In The name of God

University Tehran

Engineering Facility

Electrical and Computer Engineering

Deep Learning with Applications

HW # 2

Amin Fadaeinedjad

810195442

Spring 99

Abstract

In this Homework we were going to implement neural network using the Pytorch framework this means that there is no need to do the numerical computation by our own we can easily implement layers for the network and after that we are going to train out models for a particular result. The first model is used to detect the position of the human joints by using a small dataset. In the second question of the Homework we used an auto-encoder to detect the fault of an image and we used a hazelnut dataset for this reason and tried different models for this part to see which works best.

Question 1

We are going to use the dataset mentioned containing 2000 images of sport athletes and a matrix file which has 14 joints of the human body and every joint has 2 values one for the x value and the other one is for y value of the image the second thing we need to do is no resize the images because the image dataset has images with different sizes and we cannot give inputs with different sizes to the network because the structure of the network is designed for a certain data size in every layer of the model, so in this case we resize the image dataset there is also a need to change the values of the joints position because that value was computed by the image and by resizing the image we need to change that value as well and indeed we do that as well. According to the paper mentioned as Deep pos we need to normalize the joints so there value will be between -0.5 , 0.5 for both x, y values.

Assume $y = (\dots, y_i^T, \dots)^T, i \in P = \{1,2, \dots, k\}$ and after that we are using a transform assuming $b = (b_c, b_w, b_h), b_c \in \mathbb{R}^2, b_w, b_h \in \mathbb{R}$ and the transform is like $eq1$

$$N(y_i; b) = \begin{pmatrix} \frac{1}{b_w} & 0 \\ 0 & \frac{1}{b_h} \end{pmatrix} (y_i - b_c)$$

Transform from $eq1$ will make the center of the image map at $(0, 0)$ of the joints and the bounds are between -0.5 to 0.5 and the loss function is L2 norm

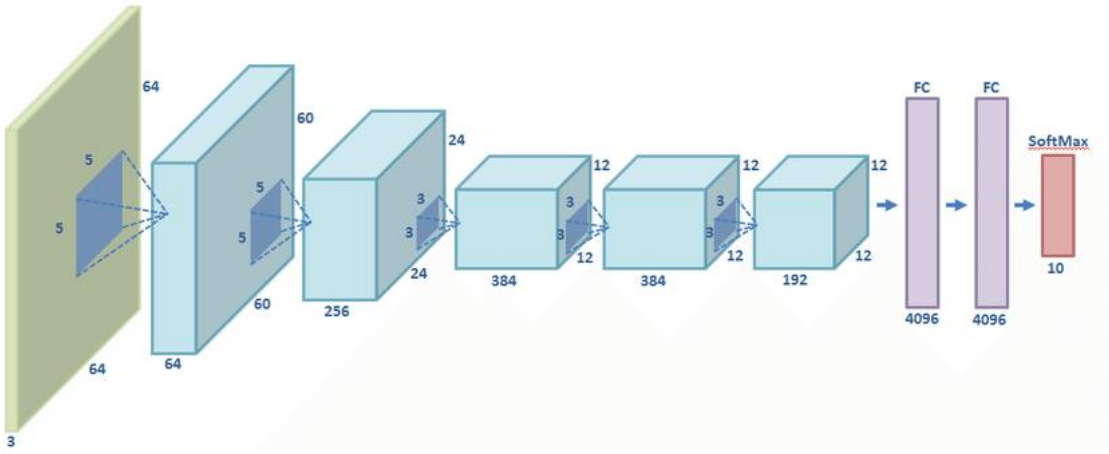

Figure 1 AlexNet

In Figure 1 we can see the AlexNet which the Deep neural network that we are going to use in this question has a similar structure which has a good performance on image data.

We also got help from a Git repository and the link in mentioned in the references but is wasn't used completely only a small part was needed to get the idea about how this DNN works.

Part 1) The first part we designed a DNN as it was mentioned in the paper and after that we used the Adam algorithm for solving the optimization problem of this network.

Adam algorithm is like the following equations assuming that g_t is the gradient of the loss function in the ith step:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

We also need to normalize the two terms:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

And at last in the weight updating:

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Where ϵ is a small number in case the standard deviation is zero.

This results are from using Adam optimizer for 50 epochs with learning rate 0.00005 as said in the paper and we used drop-out as well with the probability of 0.6.

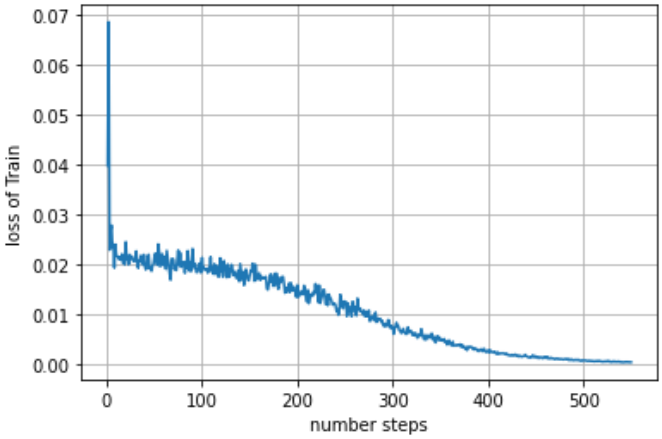

Figure 2

Figure 2 shows the loss for training data as it is expected it decreases by number of step we have about 50 epochs but we are using batches of size 128 so in this case we are going to have $50 \times \left[\frac{1400}{128}\right]$ steps in it.

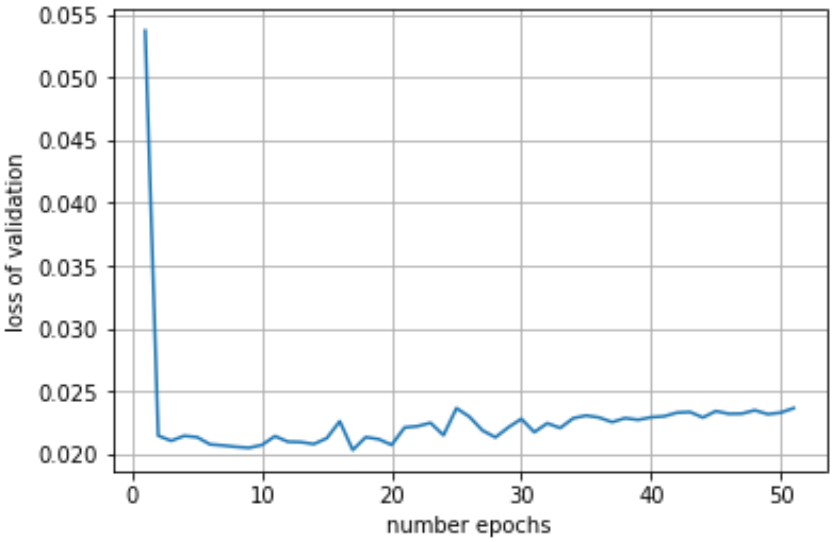

Figure 3

Figure 3 shows the loss for the validation dataset as we can see the result maybe the train loss decreases but the loss for the validation dataset will decrease at first but after a while it will start to increase which with using the early stop algorithm we know that we should stop at that point because the model is memorizing the data and the model is over fitting.

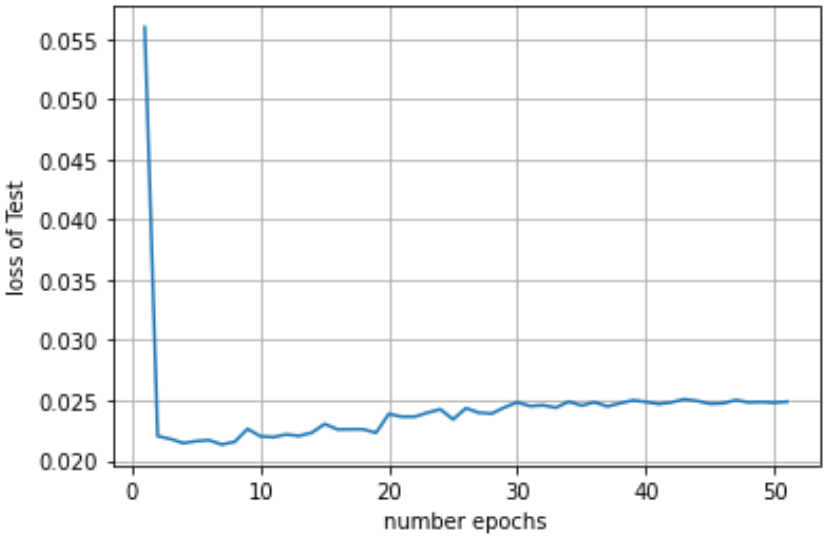

Figure 4

Figure 4 shows the loss for the number of epochs that we have passes and we can see that result for it similar to the loss for the validation dataset so that's one of the reasons that we use early stop algorithm to find the best place to stop training and by the look at the validation loss graph we can predict that the best epoch is about epoch 8 or 9.

The following we are going to show the result of PCP and the PDJ:

The PCP (Percentage of Correct Parts) is this way we are going to find the size of a particular limb and we use the main data for this part and the next part we need to calculate the distance between the true location of the limb and the predicted location of the limb and if it is less than half of the limb true size then we state this prediction as a true and it is more we say it predicted wrong and in this case we have 3 figures to show this parameter.

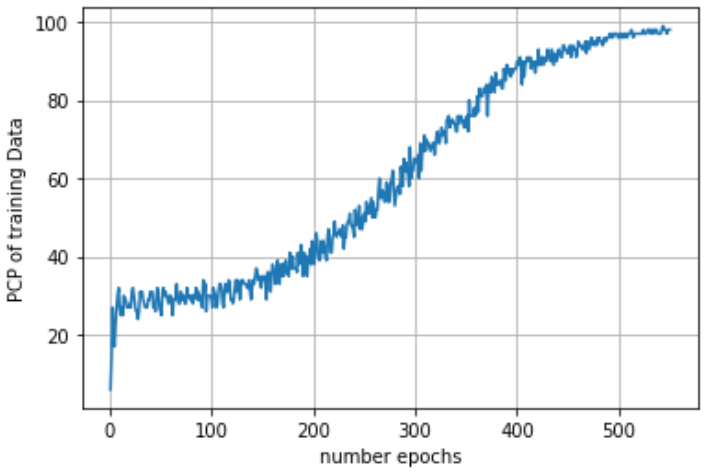

Figure 5

In figure 5 we can see the result for the batches that the parameter is increasing and it is getting near 100% which we can assign as an over fit state.

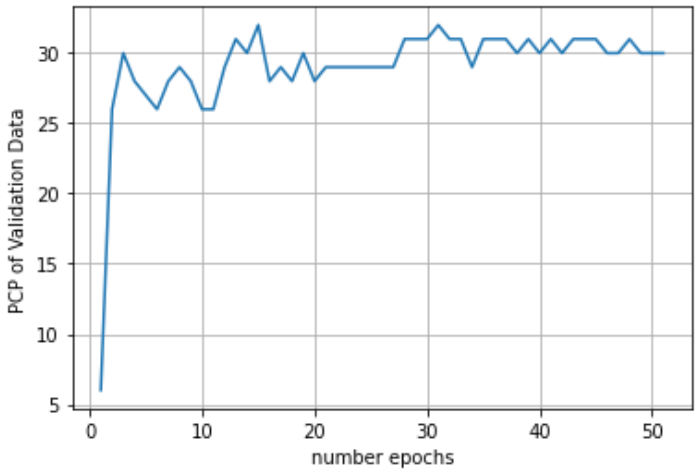

Figure 6

Figure 6 shows the PCP parameter for validation dataset and we can see at most it will reach about 33% and that is not a good number to reach but we can see that it will reach a better point in epochs number 32 and this means that the loss that showed the minimum result is in epoch 8 it does not really matter.

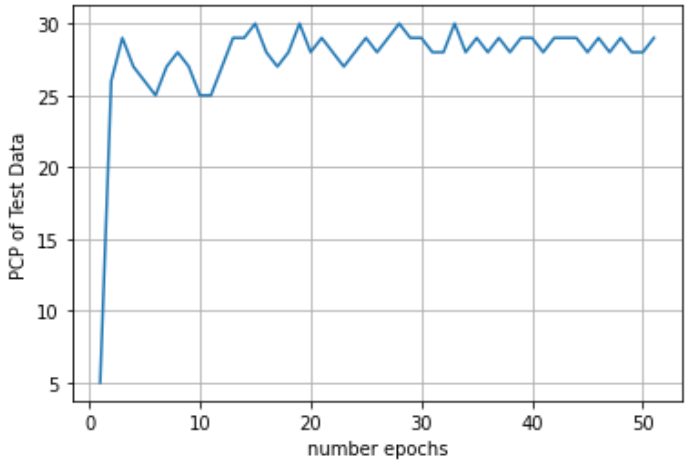

Figure 7

In the figure 7 we can see the same result in the previous part that it will reach a maximum number of 30% accuracy which is not a good number and in this case it is like validation dataset.

The PDJ (Percent of Detected Joints) in this accuracy parameter we need to calculate the torso diameter which according to the paper it is the distance between the true and predicted position of a joint must be in a fraction of torso diameter. The torso diameter is the distance between the right hip and the left shoulder and in this question the fraction that we have chosen for this distance was 0.5

The figures below will show this parameter in the passing epochs.

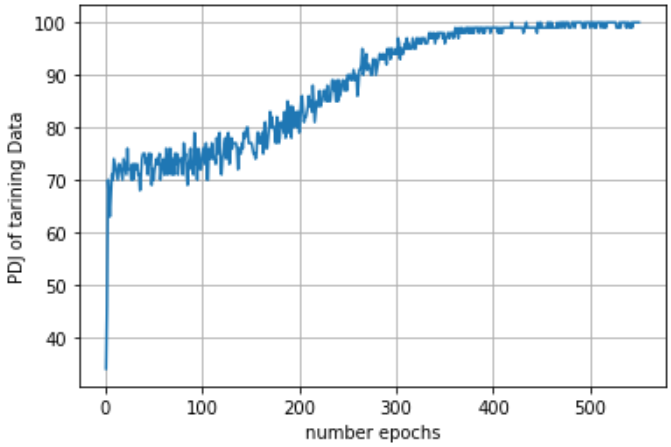

Figure 8

Figure 8 show the PDJ for the training dataset and like the PCP parameter it will increase until it reaches 100% accuracy for this it is obvious that the model is over fitting the reason can be seen in the two validation and test dataset result in computing this parameter.

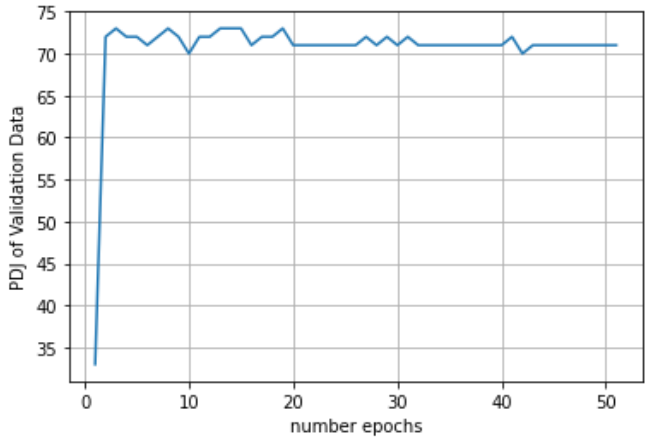

Figure 9

Figure 9 shows the PDJ parameter for the model on the validation dataset and we can see that the PDJ will increase at first but after a while it while it will start to decrease because the model is over fitting to the train data and that is not god

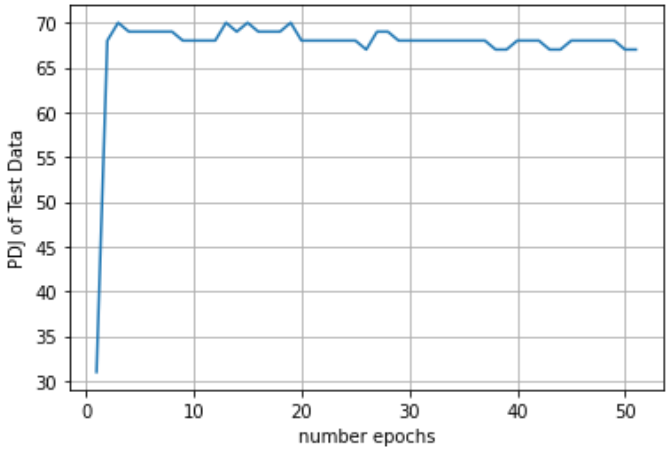

Figure 10

Figure 10 has the same result as figure 9 that the PDJ parameter will decrease at the beginning and after a while it will start to decrease which means is memorizing the train data it is best to see the model in 8^{th} epoch.

Now we are going to show some results for the image and what is happened to it after that we have used the model.

At first we show the Test images of the dataset.

Figure 11 Test Dataset

Figure 1211 Test Dataset

Figures 11 and 12 are the result for the joint detection for the test data as we can see we don't have much accuracy the reason for that is that we don't have much data for our dataset and that means it we train it for a long amount of epochs the network will start to memorize the data instead of trying to learn the pattern of the data and generalizing the images we can see figure 13 and 14 that we gave the train data at have a very good performance that means that the network had memorized the images and result the joints that it already know and that isn't good overall.

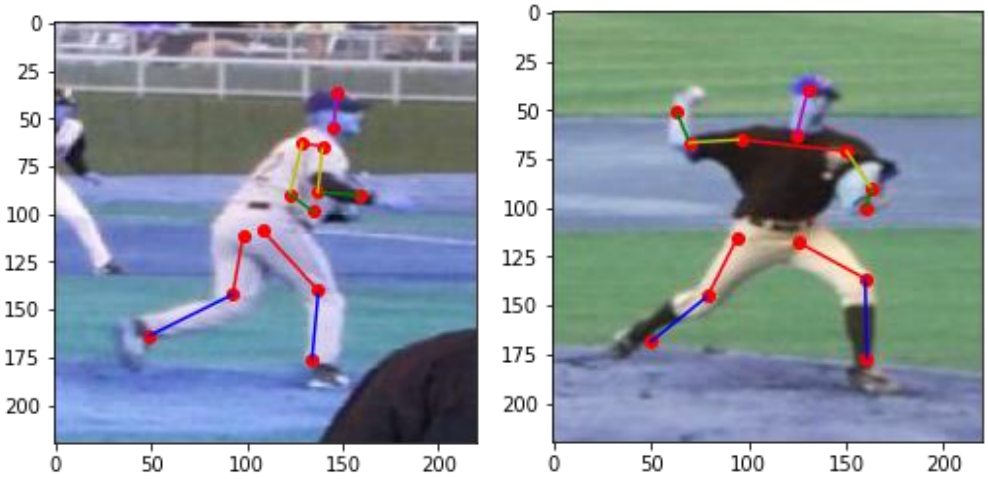

Figure 12

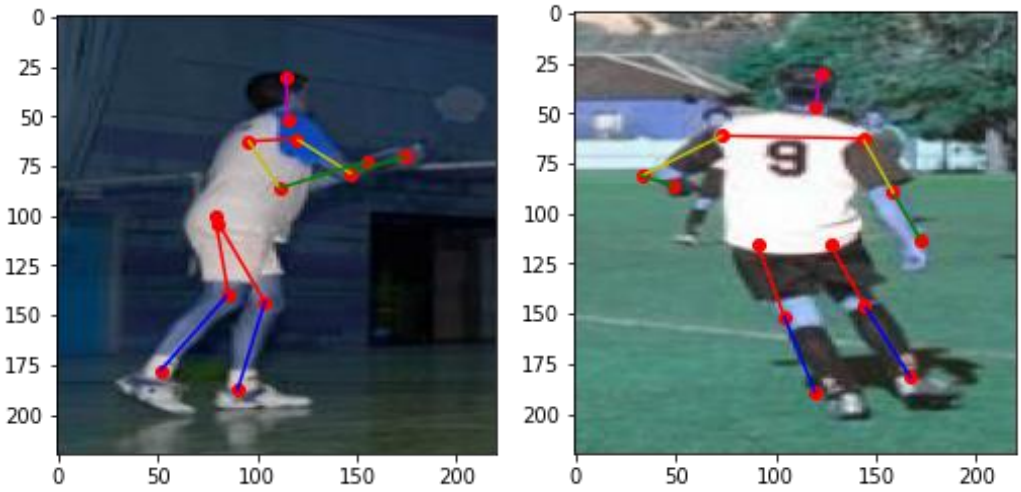

Figure 13 Train Dataset

So as a result it can be figured out that the model with 7 layers of convolutional and fully connected layers have enough parameters to memorize the training dataset and the loss of them and the PCP and PDJ parameters show the same in the graphs.

