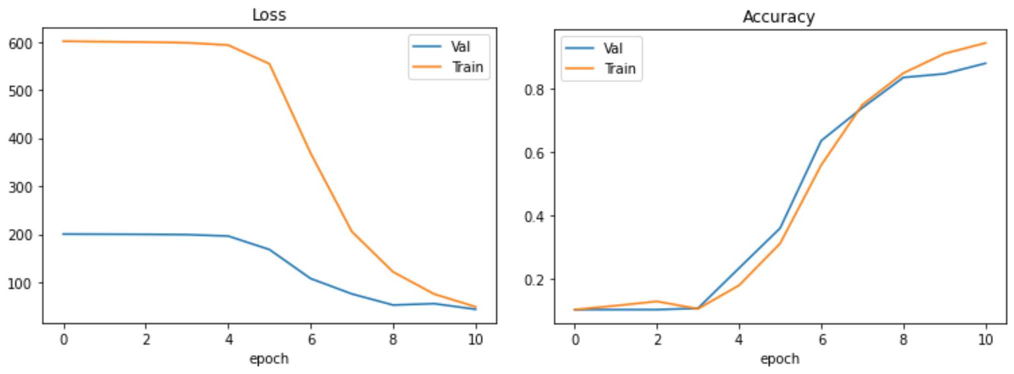
```
l1_norm1 = sum(p.abs().sum() for p in cnnet.conv1.parameters())
l1_norm2 = sum(p.abs().sum() for p in cnnet.conv2.parameters())

loss = criterion(outputs, labels)
reg_loss = loss + l1_lambda *(l1_norm1 + l1_norm2)
reg_loss.backward()
```

We added L1 regularization to our model structure in the Conv2D layer and achieved validation **accuracy** of **88%** in 11 epochs.



```
Val Acc 0.10344827586206896
Val Acc 0.10344827586206896
Val Acc 0.10344827586206896
Val Acc 0.10775862068965517
Val Acc 0.23419540229885058
Val Acc 0.3599137931034483
Val Acc 0.6350574712643678
Val Acc 0.7385057471264368
Val Acc 0.834051724137931
Val Acc 0.8455459770114943
Val Acc 0.8785919540229885
Finished Training
```

Results:
On evaluating this model on the test set we received **accuracy** of: **82%**. This **accuracy** is slightly better than what we achieved without any regularization.



As we can see in the graph, the training and validation loss decreases while the training and validation **accuracy** increases. The validation **accuracy** is slightly worse than the training **accuracy**.

2.4 L2 Regularization

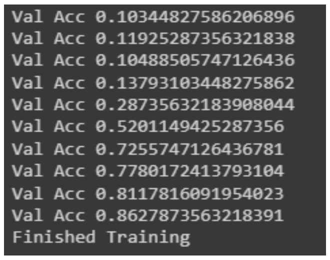
Experimental setup: Here, we added L2 regularization in the loss function by adding the l2 norms of the conv2D layers to the model.

We used early stopping, and stopped training once we had >85% validation accuracy.

We used `l2_lambda = 0.001`

```
# l2 regularization
l2_norm1 = sum(p.pow(2.0).sum() for p in cnnet.conv1.parameters())
l2_norm2 = sum(p.pow(2.0).sum() for p in cnnet.conv2.parameters())

loss = criterion(outputs, labels)
reg_loss = loss + l2_lambda *(l2_norm1 + l2_norm2)
reg_loss.backward()
```

We obtained the validation **accuracy** of **86.2%**.