Part 2) In this part we are going to use Augment data but in our project we were using google Colab and we have a limited cloud memory and we can't augment all data's in the dataset will all the ways possible we used methods like rotation in clock wise and counter clock wise direction and scaling the image and adding noise and blur to the image we need to know that if we add a rotation to the image we need to change the joints as well because in the rotation the location of the joints will change and we are doing this in order to improve the model so in this case we need to change the joints as well but there is no need to change the joints for adding noise or blur to the image in the scaling we need to scale the joints as well, one cool thing we have learned from in dada augmentation is that flipping is not a good option for this augmentation because when you flip an image the position of the joints will not change the way we expect because in the flip operation to change the locations of the joints it really doesn't just flip the joints the picture will still stay the same as they were so in this case we only use rotation and scaling and noise and blur to the image (noise and blur doesn't affect the position of the joints)

We expect to have a better accuracy because we are using more data than before and we are using noise and rotation and scaling in order make this data's

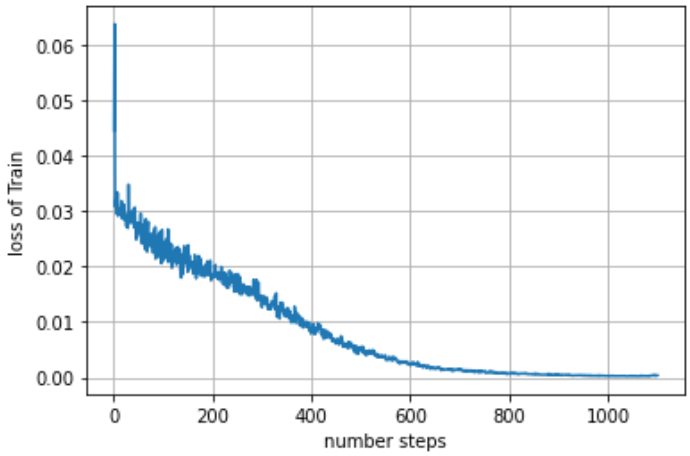

Figure 14

Figure 15 shows the loss for the training data and as we can expect the loss decreases by time but this maybe cause overfitting which we need to investigate in the next part

We can also see in figure 16 and 17 that the loss of the model decreases at first but still after a while the loss starts to increase and yes that is where the model starts to over fit it is in a higher number of epochs but still it happens and we can still see in the results

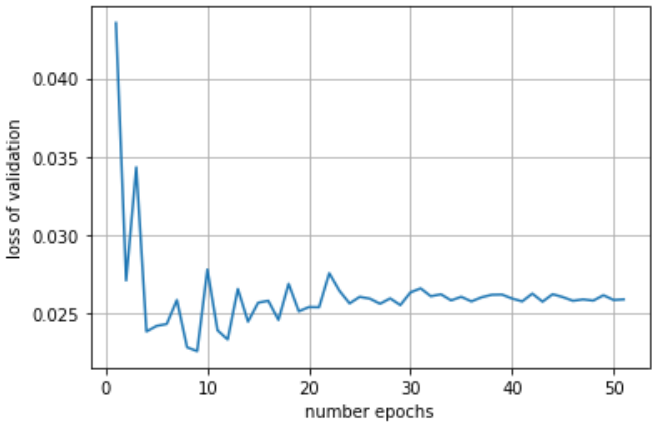

Figure 15

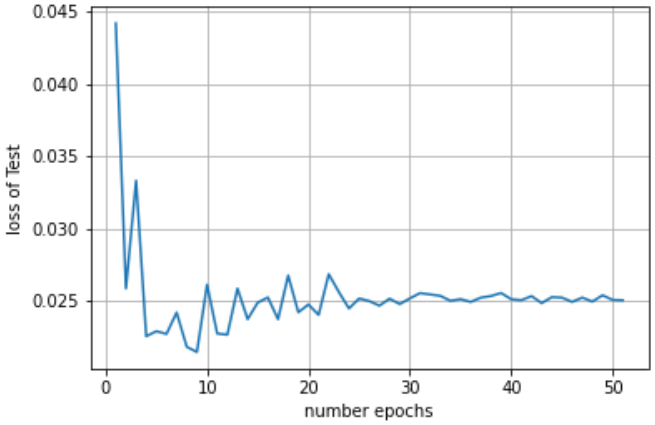

Figure 16

In the next step we are going to plot the PCP parameter for this model for the 3 dataset train validation, test

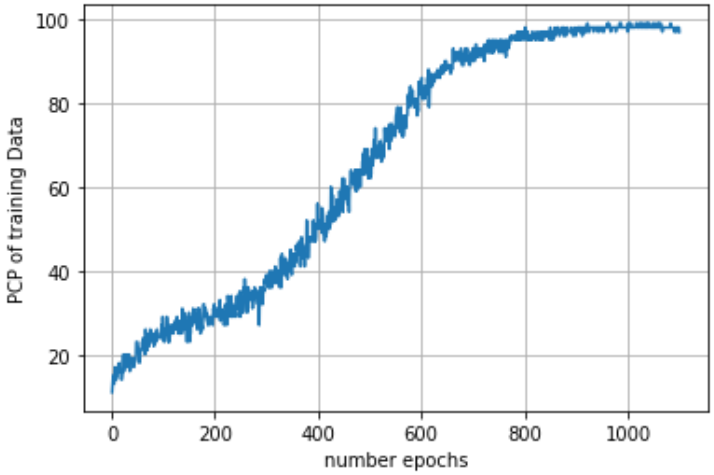

Figure 17

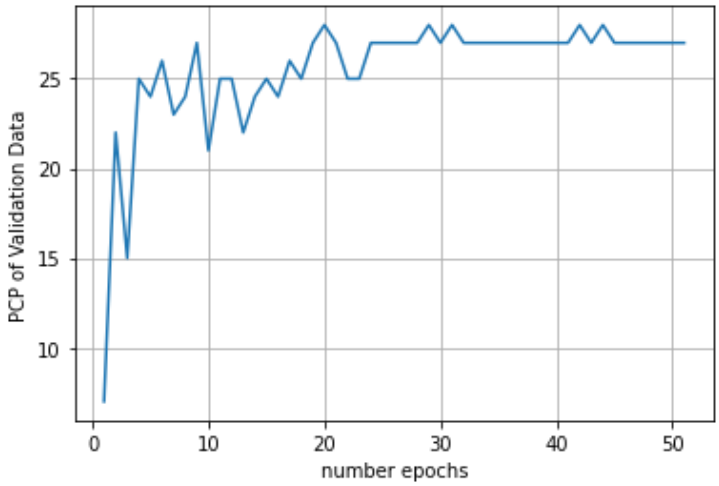

Figure 18

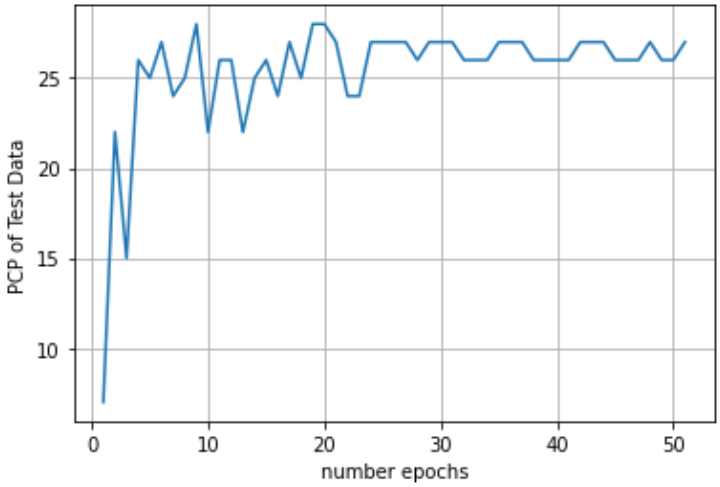

Figure 19

Figure 19 and 20 is for validation and the test dataset that we have used and as we expected it will increase and reach a certain point which it can't improve from that point beyond.

Now we are going to compare the PDJ parameters for different datasets.

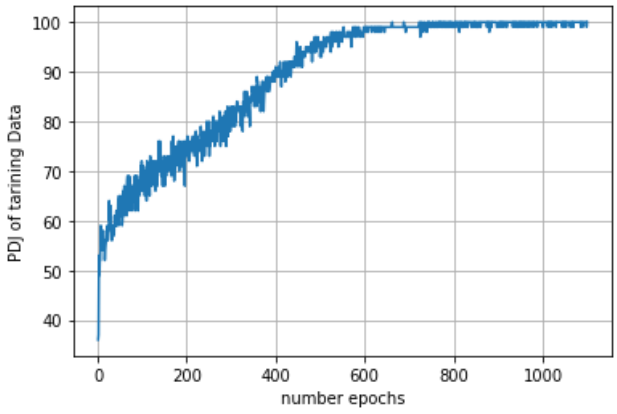

Figure 20

In figure 21 we can see the result is like parameter PCP and it reaches 100% but as we said before it is obvious that the model is overfitting again.

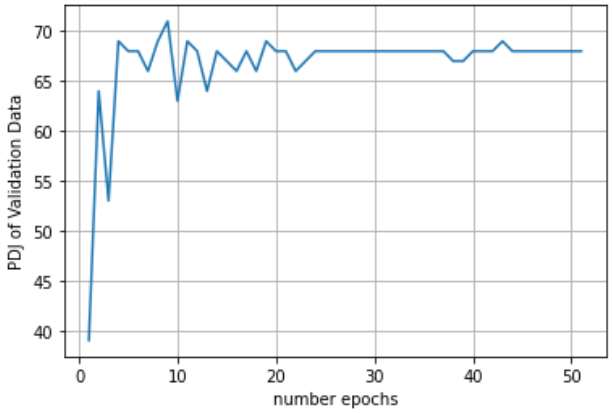

Figure 21

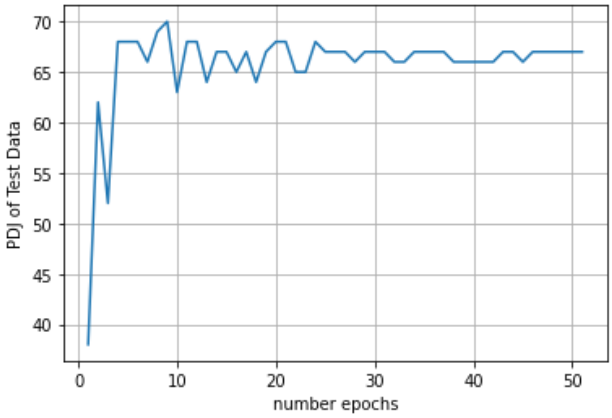

Figure 22

We can see the again in figure that it will reach a certain point.

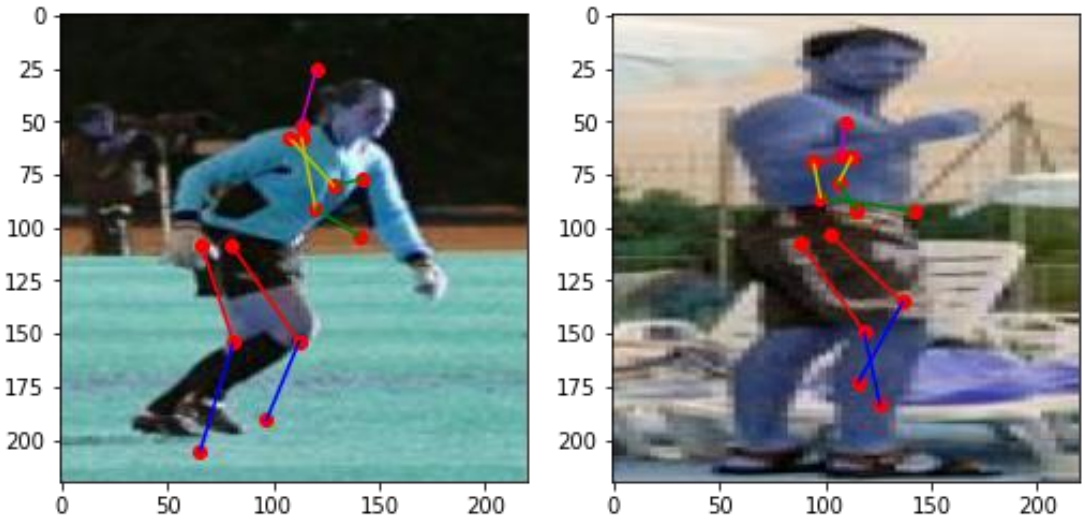

Figure 24 Test Dataset

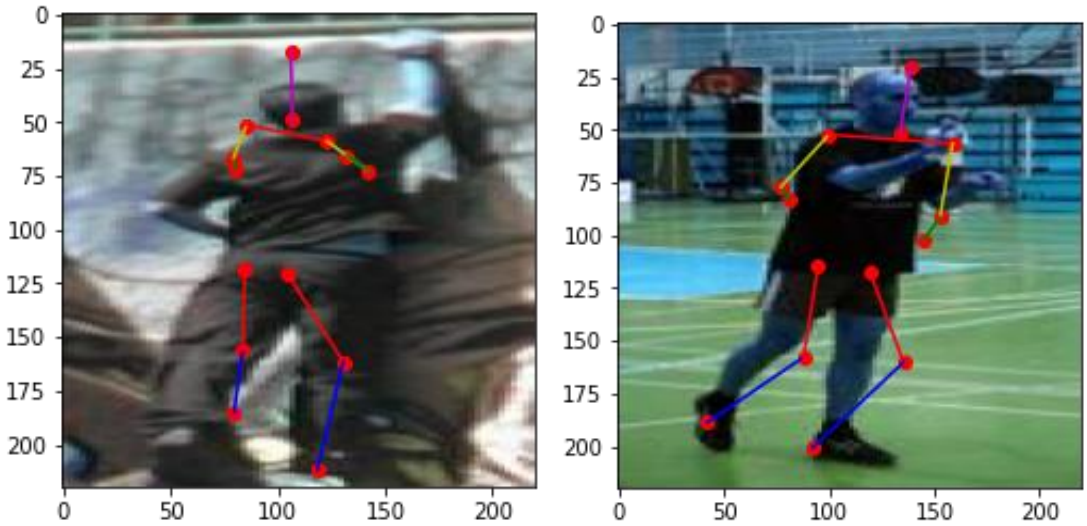

Figure 25 Test Dataset

In figure 24 and 25 are the result for the new improved dataset hasn't changed much but still in improved a bit because of the small dataset that we are using and because of the small memory we have on google Colab gave us on GPU we can't use the full potential of the data augmentation because of that but still the images, loss and the two accuracy parameters PCP and PDJ show it has improved a bit and we hope that if we had more data like as it was mentioned on the paper they used two dataset for train as mentioned FLIC with 5000 train images and 1000 test images and LSP dataset with 11000 images for sure we have got a better performance over there but of course we need a better GPU with a lot more memory to handle this kind of datasets.

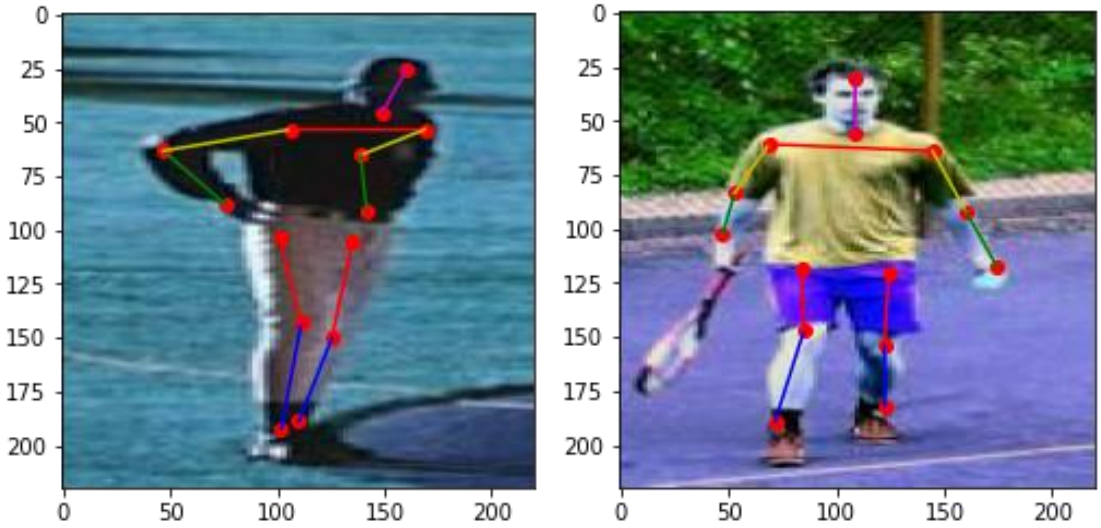

Figure 26 Train dataset

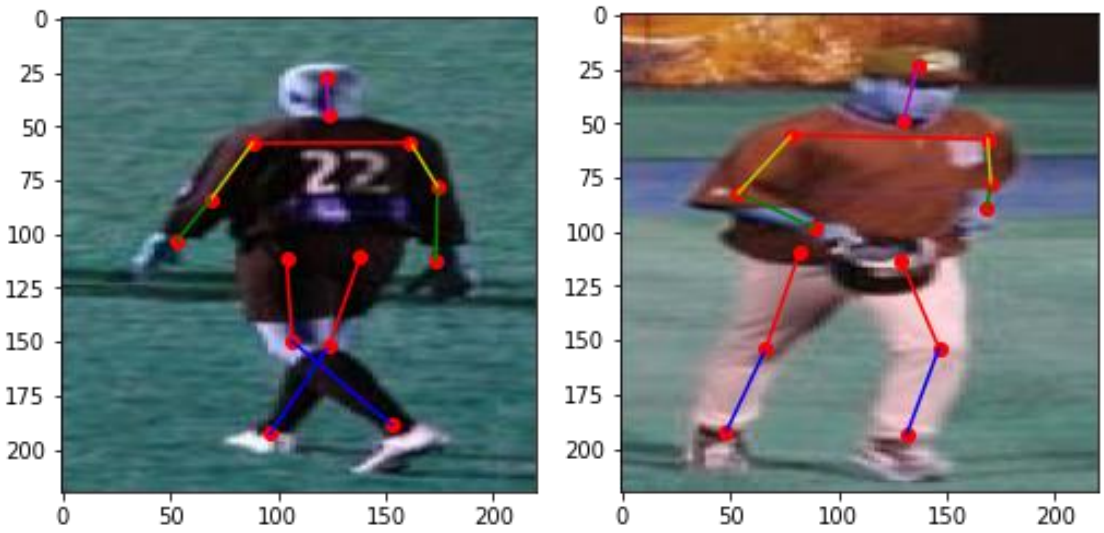

Figure 27 Train dataset

In figure 26 and 27 that the network still has memorized the training data but it done a better generalization for the new test data.

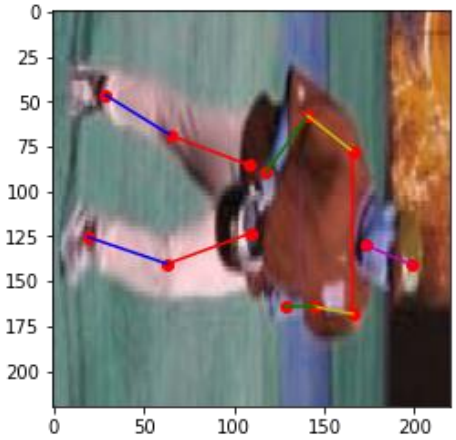

Figure 28

Figure 28 shows that the model is so over fitted even though we rotated the image we can

Part 3) In this part we need to add extra structures to the networks model the previous part of the question we just build the initial stage of the model in the next stage we are going to implement the S stage of the model.

S stage: This model works in a $s = \{1,2, \dots, k\}$ sequence after the initial stage we are going to make a bounding box around the estimated joint and the size of the bounding box in about the length of the torso of the body which we mentioned in the previous part of the question and after making the new bounding box of the image we need to scale the joints again and after that we send the image and the joints to the network again and we need to find a better location for this joint again and what we have done in this case we predict that joints again and tried to minimize the loss of the distance of that joint and the real joint of its bounding box and we do that with all the 14 joints of the body and we done the same thing that was mentioned in the paper, but there has a problem due to the google Colab storage that we have we can't use it for all the training data instead we used it over 100 of our data because when we used it the Colabratory gave us an error that we used all the limited memory in the GPU so we have problems with the accuracy of course and we can't do anything with it but still we show the implementation on the model and the result the loss graph and the PCP and the PDJ result it might not have a good result but there is the point that we didn't have much train data and the result isn't that good on the result.

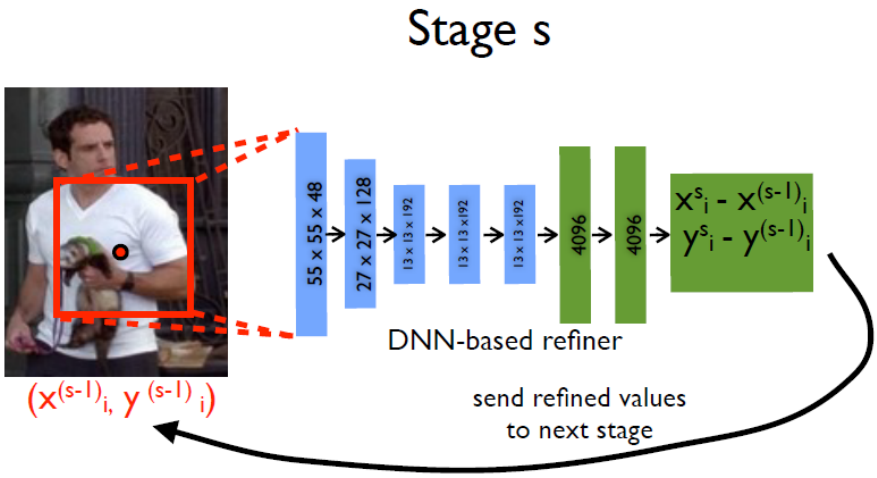

Figure 29

Figure 29 belongs to the *Deep pos* paper that in mentioned in the references and we are going to show some of the results for our network

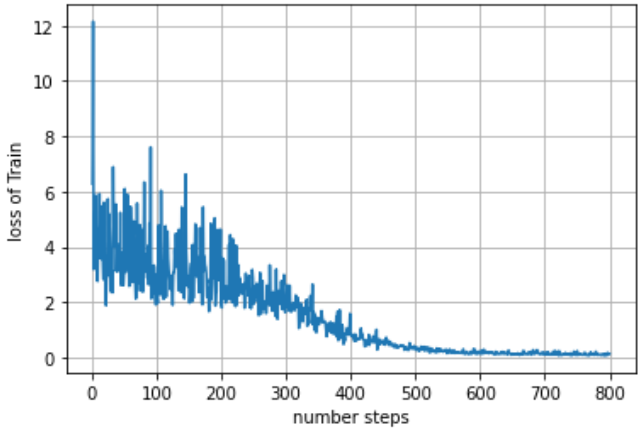

Figure 30

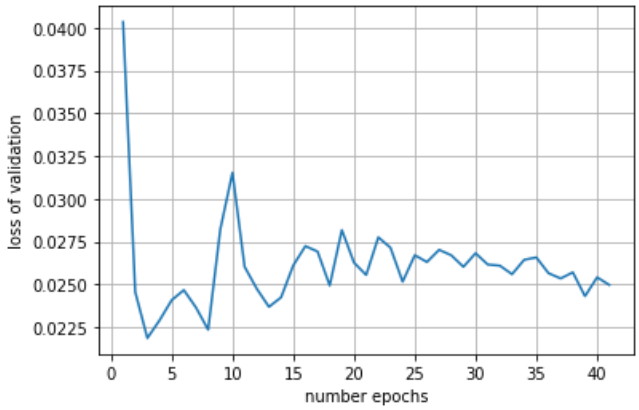

Figure 31

Figure 30 and 31 show the lost function for the model we added a new term to the model in order to solve the next step of the network.

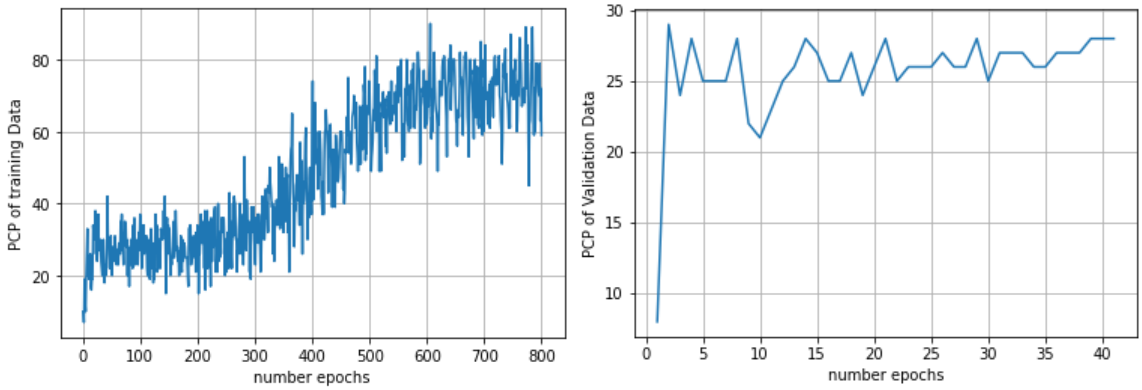

Figure 32

Figure 32 shows the result PCP parameter for the training data and the validation dataset

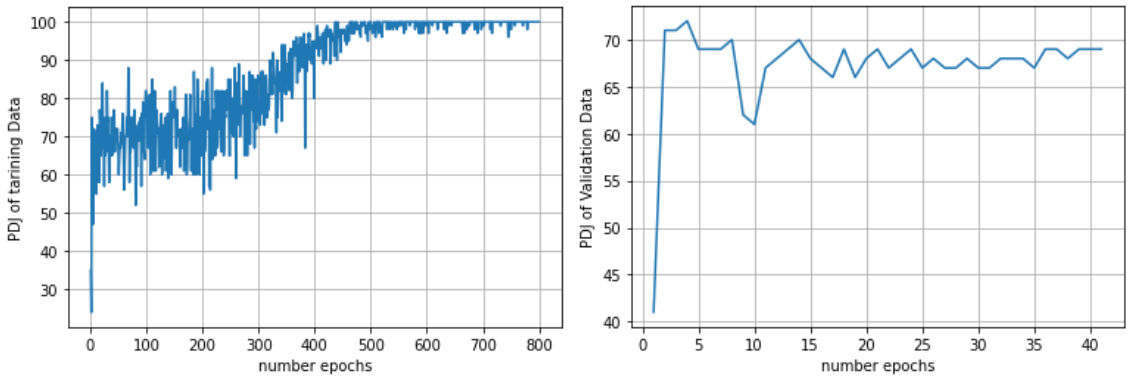

Figure 33

Figure shows the PDJ parameter for the new model we can see for train it reaches 100% at last.

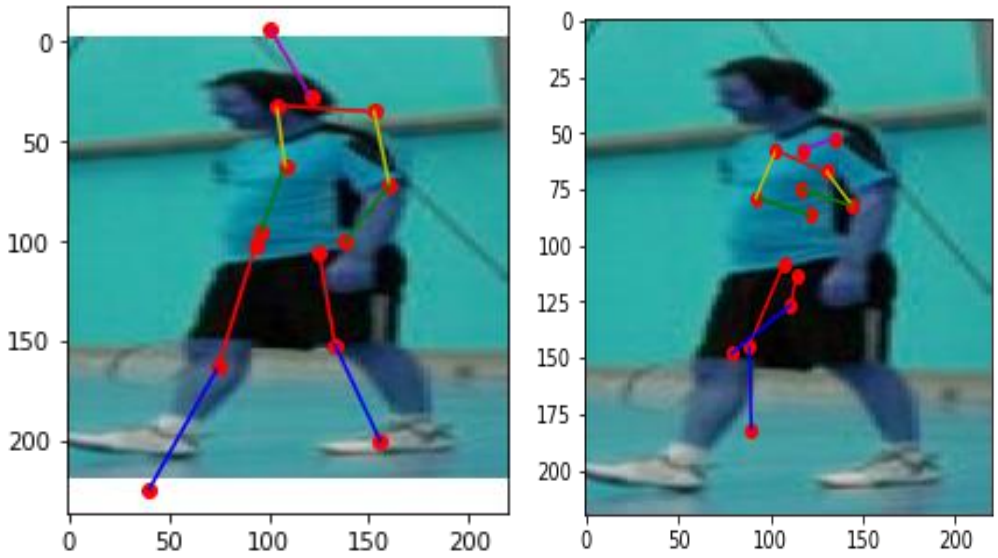

Figure 34

Figure 34 shows the result of the model for one sample image we can see we haven't done much good in it might have a lots of reasons like not having enough data and we were using some extra data and we only trained this model on a small dataset in order of 100 images and of course we can't expect. For that image we are going to show all the 14 bounding boxes that the model has faced.

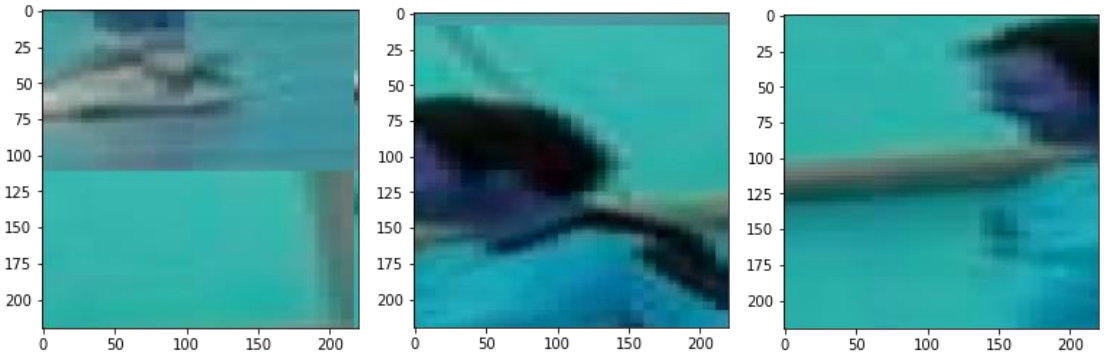

Figure 35

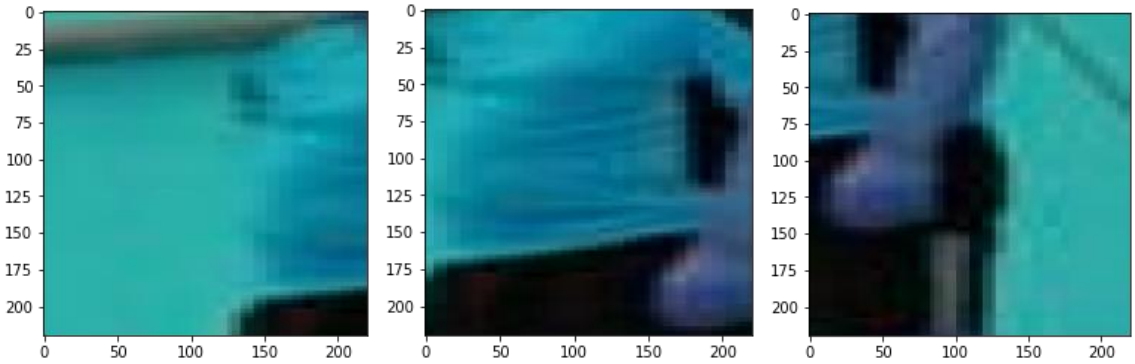

Figure 36

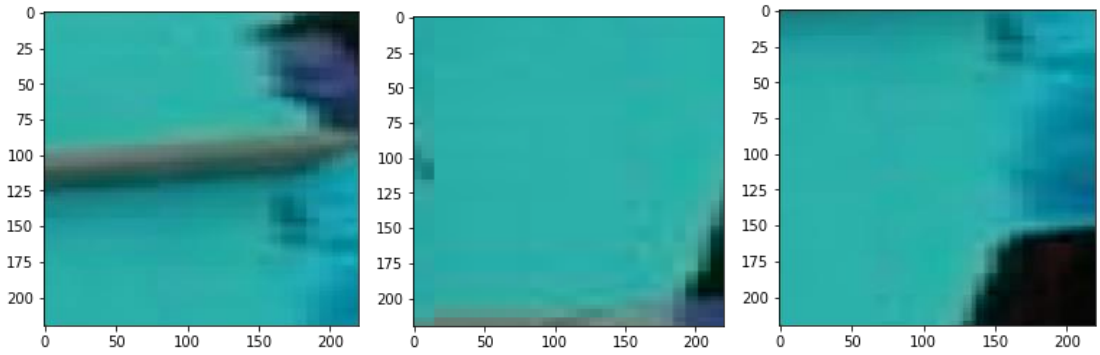

Figure 37

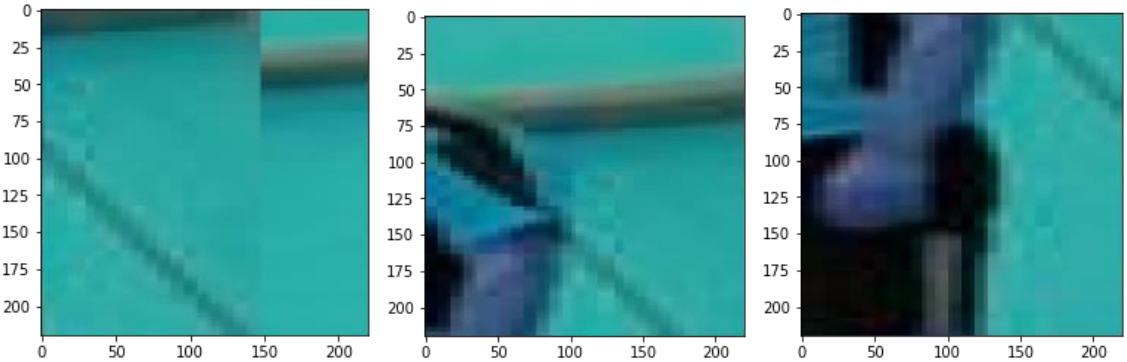

Figure 38

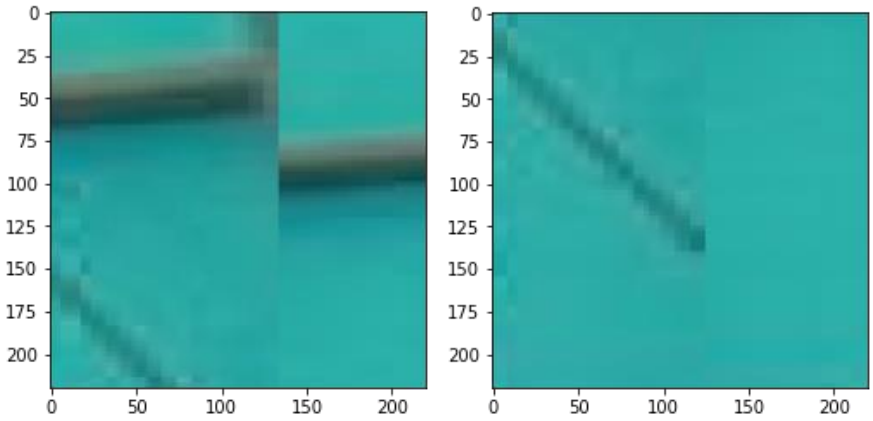

Figure 39

Figures 35 36 37 38 39 are the frames that has been used to make this image and they are the bounding box of the 14 joints that we have we can see some of them are for a particular joint but 2 of them are not showing anything useful.

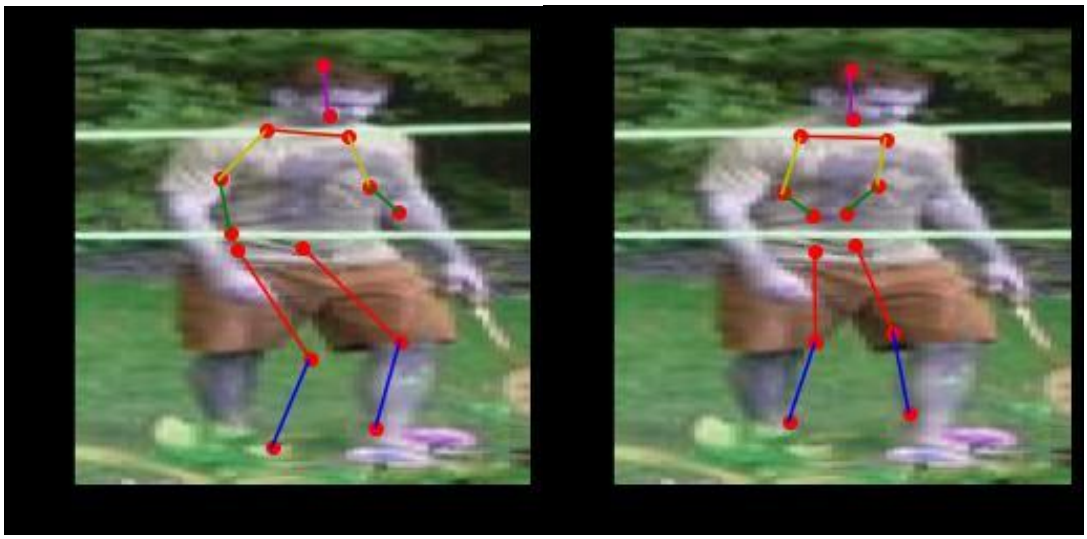

Figure 40

Figure 41

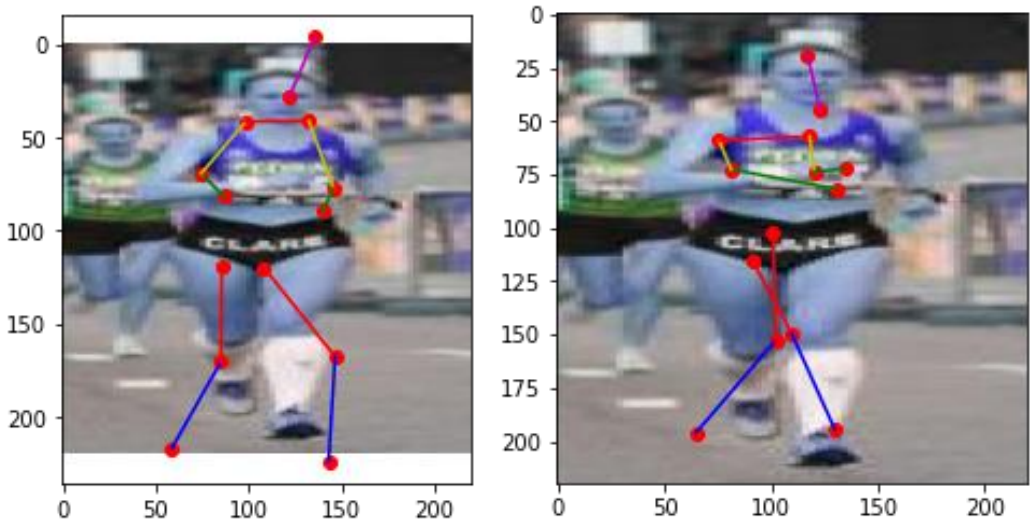

Figure 42

In figures 34 40 41 42 we can see the result for the model when we use a S stage model and as the result it didn't perform quite well but it is still good with the small datasets that we used.

Question 2

In this question we are using auto encoders in order to retain a fault image when we mention fault in an image it may have cracks, cuts, holes or paintings on it and we are going to use this structure to find the errors in the image.

How does an auto-encoder work, is starts with encoder which reduce the dimension of the space that it is for example the input is a $128 \times 128 \times 3$ image and the bottle neck of the structure has 100 neurons.

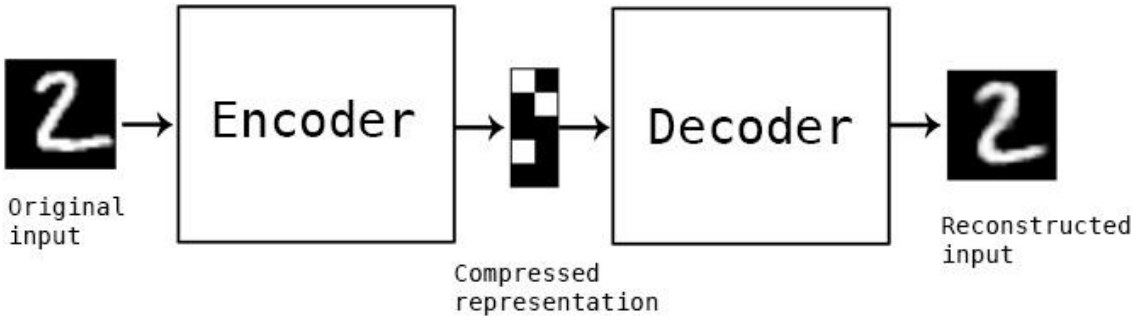

Figure 43

Figure 43 shows an example of what auto-encoders can do by feeding the network with tons and tons of data it will start to learn the patterns of the input image and this will help in different ways like when we feed the network with a fault image because in the bottle neck of the network was small number of neuron which means we can represent the output image

with a small number of features and we will train the network with good images and then if we gave them an image will a crack in it because of the small number of features it has we expect to see the image without fault in the output, it's because of the space that we are in that space has less number of features and the network must learn how to squeeze the input data and convert it to a loss dimensional vector and after that it tries to convert that low dimensional vector to a output image and it must be similar to the input as possible.

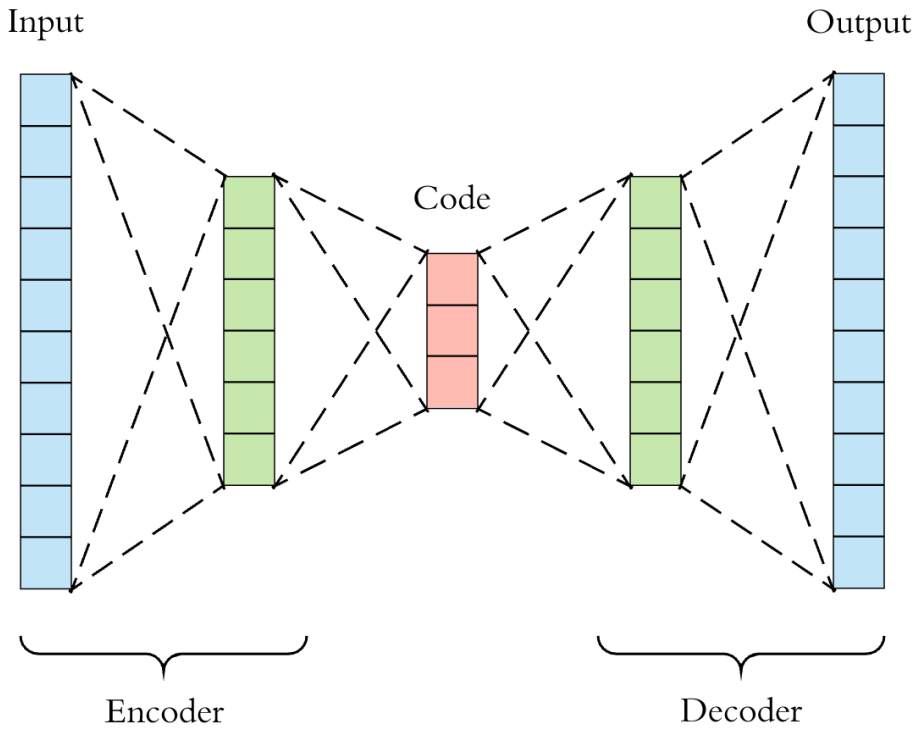

Figure 44

Figure 44 shows the structure of the auto-encoder and the way it looks it can we seen that the dimension of the space in the bottle neck is quite smaller than the input and output dimension, in the following we are going to use different type of layers to see the accuracy and effect in the output.

Part 1) In the first one we are going to $Leaky\ ReLU$ activation function and sigmoid function of the last layer. We used batch size 32 and got this result in the following.