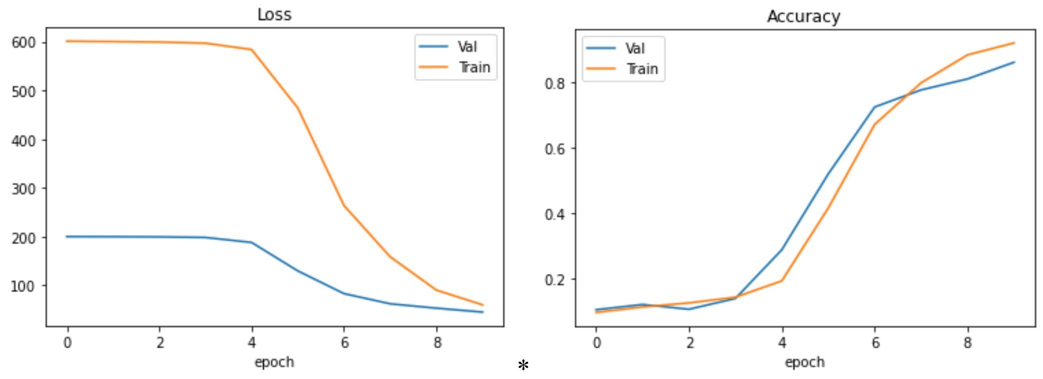


```
Val Acc 0.10344827586206896
Val Acc 0.11925287356321838
Val Acc 0.10488505747126436
Val Acc 0.13793103448275862
Val Acc 0.28735632183908044
Val Acc 0.5201149425287356
Val Acc 0.7255747126436781
Val Acc 0.7780172413793104
Val Acc 0.811781609195402З
Val Acc 0.8627873563218391
Finished Training
```

Results

The test **accuracy** is obtained as **85.05%**. This **accuracy** is better than L1 as well as **accuracy** without any regularization.

As we can see in the graph, the training and validation loss decreases while the training and validation **accuracy** increases.

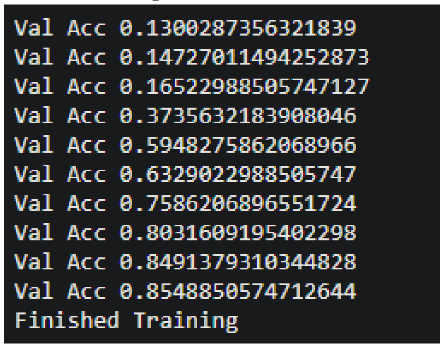


From the graphs and testing **accuracy** of all three we can say that L2 regularization further improves model **accuracy** and performance.

2.5 Dropout Regularization

Experimental setup:

We implemented the dropout regularization by adding a dropout layer just before the output **with 25% dropout**, keeping the model architecture almost same and achieved validation accuracy of **85.5%** in 10 epochs.

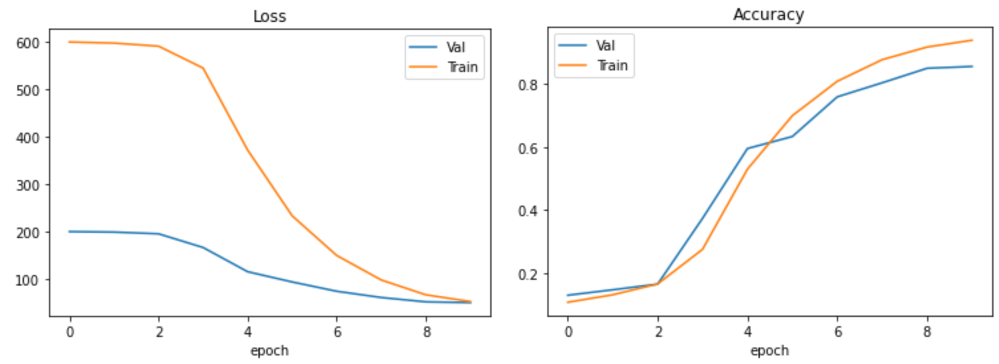


```
Val Acc 0.13002873563218З9
Val Acc 0.14727011494252873
Val Acc 0.16522988505747127
Val Acc 0.3735632183908046
Val Acc 0.5948275862068966
Val Acc 0.6329022988505747
Val Acc 0.7586206896551724
Val Acc 0.8031609195402298
Val Acc 0.8491379310344828
Val Acc 0.8548850574712644
Finished Training
```

Results:

The test **accuracy** we obtained is **84%**. This is better than what we did in section 2.1 and L1 regularization but slightly worse than L2.

Finally plotting the results of Loss vs epoch and **Accuracy** vs epoch



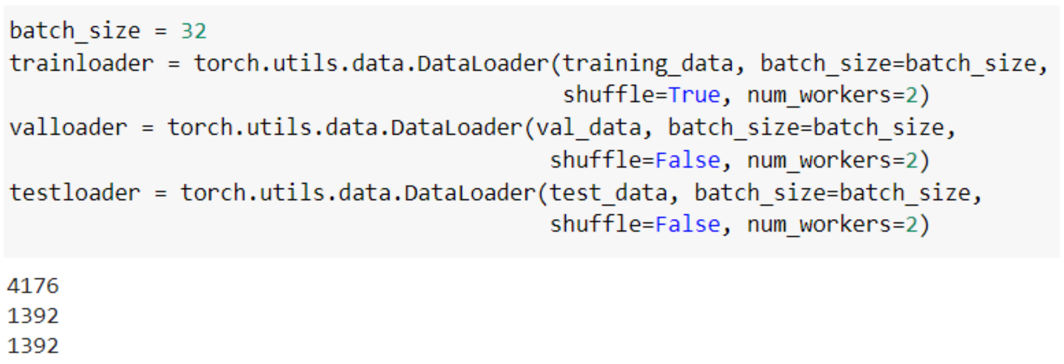
From the graphs, we can see that training and validation loss values fall sharply while the accuracy value increases.

3 Resnet 50

In this part, we loaded the pre-trained model Resnet-50 for the imageNet dataset. We added a trainable fully connected layer (output layer for our use case). We trained the resnet model by freezing a different number of layers each time. Training resnet was quite long so we limited our training to 10 epochs. We also used early stopping and stopped training once we had >.8 validation accuracy.

3.1 Freezing all parameters except last

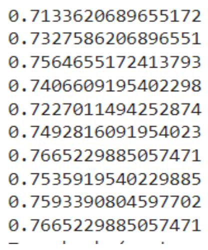
Here we took batch size of 32 and below are the results obtained for trainloader, valloder, testloader



```
batch_size = 32
trainloader = torch.utils.data.DataLoader(training_data, batch_size=batch_size,
                                          shuffle=True, num_workers=2)
valloader = torch.utils.data.DataLoader(val_data, batch_size=batch_size,
                                        shuffle=False, num_workers=2)
testloader = torch.utils.data.DataLoader(test_data, batch_size=batch_size,
                                         shuffle=False, num_workers=2)
```

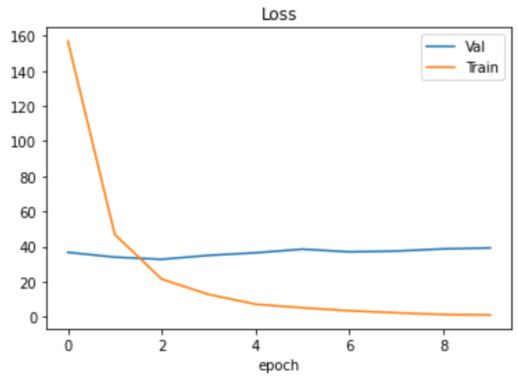
```
4176
1392
1392
```

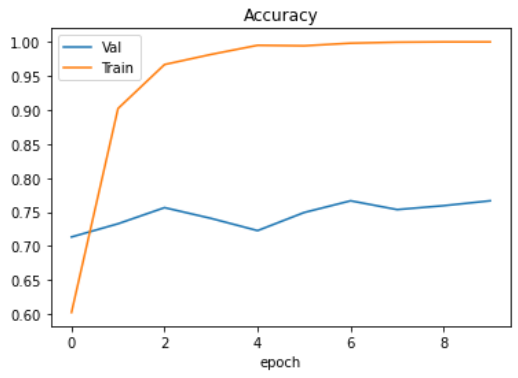
After training for 10 epochs we obtained the results as below we got following validation accuracies



```
0.7133620689655172
0.7327586206896551
0.7564655172413793
0.7406609195402298
0.7227011494252874
0.7492816091954023
0.7665229885057471
0.7535919540229885
0.7593390804597702
0.7665229885057471
```