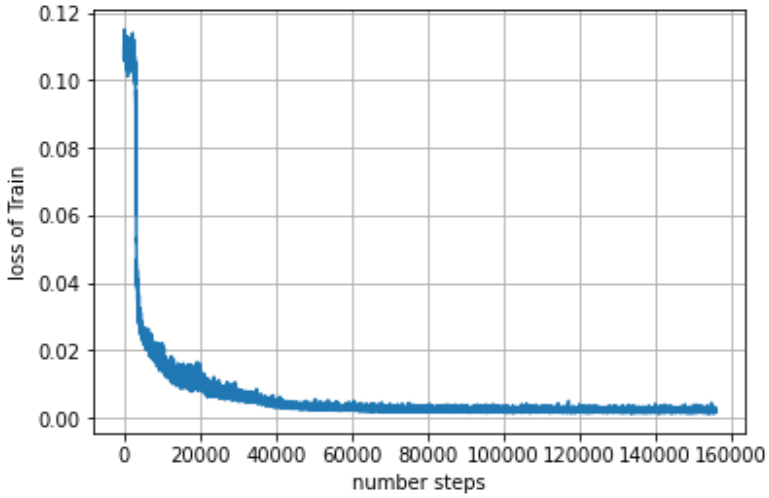

Figure 45

Figure 45 show the loss for the

For the accuracy we are using 3 methods

$$Method1\ Accuracy = \frac{num(fault\ pixels \cap predicted\ fault\ pixel)}{num(fault\ pixels)}$$

$$Method2\ Accuracy = \frac{num(fault\ pixels \cap predicted\ fault\ pixel)}{num(predicted\ fault\ pixels)}$$

$$Method3\ Accuracy = \frac{num(fault\ pixels == predicted\ fault\ pixel)}{num(\ all\ pixels)}$$

Part 1) We are going to use the sigmoid function on the last layer of the network

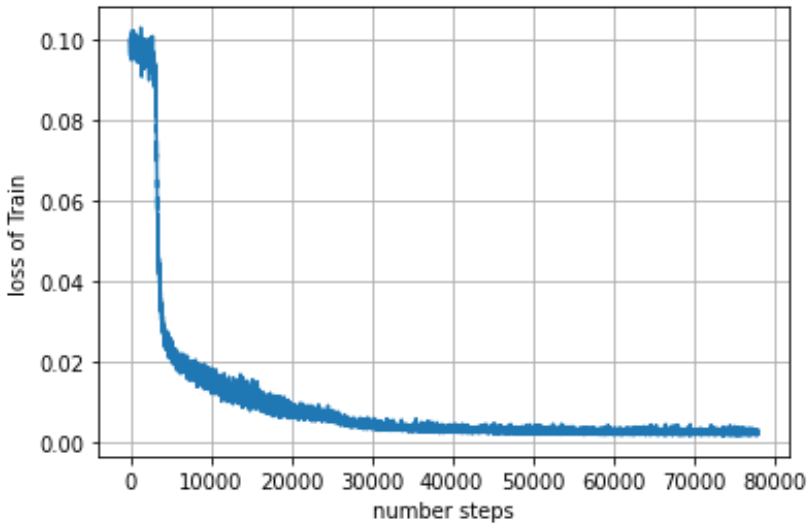

Figure 46

Figure 46 shows the loss of the model we can see that as we expect the loss is decreasing by epochs and because we are using batches the number of steps is $\frac{DataSetSize}{batchSiz} \times Epochs$ that why we see a lot of steps in it.

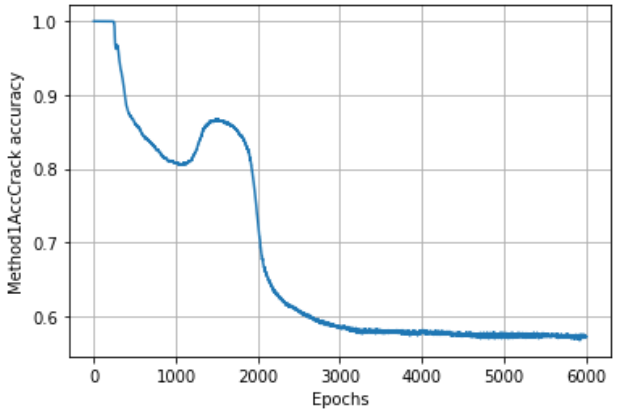

Figure 47

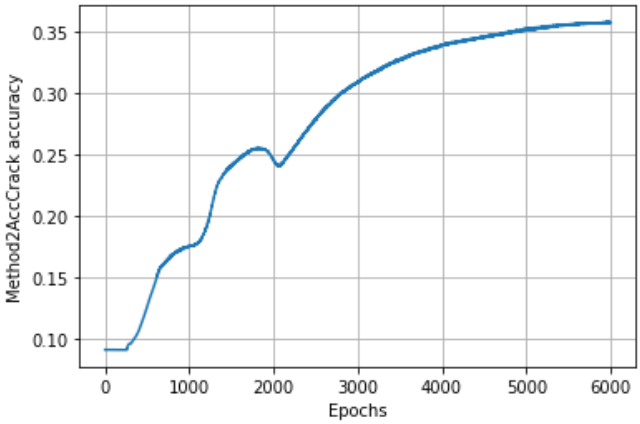

Figure 48

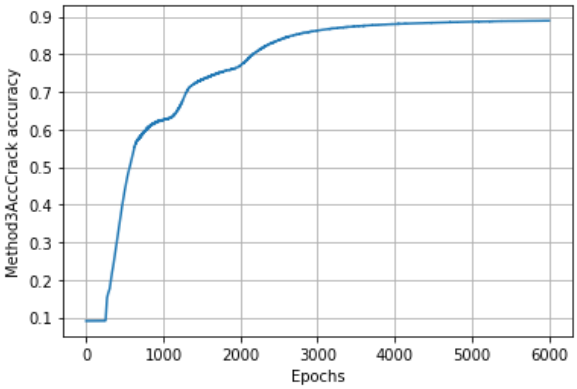

Figure 49

Figure 47 48 49 shows the 3 accuracy that we have and we mentioned every method for the accuracy. The accuracy is for the crack dataset.

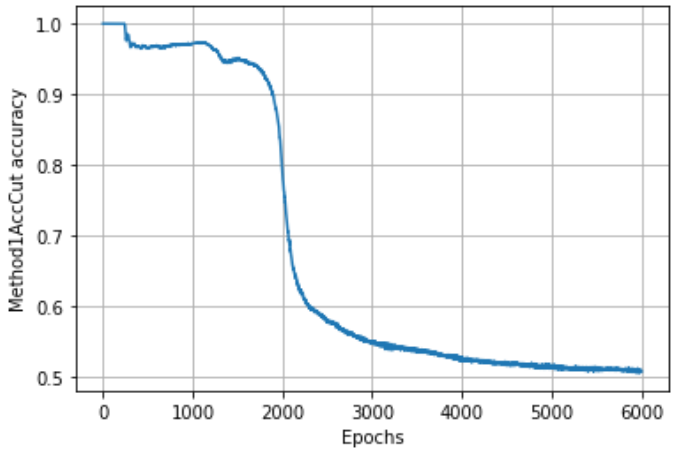

Figure 50

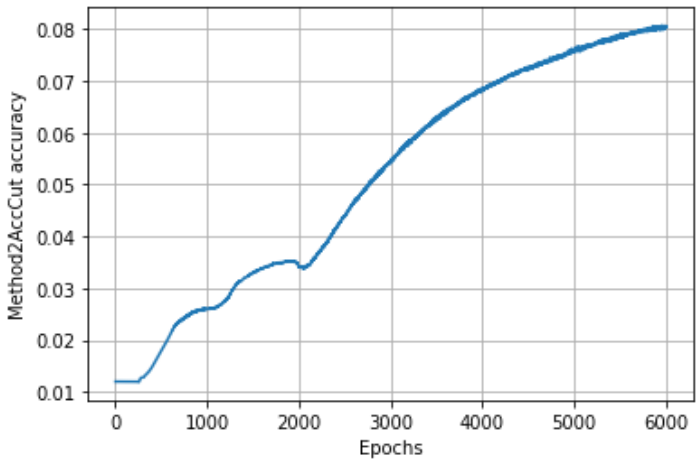

Figure 51

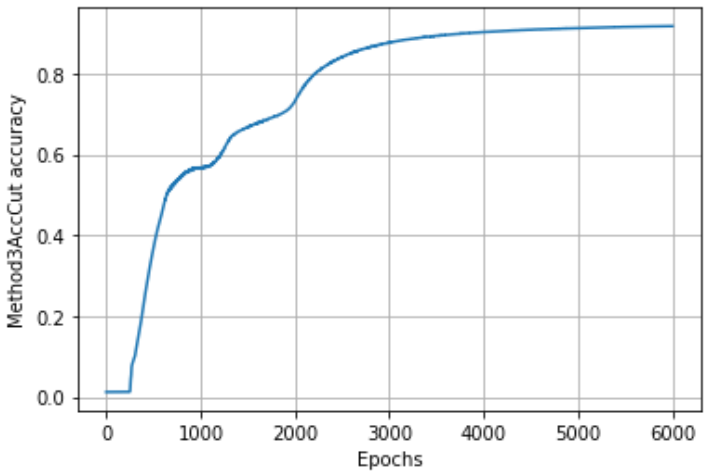

Figure 52

Figures 50 51 52 show the accuracy of the model for the 3 methods that we mentioned, we used the cut dataset

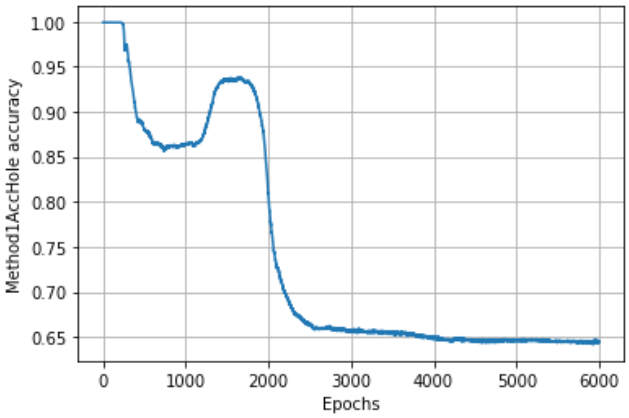

Figure 53

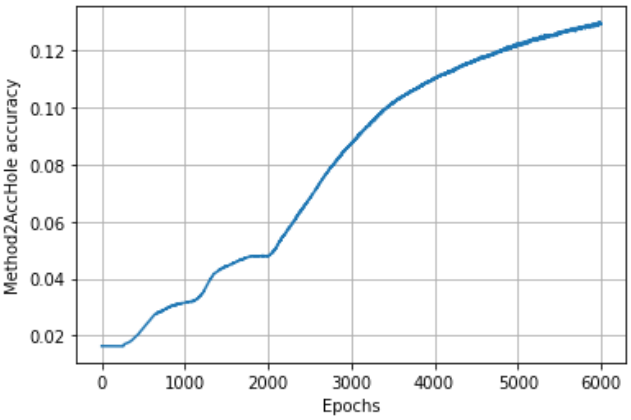

Figure 54

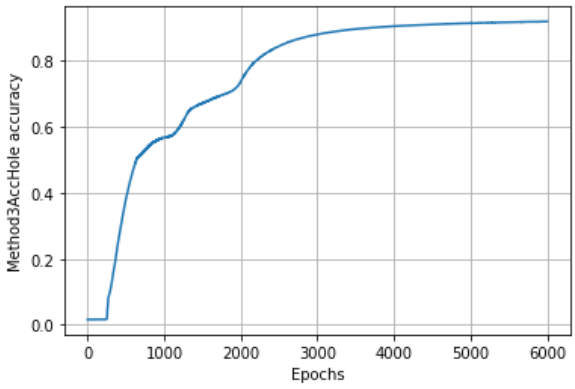

Figure 55

Figures 53 54 55 show the accuracy of the model for the 3 methods that we mentioned, we used the hole dataset

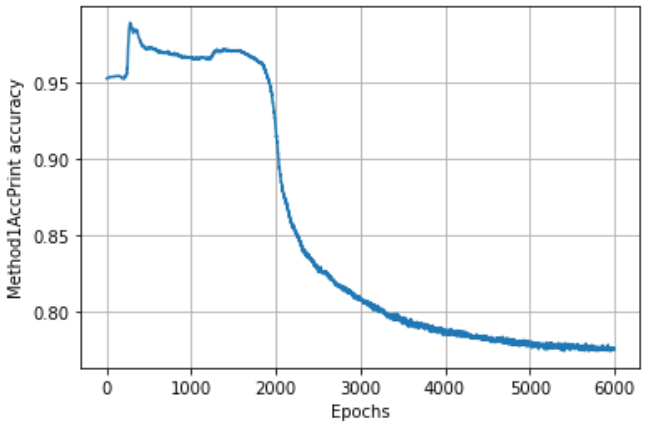

Figure 56

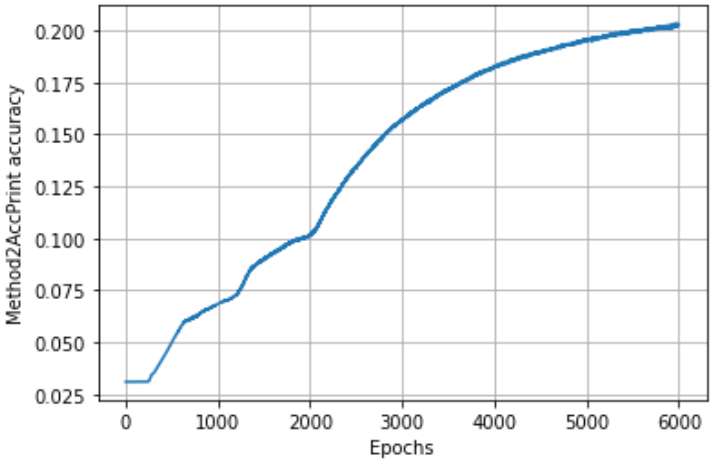

Figure 57

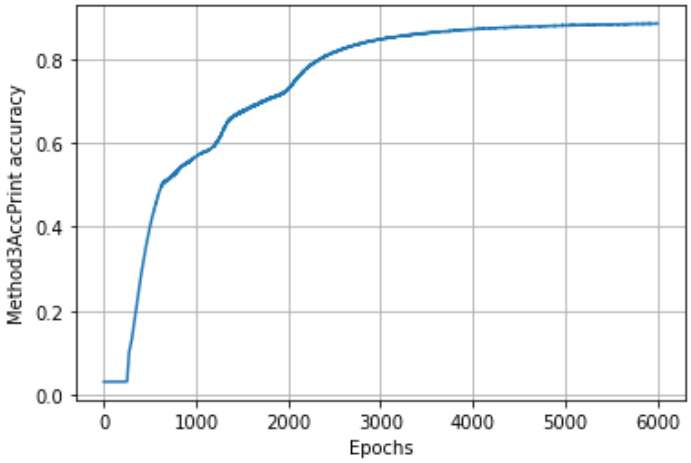

Figure 58

Figure 56 57 58 shows the accuracy with the 3 different methods for the print dataset.

This graphs were for the $threshold = 0.1$ and the next 3 part we are going to use the same but we can see the result with different images.

Crack:

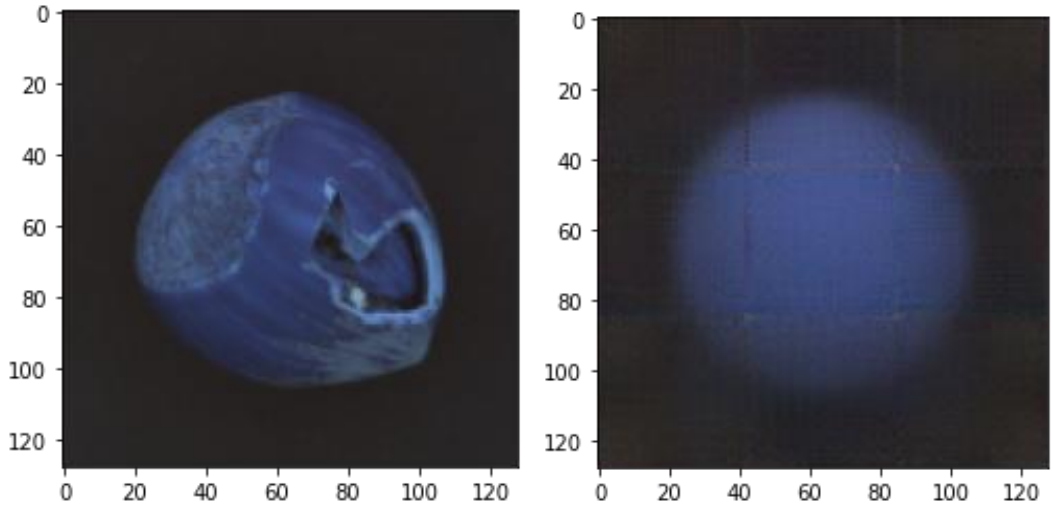

Figure 59

Figures 59 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them

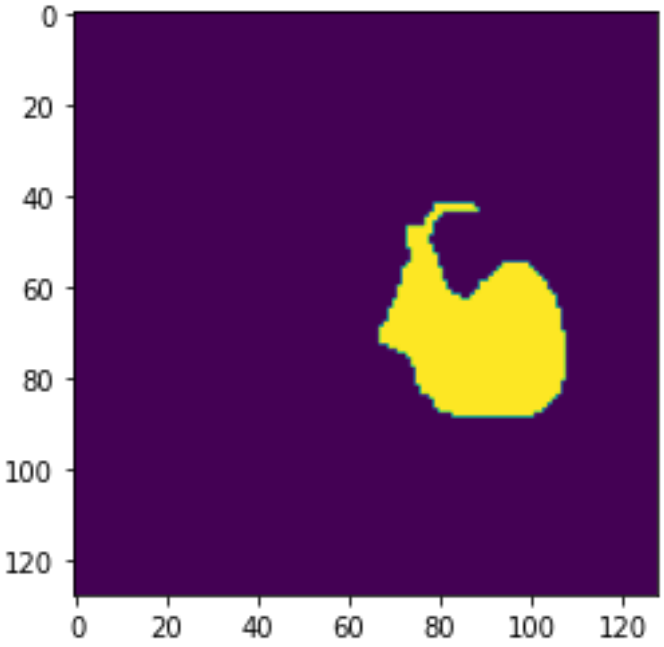

Figure 60

Figure 60 shows the real mask that where the image hast problems and the network is supposed to find them.