Finally plotting the results of loss vs epoch and Accuracy vs epoch



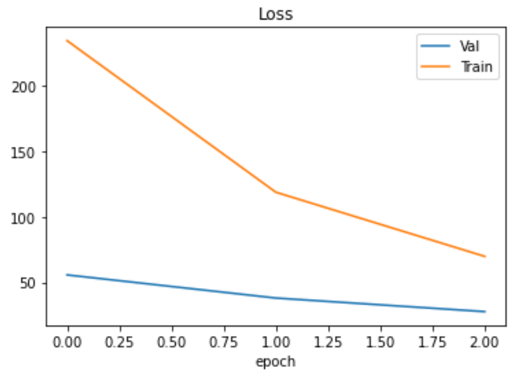


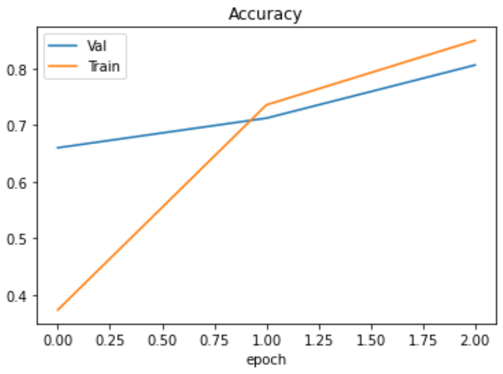
3.2   Freezing all parameters except last 2

Here, we loaded the pre-trained model Resnet-50 for the imageNet dataset .Then we fine tuned the model to fit our dataset of 10 classes and trained it for 50 epochs.We obtained an validation **accuracy** of **80.6%**.

```
0.7126436781609196
0.8067528735632183
```

Below are the plots obtained as a part of Resnet training in part 2



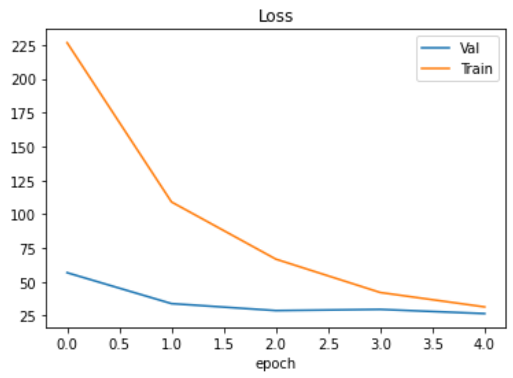


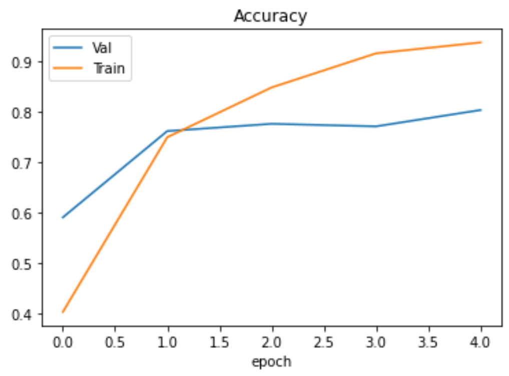
3.3 Freezing all parameters except last 3

Experimental setup: The validation **accuracy** we obtained on validation data is **80.3%** in 5 epochs.

```
0.5912356321839081
0.7622126436781609
0.7765804597701149
0.771551724137931
0.8038793103448276
```

Finally plotting the results of loss vs epoch and **Accuracy** vs epoch





The above plots indicate a different training process with different frozen layers for loss and **accuracy**. Validation **accuracy** tends to be closer to that of training in all except 3.1.