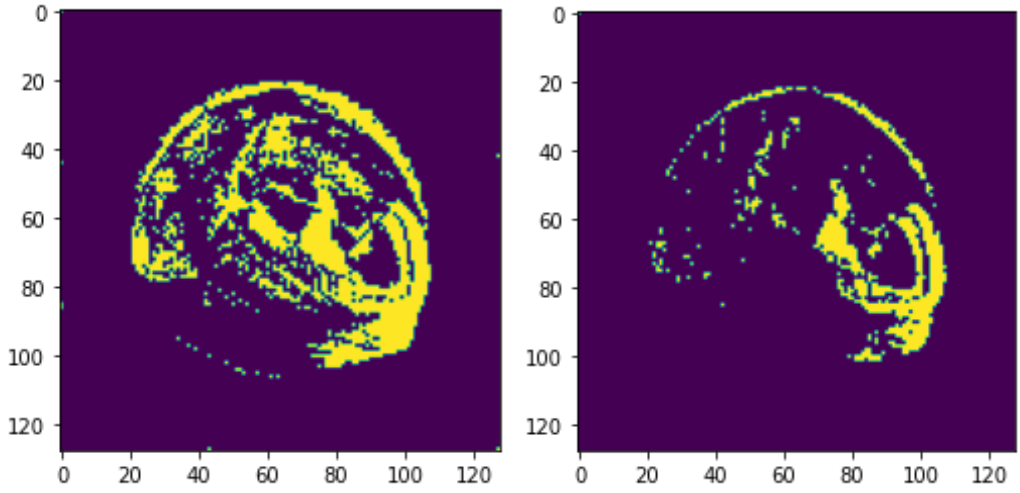

Figure 61

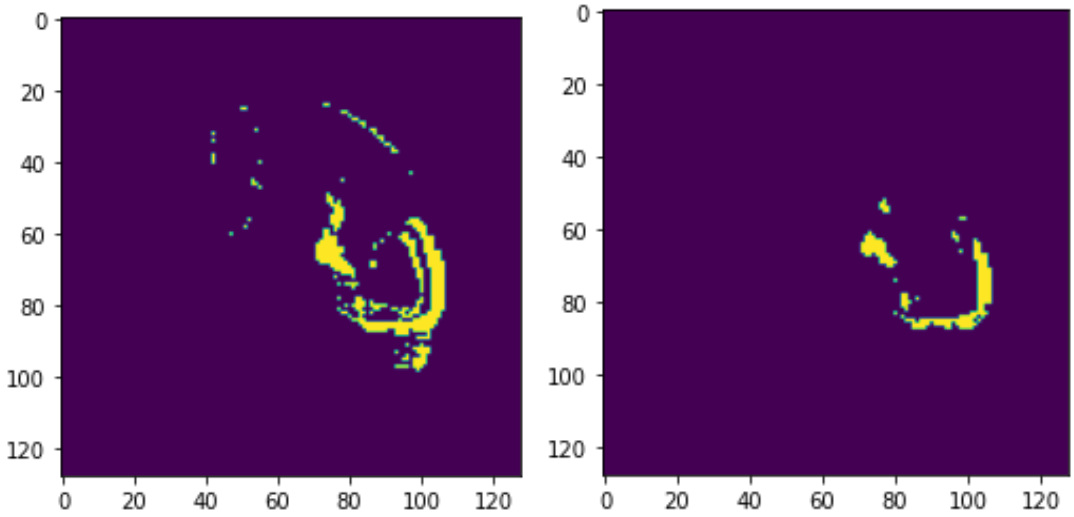

Figure 62

Figure 61 62 from left to right we are using $threshold$ 0.1, 0.15, 0.2, 0.3 for them and we can see the result

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	62%	47%	33%	17%
Method2	30%	58%	82%	100%
Methdo3	87%	93%	94%	94%

Cut:

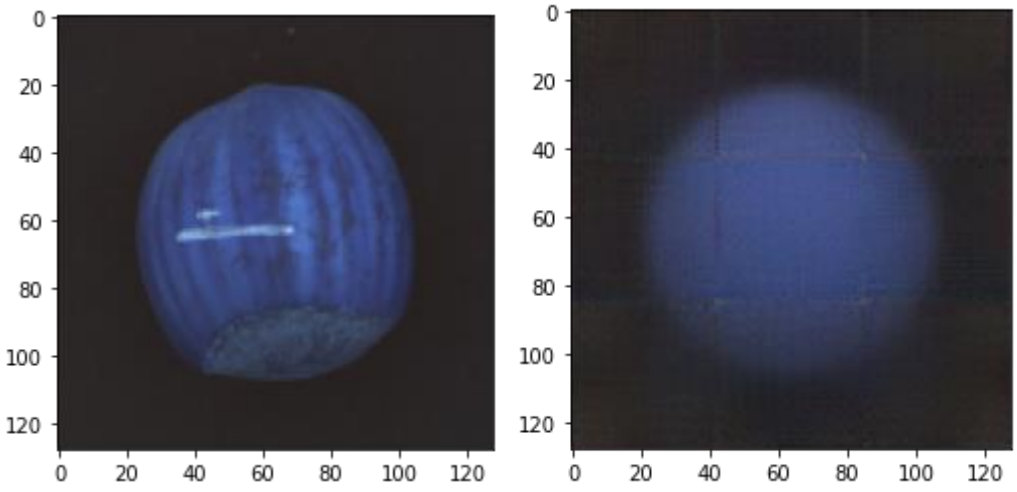

Figure 63

Figures 63 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them

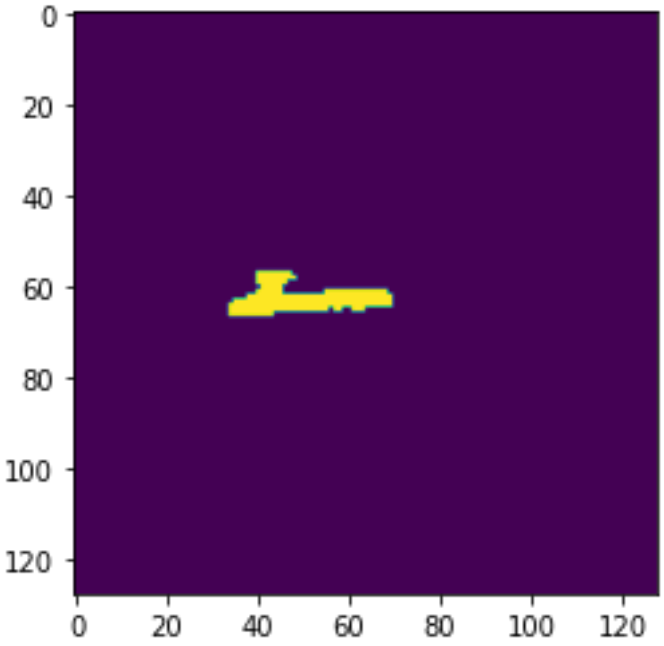

Figure 64

Figure 64 shows the real mask that where the image hast problems and the network is supposed to find them.

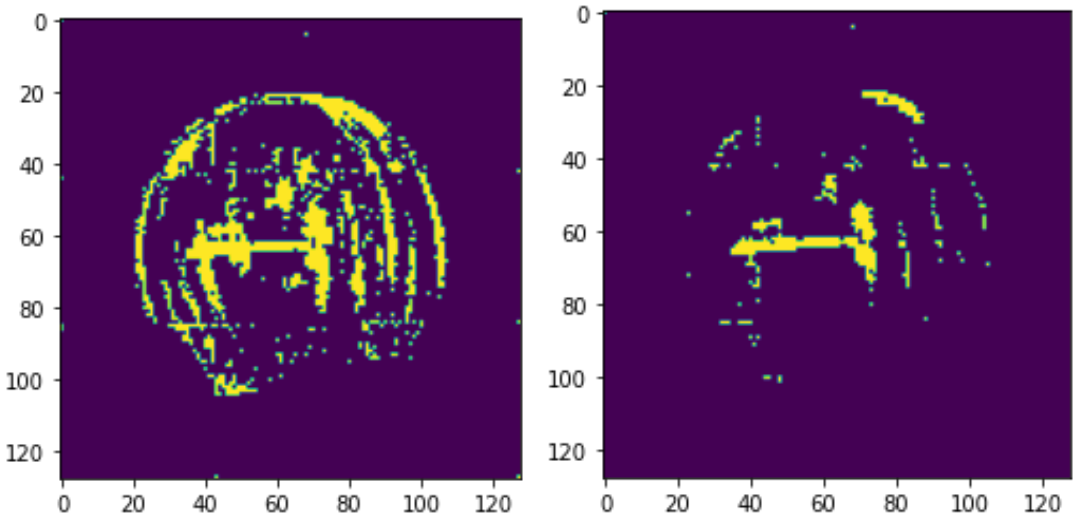

Figure 65

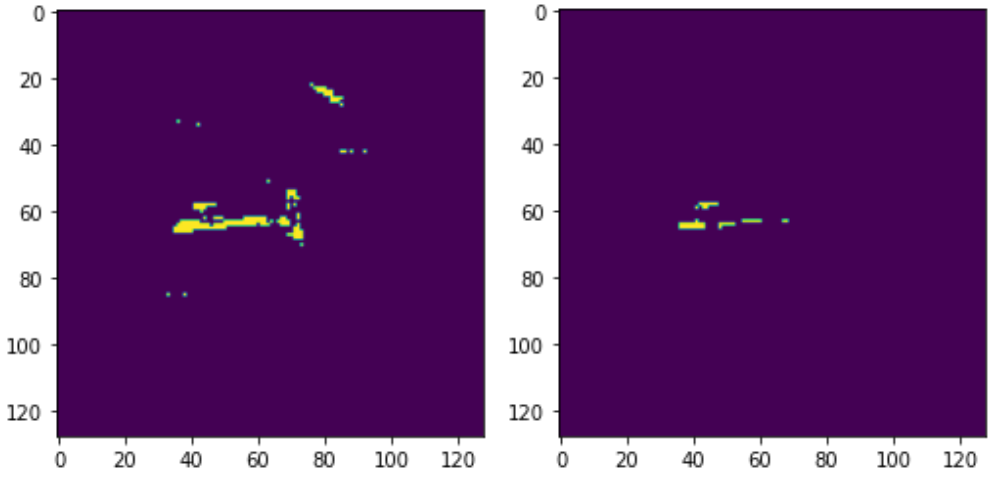

Figure 66

Figure 65 66 from left to right we are using $threshold$ 0.1, 0.15, 0.2, 0.3 for them and we can see the result.

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	77%	68%	52%	21%
Method2	10%	30%	62%	100%
Methdo3	91%	97%	99%	99%

Hole:

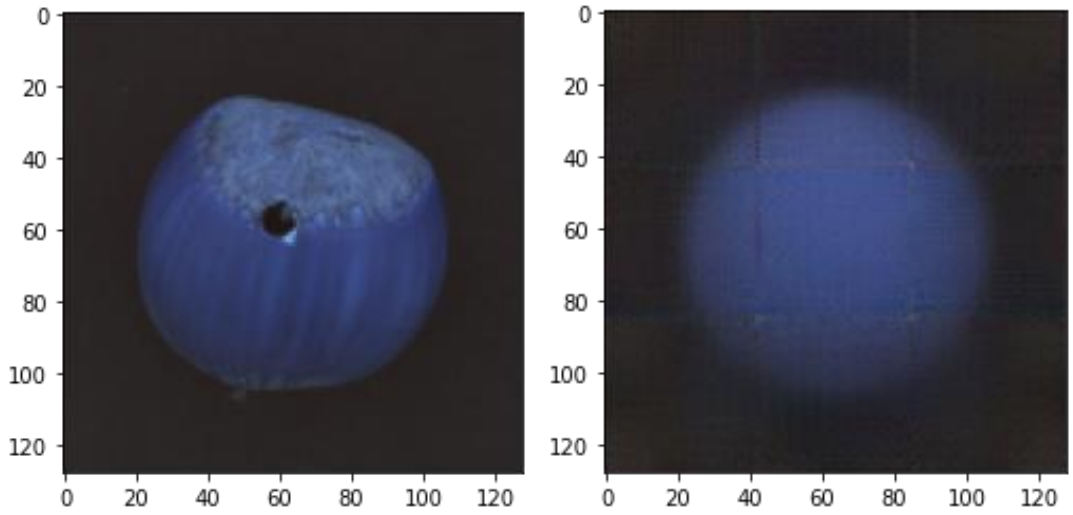

Figure 67

Figures 67 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them.

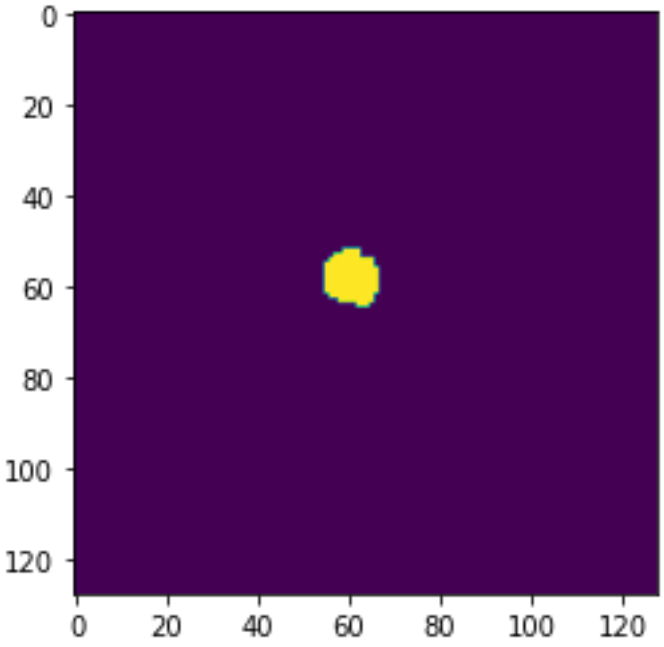

Figure 68

Figure 68 shows the real mask that where the image hast problems and the network is supposed to find them.

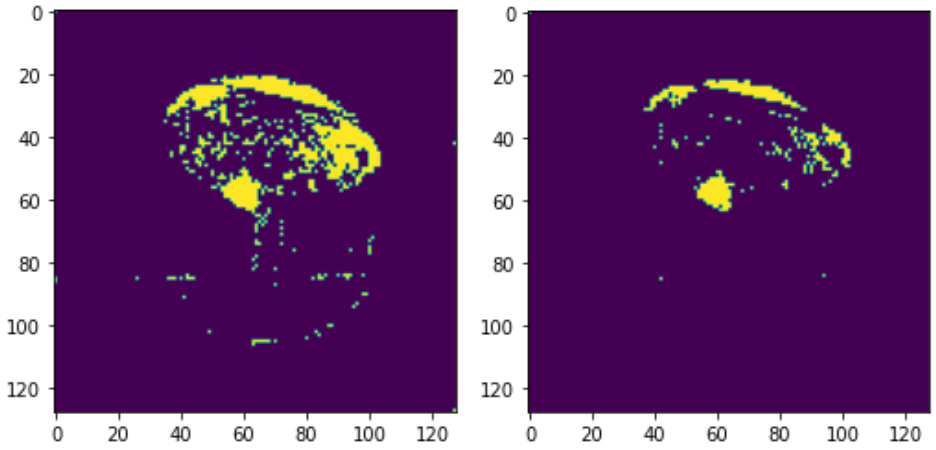

Figure 69

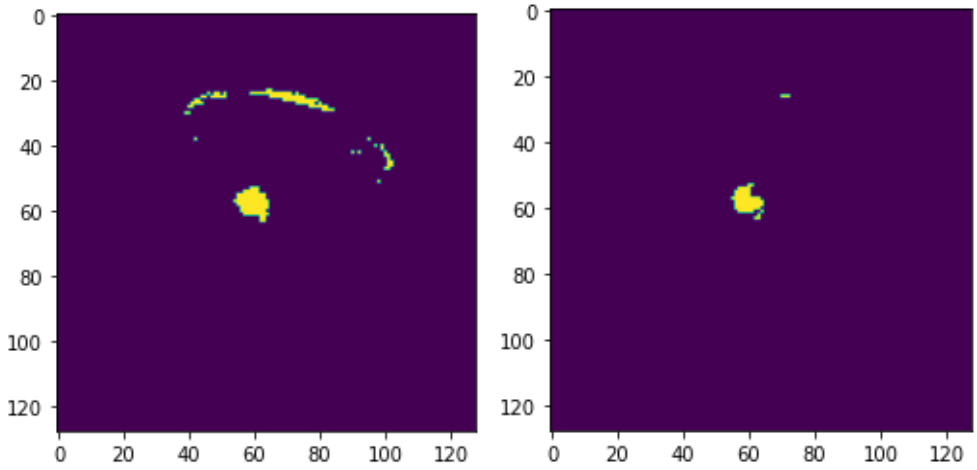

Figure 70

Figure 69 70 from left to right we are using $threshold$ 0.1, 0.15, 0.2, 0.3 for them and we can see the result.

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	79%	69%	60%	21%
Method2	10%	22%	44%	100%
Methdo3	94%	98%	99%	99%

Print:

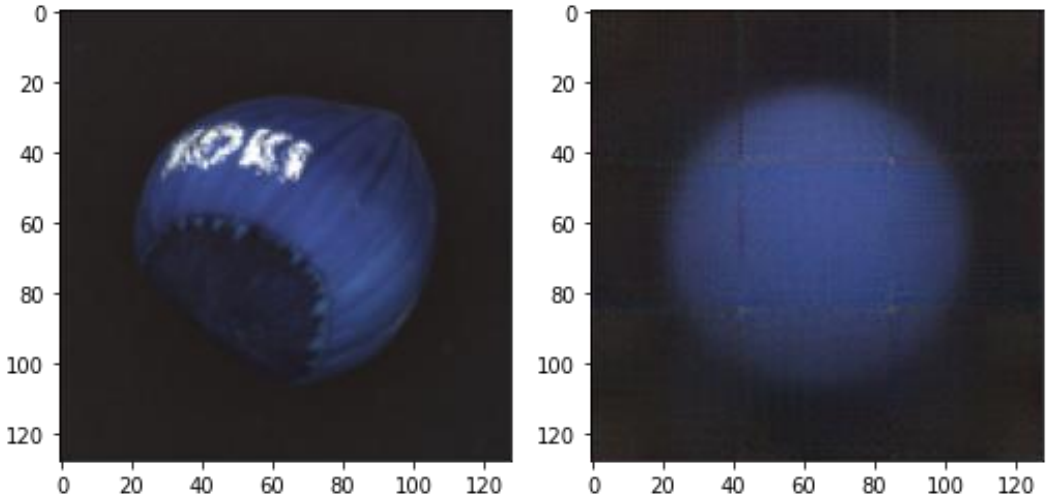

Figure 71

Figures 71 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them.

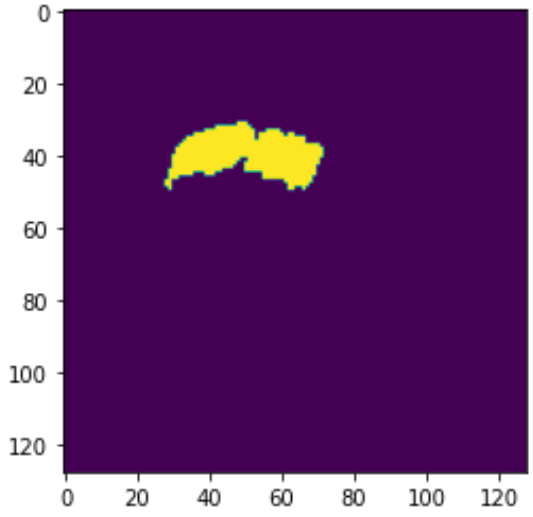

Figure 72

Figure 72 shows the real mask that where the image hast problems and the network is supposed to find them.

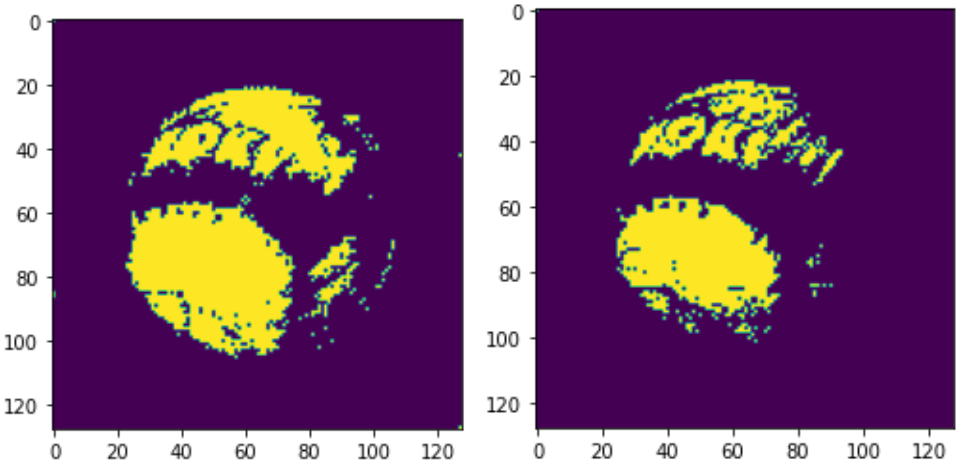

Figure 73

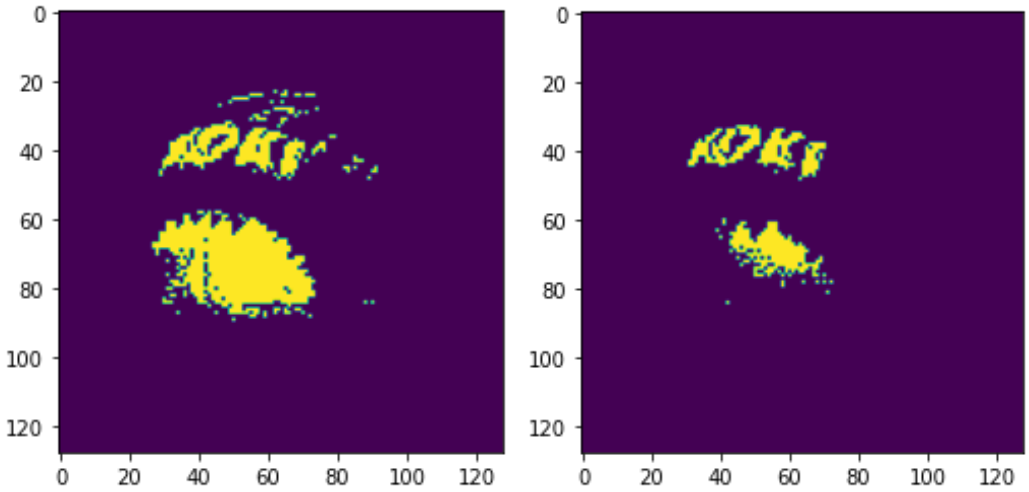

Figure 74

Figure 73 74 from left to right we are using $threshold$ 0.1, 0.15, 0.2, 0.3 for them and we can see the result.

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	78%	66%	62%	21%
Method2	13%	17%	25%	100%
Methdo3	83%	89%	93%	99%

Part 2) In this part we are adding a batch normalization to our model

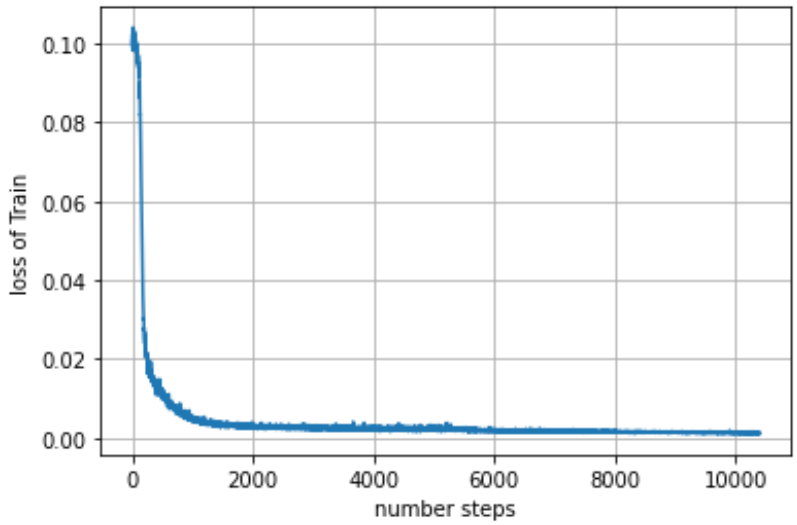

Figure 75

Figure 75 shows the loss of the model we can see that as we expect the loss is decreasing by epochs and because we are using batches the number of steps is $\frac{DataSetSize}{batchSiz} \times Epochs$ that why we see a lot of steps in it.

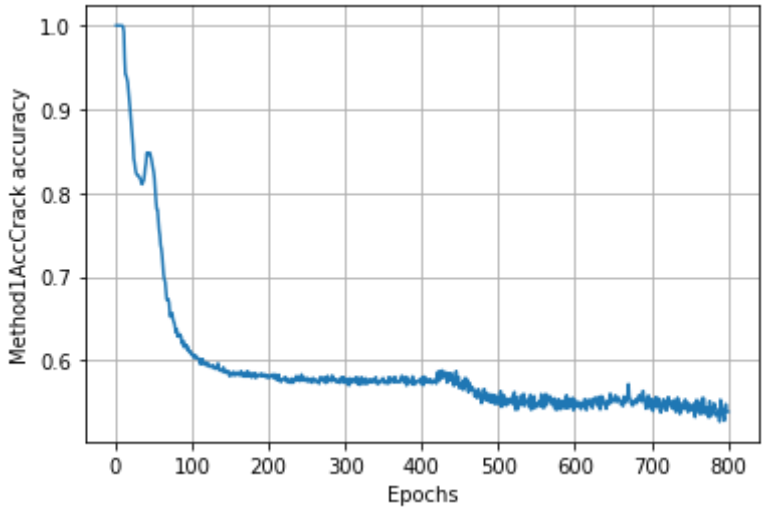

Figure 76

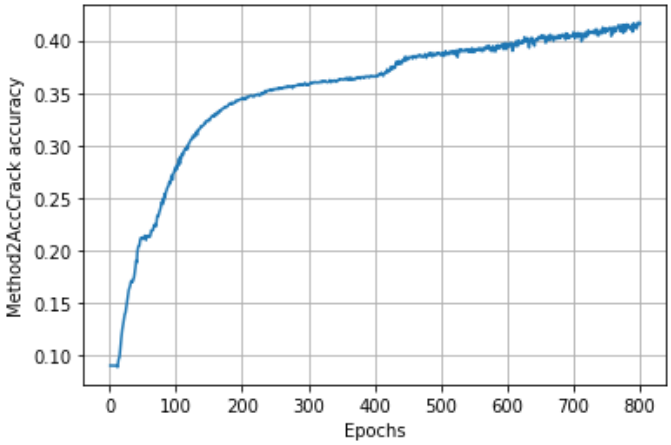

Figure 77

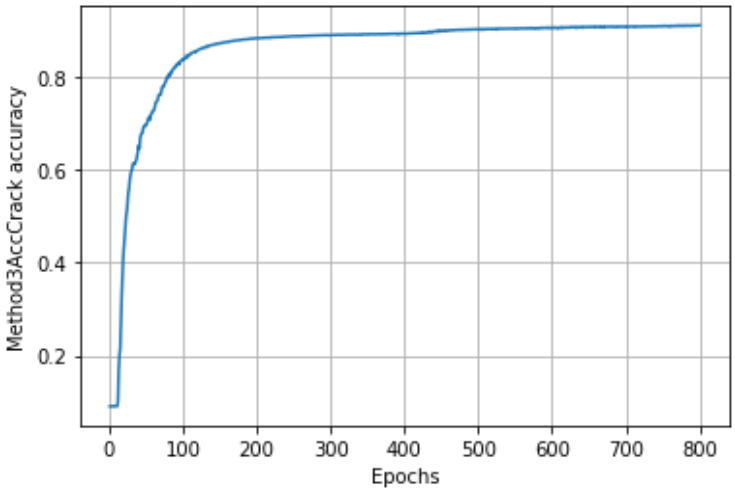

Figure 78

Figure 76 77 78 shows the 3 accuracy that we have and we mentioned every method for the accuracy. The accuracy is for the crack dataset.

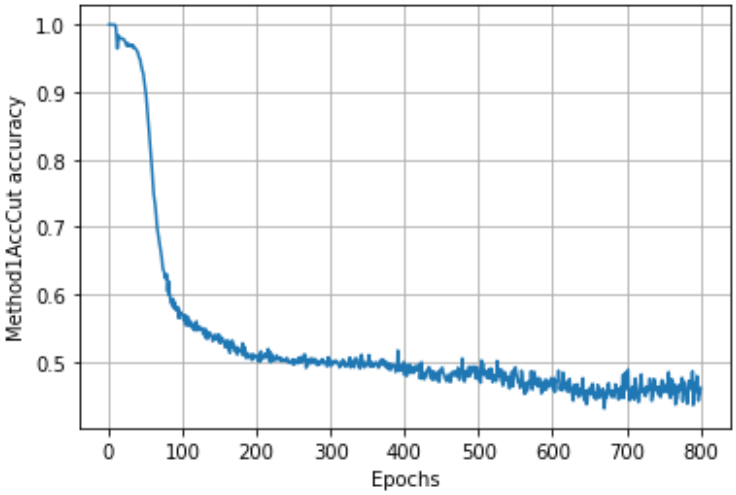

Figure 79

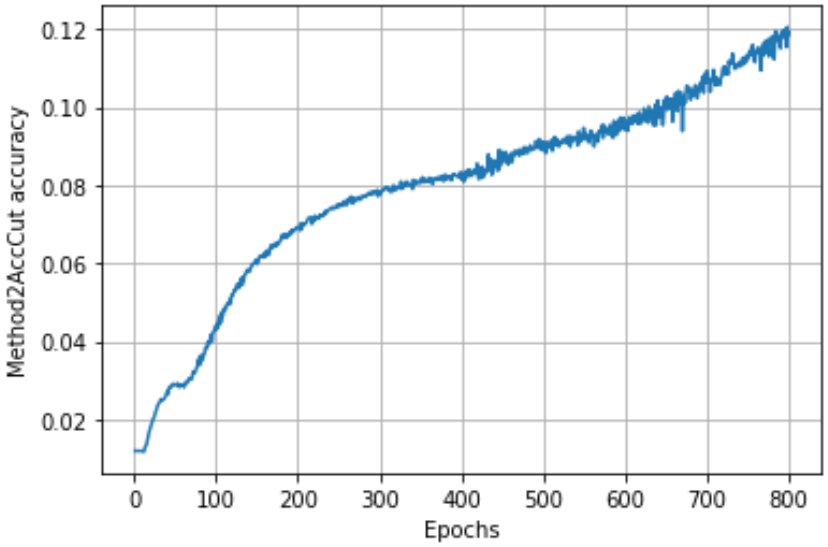

Figure 80

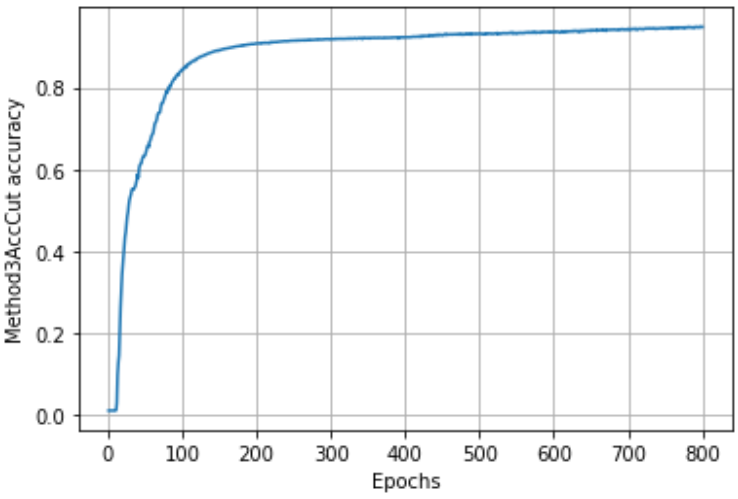

Figure 81

Figures 79 80 81 show the accuracy of the model for the 3 methods that we mentioned, we used the cut dataset

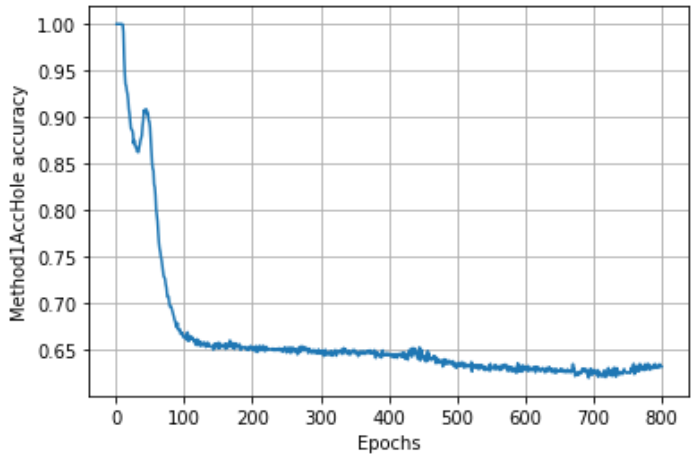

Figure 82

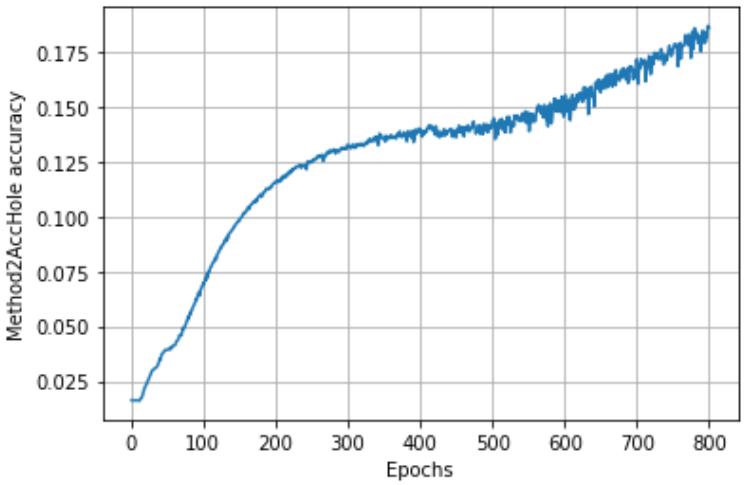

Figure 83

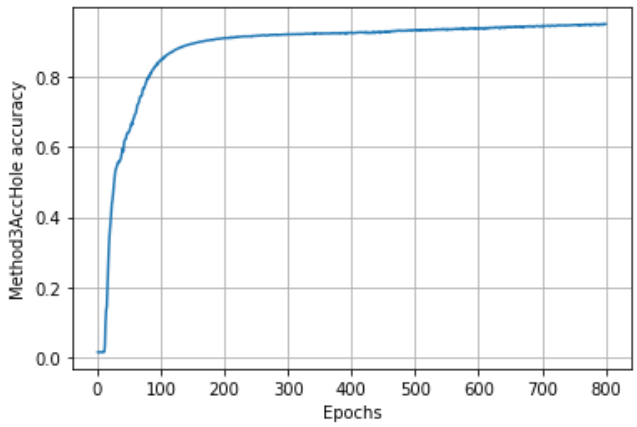

Figure 84

Figures 82 83 84 show the accuracy of the model for the 3 methods that we mentioned, we used the hole dataset

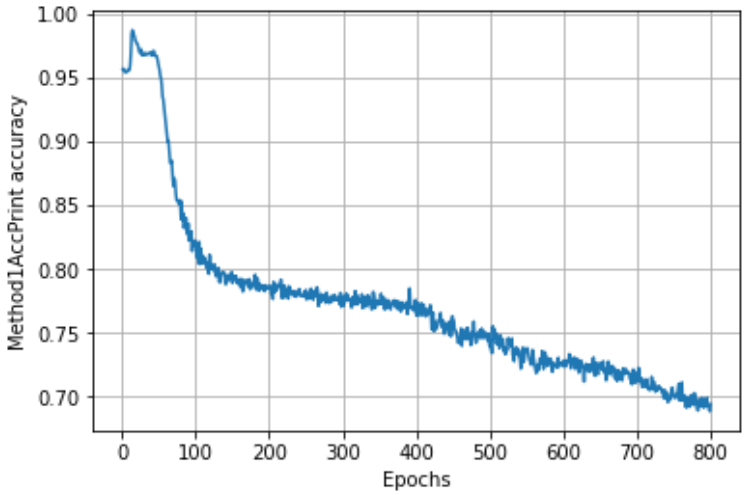

Figure 85

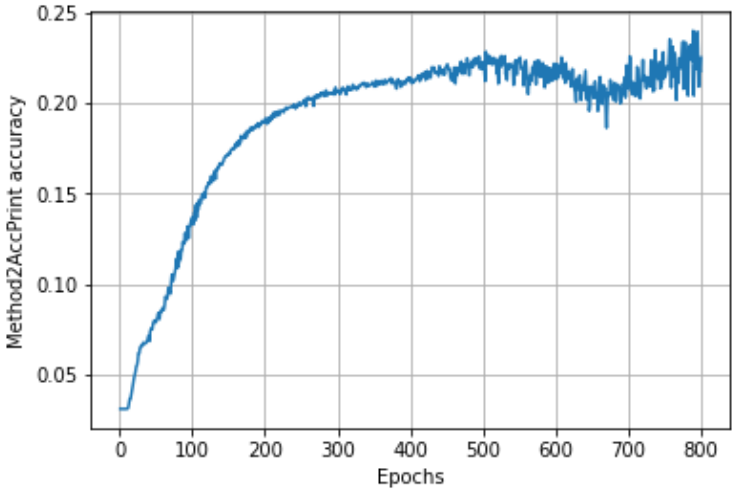

Figure 86

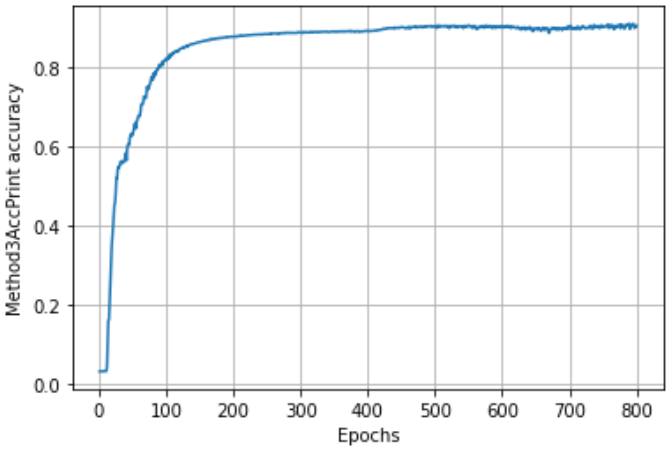

Figure 87

Figure 85 86 87 shows the accuracy with the 3 different methods for the print dataset.

Crack:

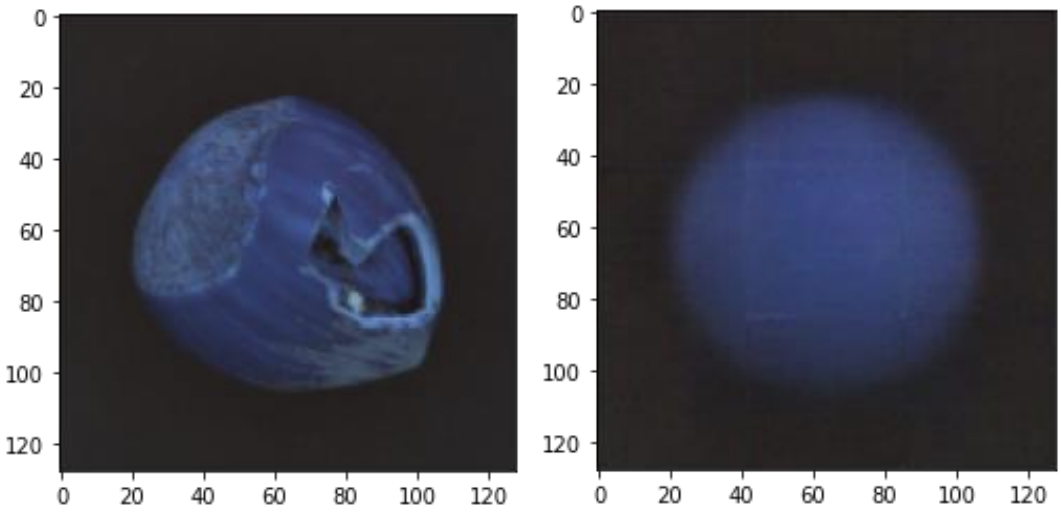

Figure 88

Figures 88 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them

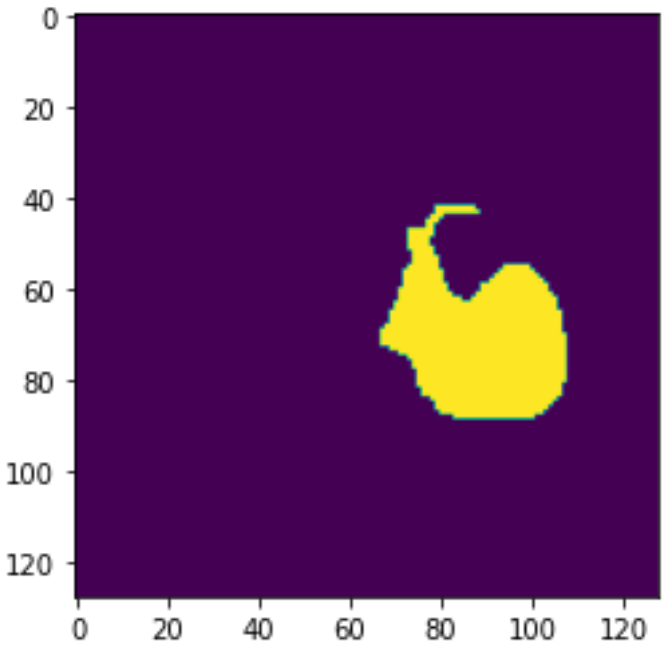

Figure 89

Figure 89 shows the real mask that where the image hast problems and the network is supposed to find them.

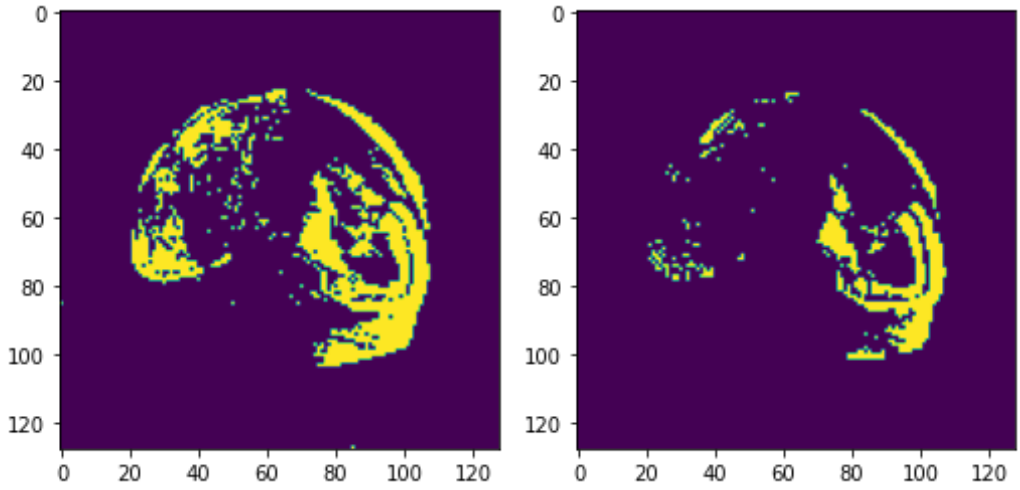

Figure 90

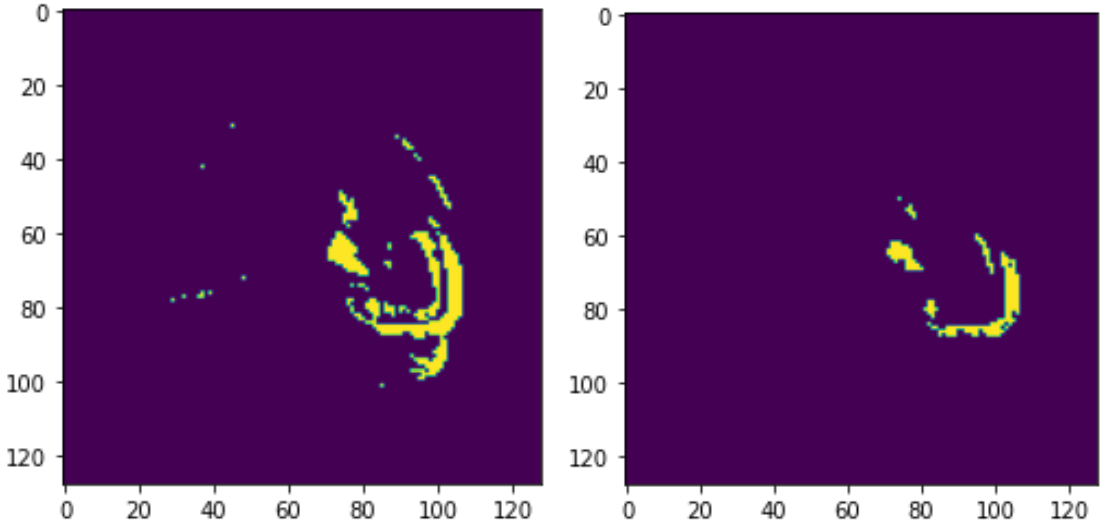

Figure 91

Figure 90 91 from left to right we are using *threshold* 0.1, 0.15, 0.2, 0.3 for them and we can see the result

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	61%	44%	32%	17%
Method2	38%	60%	82%	100%
Methdo3	90%	94%	95%	94%

Cut:

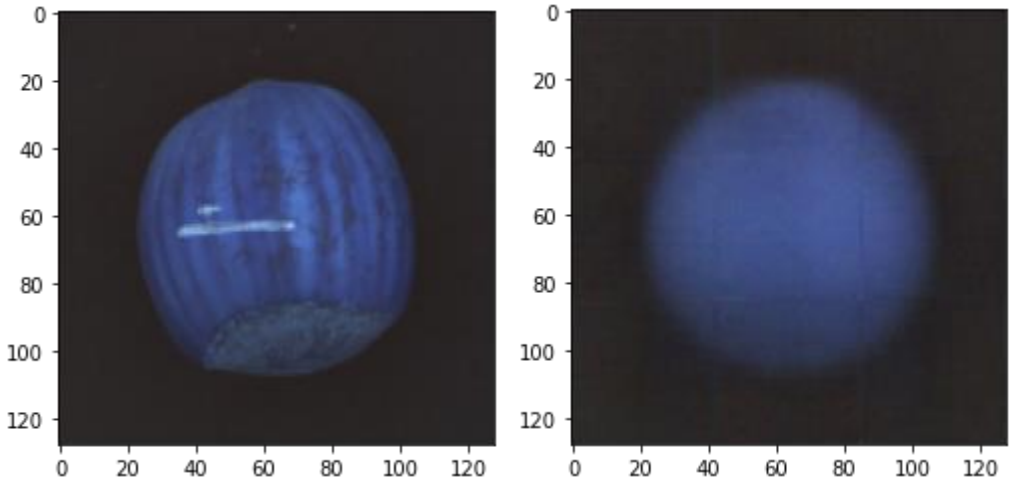

Figure 92

Figures 92 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them

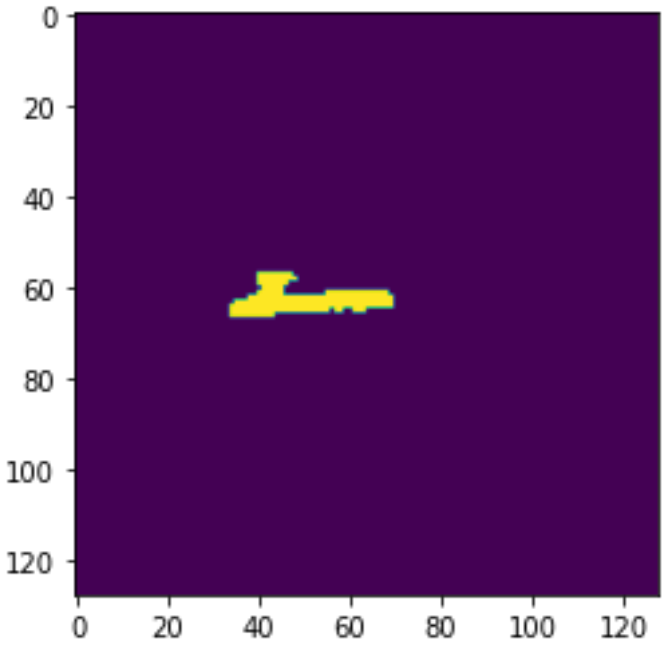

Figure 93

Figure 93 shows the real mask that where the image hast problems and the network is supposed to find them.

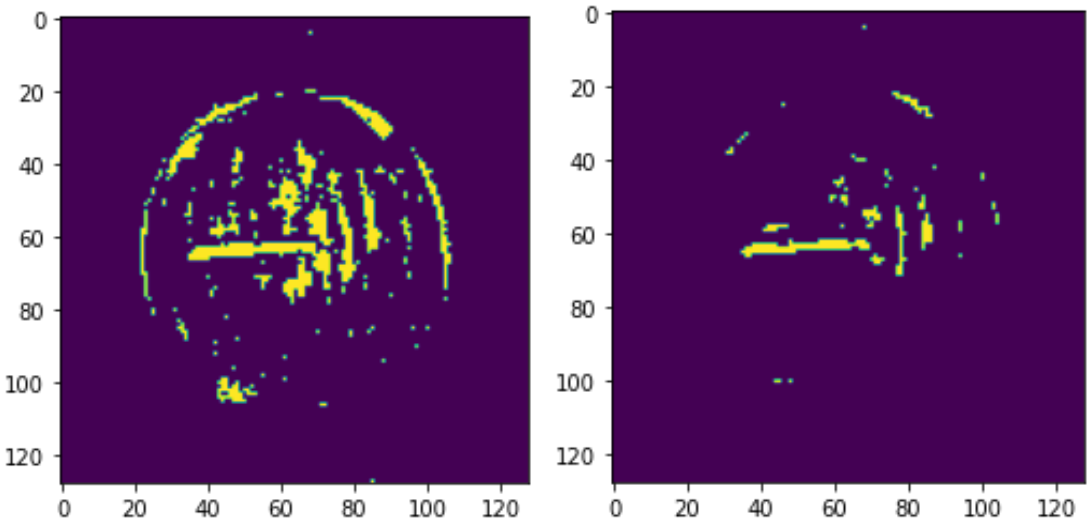

Figure 94

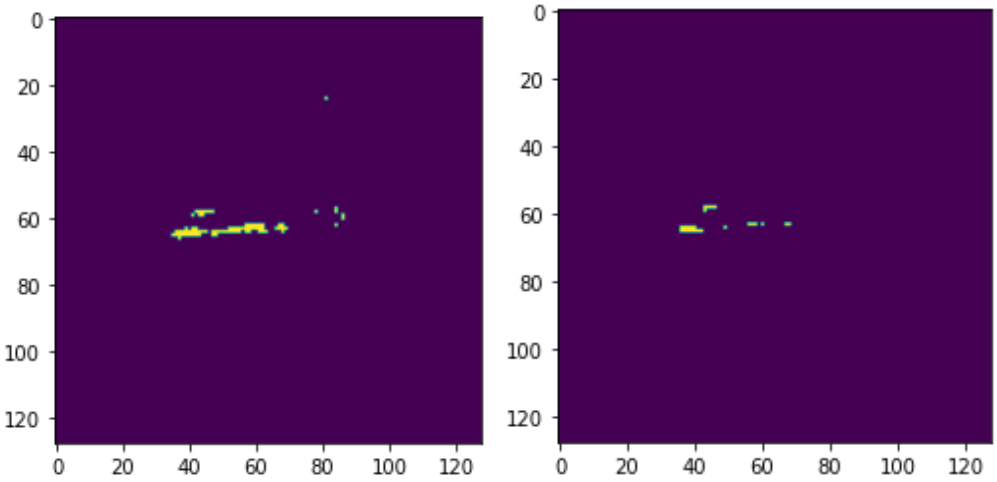

Figure 95

Figure 94 95 from left to right we are using $threshold$ 0.1, 0.15, 0.2, 0.3 for them and we can see the result.

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	69%	53%	38%	13%
Method2	14%	41%	91%	100%
Methdo3	95%	99%	99%	99%

Hole:

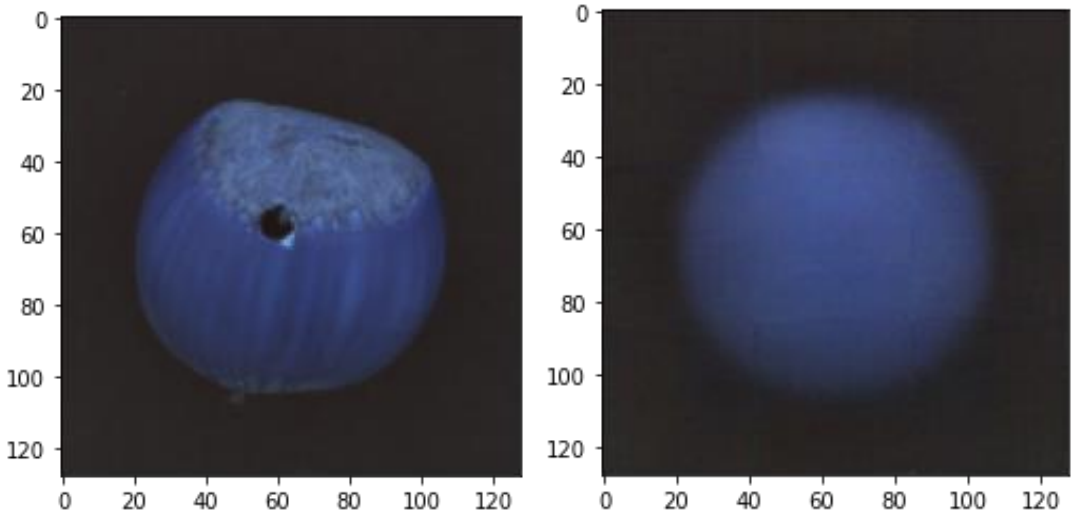

Figure 96

Figures 96 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them.

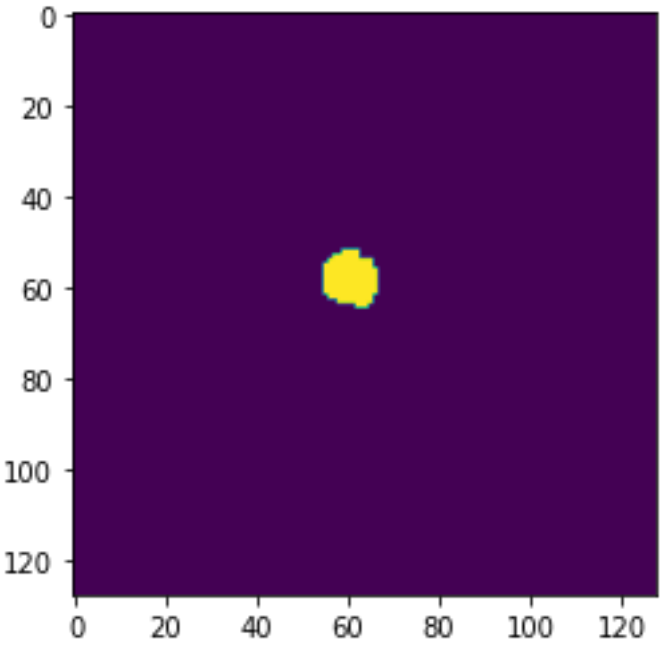

Figure 97

Figure 97 shows the real mask that where the image hast problems and the network is supposed to find them.

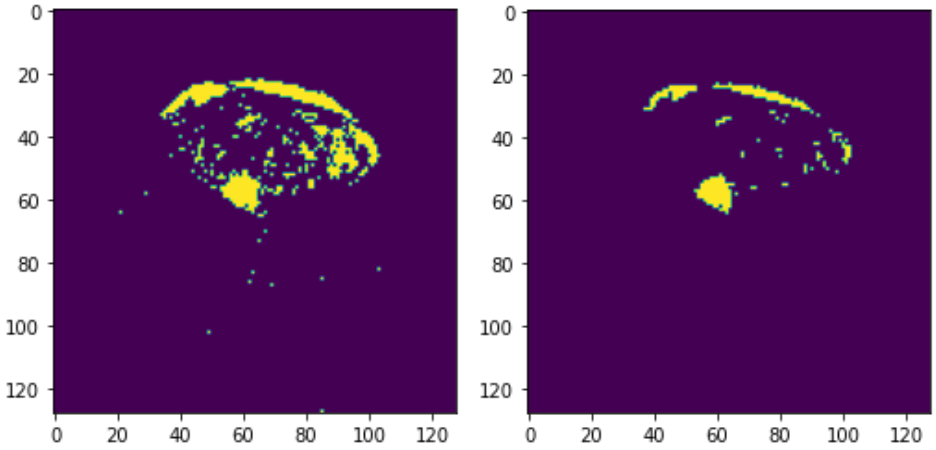

Figure 98

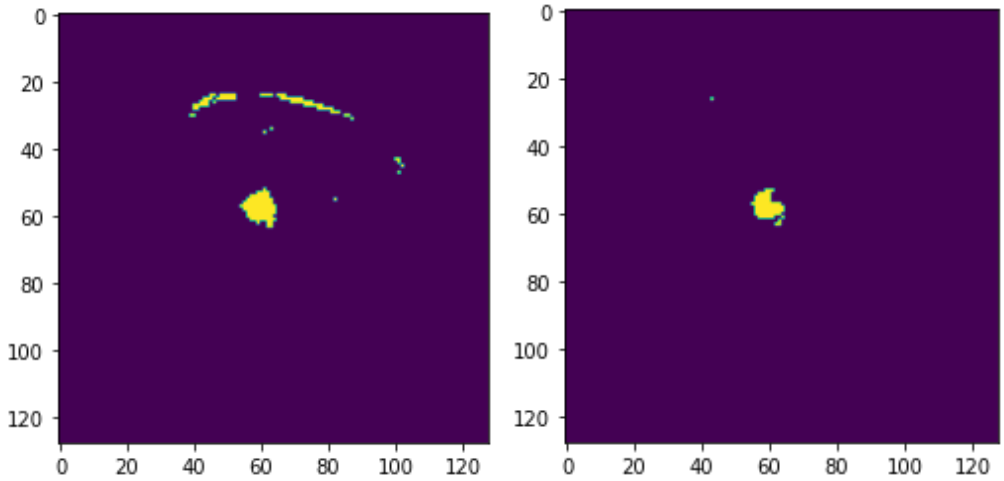

Figure 99

Figure 98 99 from left to right we are using $threshold$ 0.1, 0.15, 0.2, 0.3 for them and we can see the result.

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	79%	73%	63%	49%
Method2	14%	32%	51%	98%
Methdo3	96%	99%	99%	99%

Print:

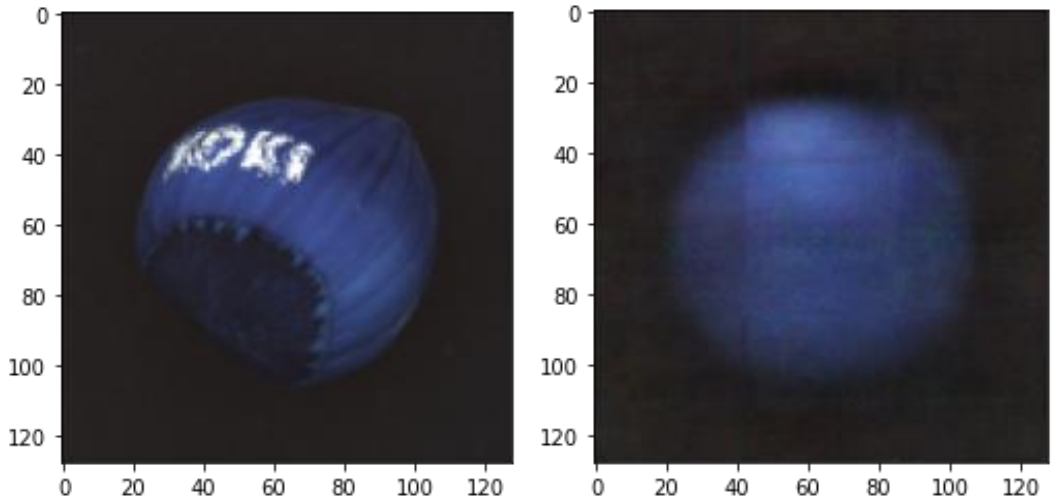

Figure 100

Figures 100 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them.

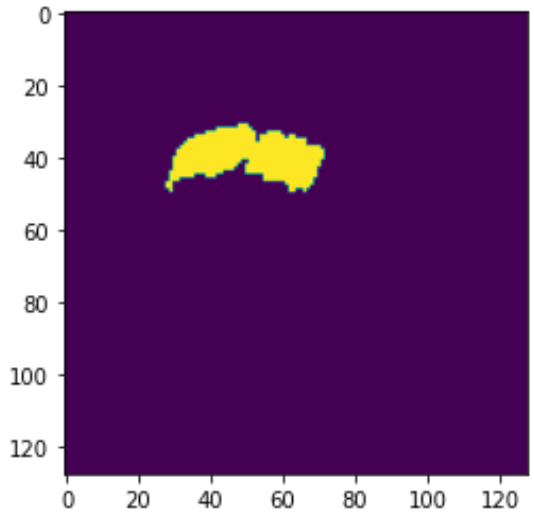

Figure 101

Figure 101 shows the real mask that where the image hast problems and the network is supposed to find them.

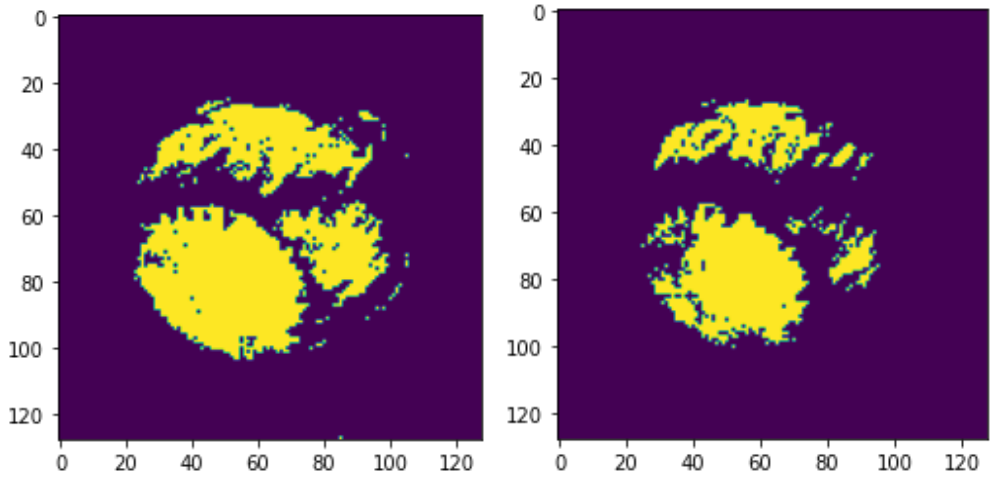

Figure 102

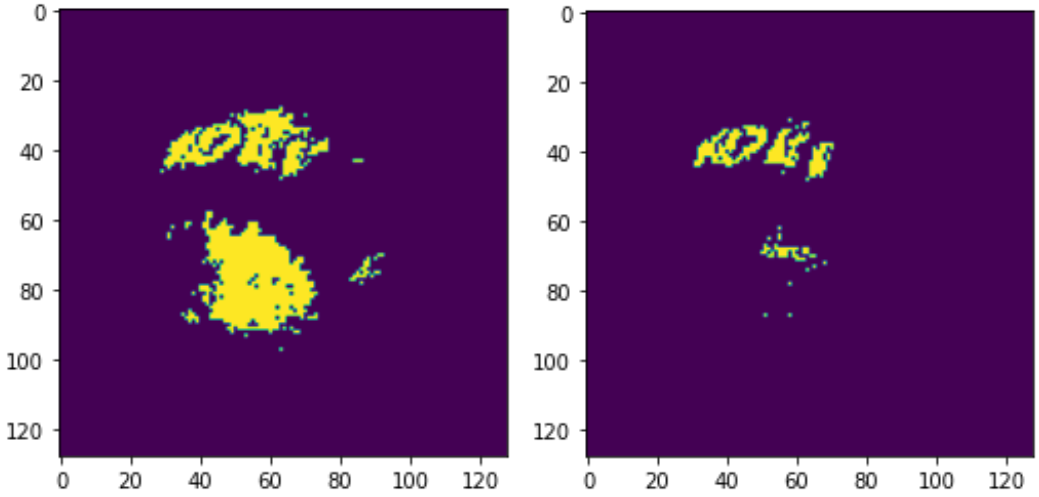

Figure 103

Figure 102 103 from left to right we are using *threshold* 0.1, 0.15, 0.2, 0.3 for them and we can see the result.

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	85%	70%	62%	47%
Method2	13%	18%	27%	82.6%
Methdo3	83%	89%	94%	98%

Part 3) In this part we are going to use a bigger image for the model and for this case we just added convolutional layer in the beginning and the ending of the network.

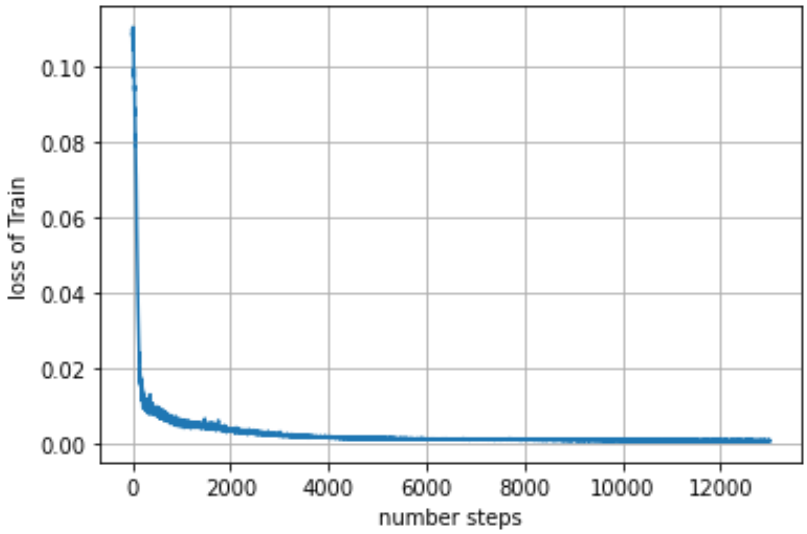

Figure 104

Figure 104 shows the loss of the model we can see that as we expect the loss is decreasing by epochs and because we are using batches the number of steps is $\frac{DataSetSize}{batchSiz} \times Epochs$ that why we see a lot of steps in it.

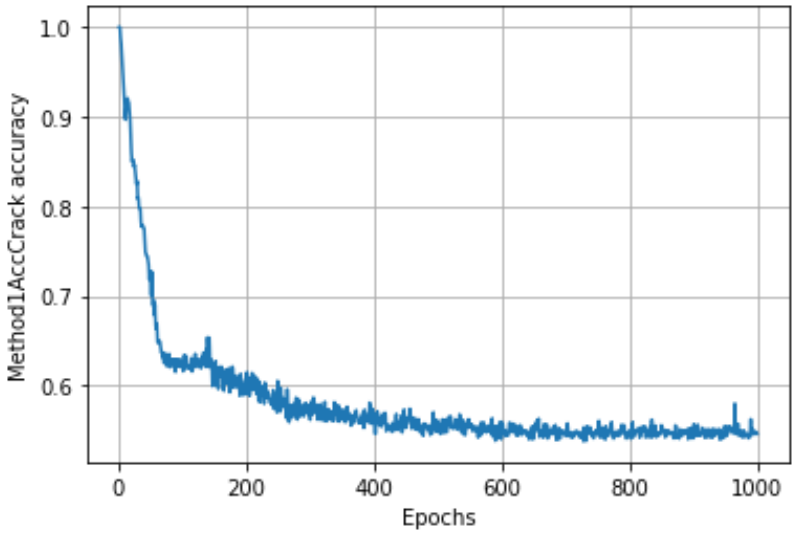

Figure 105

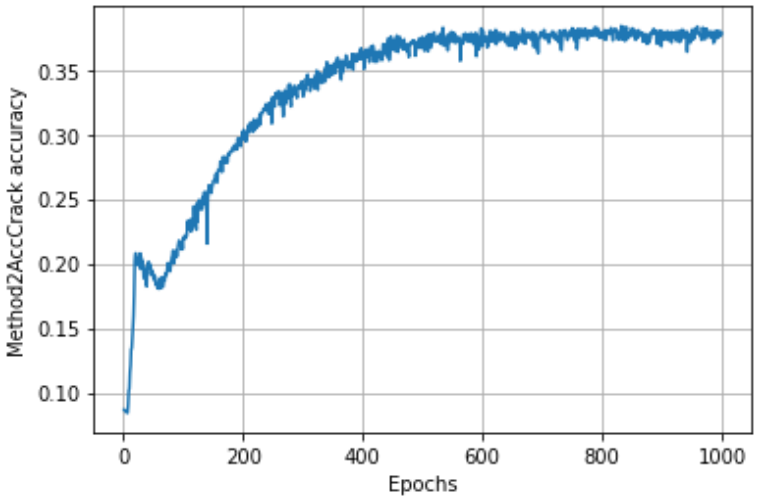

Figure 106

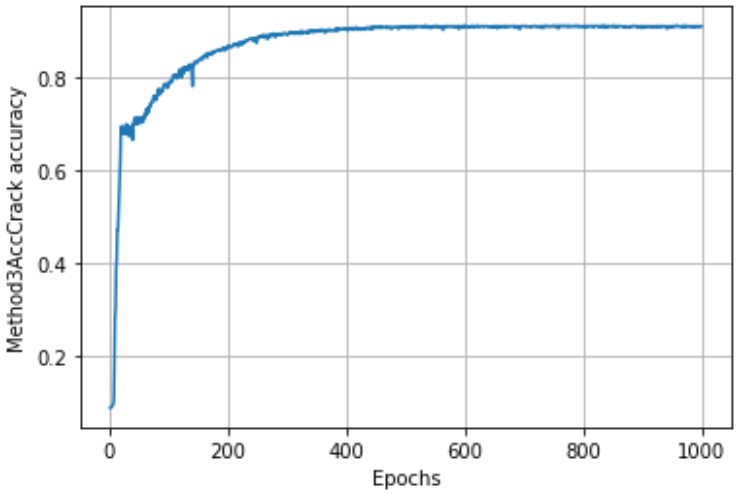

Figure 107

Figure 105 106 107 shows the 3 accuracy that we have and we mentioned every method for the accuracy. The accuracy is for the crack dataset.

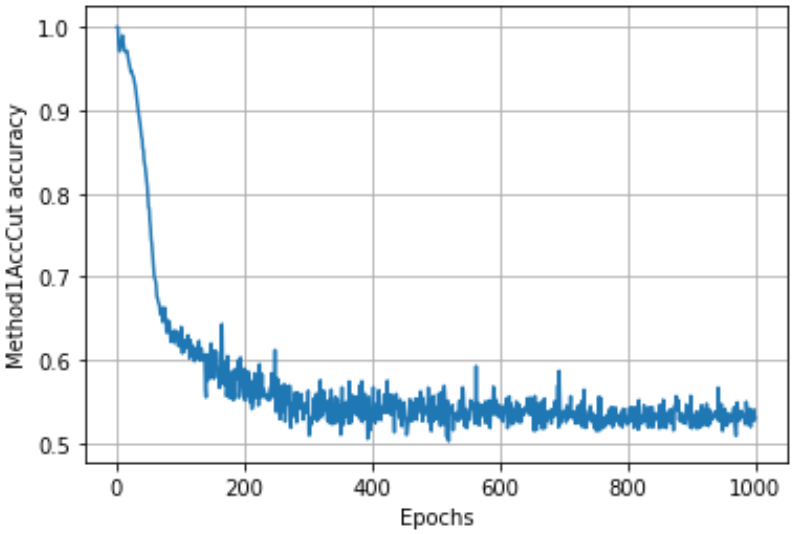

Figure 108

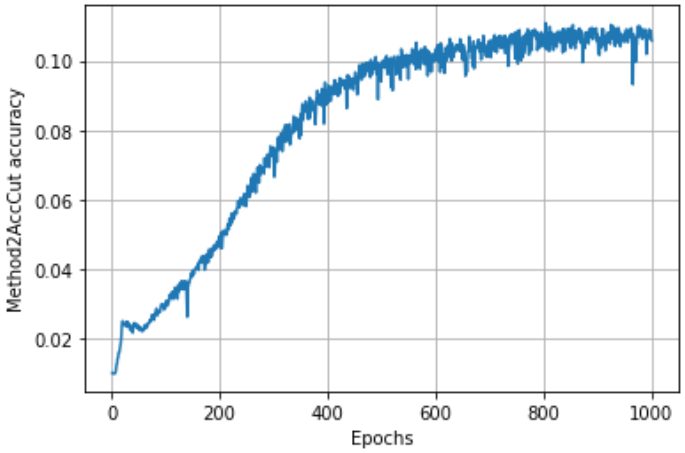

Figure 109

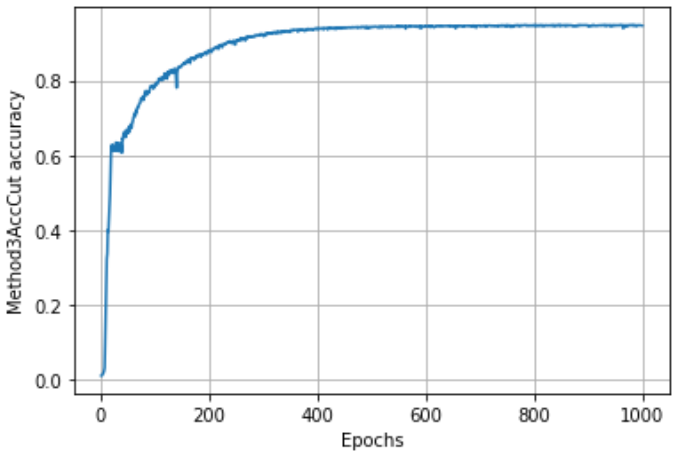

Figure 110

Figures 108 109 110 show the accuracy of the model for the 3 methods that we mentioned, we used the cut dataset

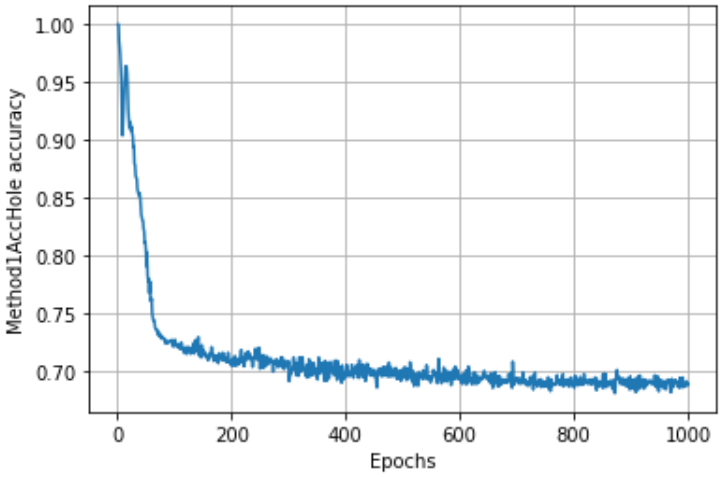

Figure 111

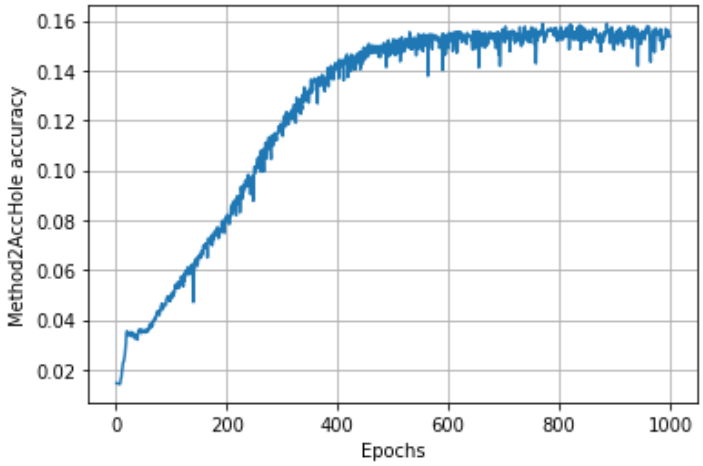

Figure 112

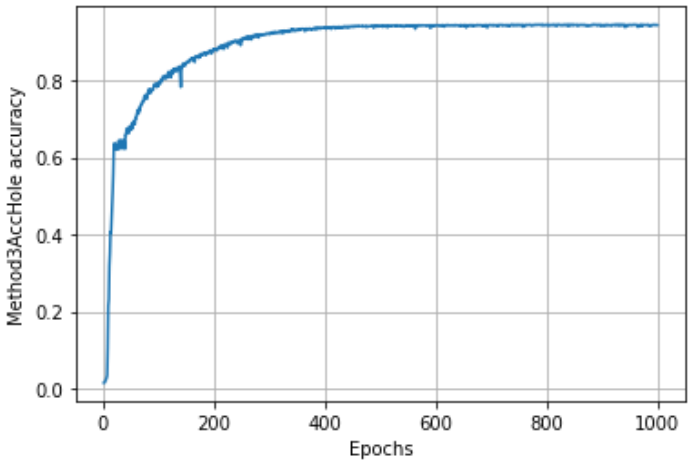

Figure 113

Figures 111 112 113 show the accuracy of the model for the 3 methods that we mentioned, we used the hole dataset

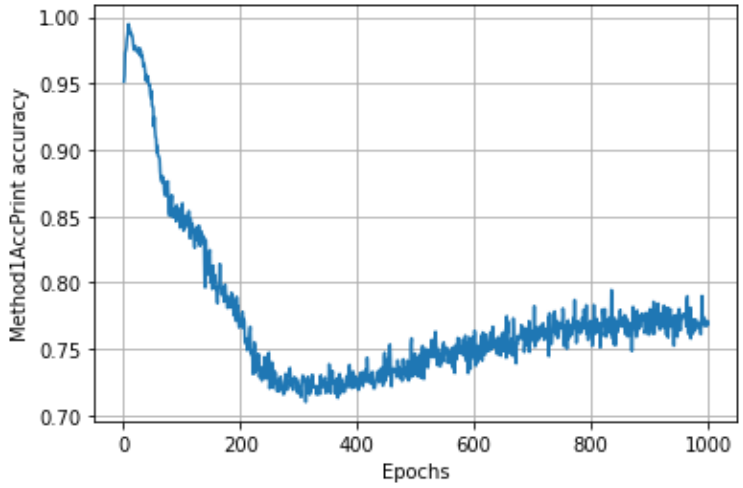

Figure 114

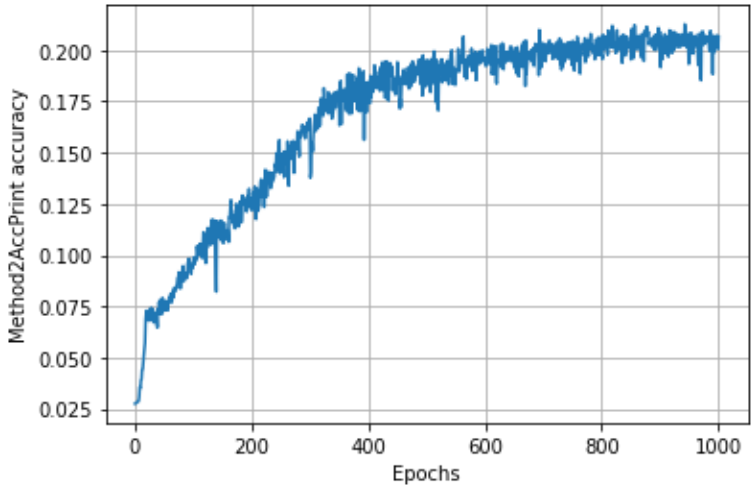

Figure 115

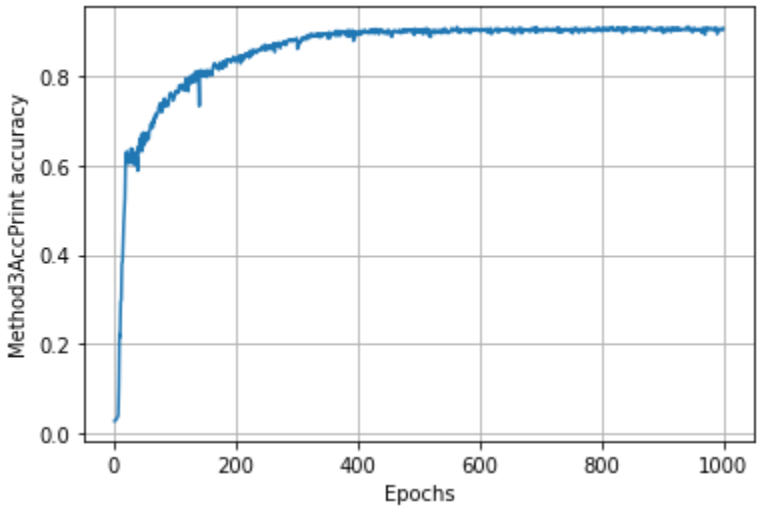

Figure 116

Figure 114 115 116 shows the accuracy with the 3 different methods for the print dataset.

Crack:

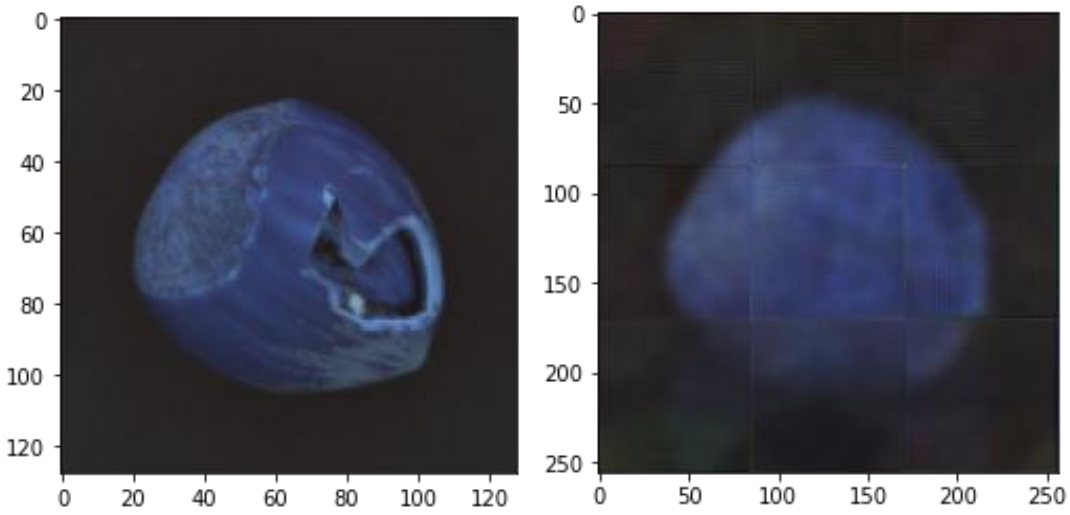

Figure 117

Figures 117 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them

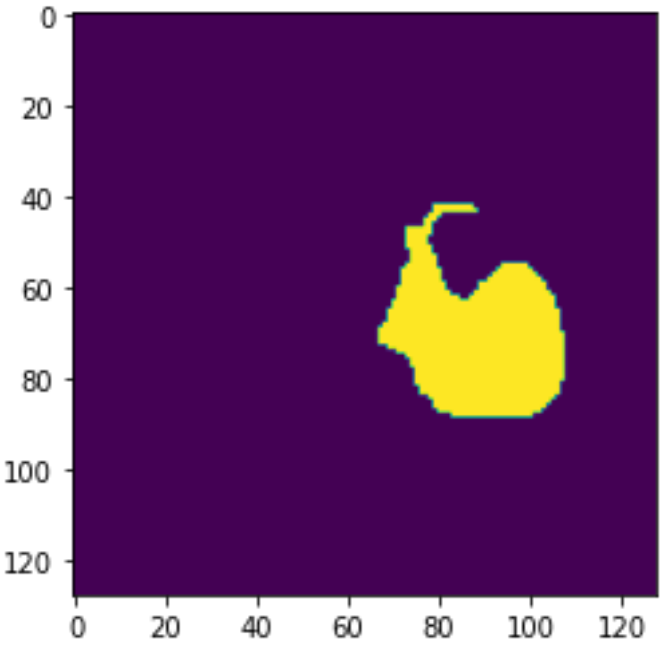

Figure 118

Figure 118 shows the real mask that where the image hast problems and the network is supposed to find them.

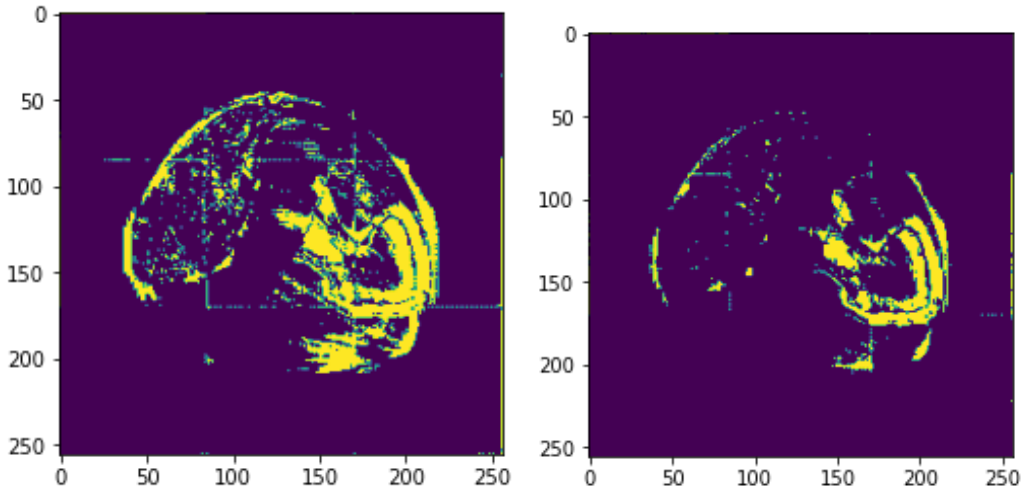

Figure 119

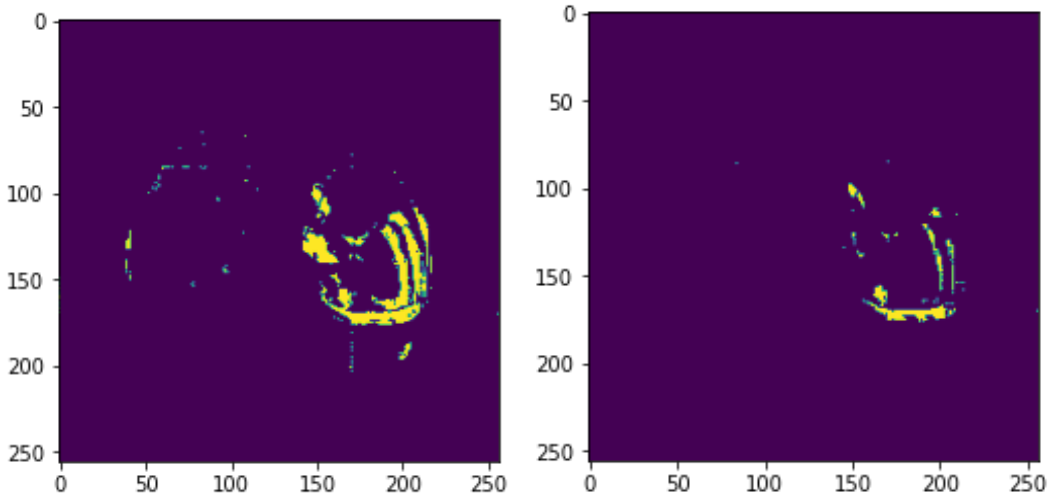

Figure 120

Figure 119 120 from left to right we are using $threshold$ 0.1, 0.15, 0.2, 0.3 for them and we can see the result

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	64%	46%	33%	11%
Method2	40%	64%	86%	99%
Methdo3	91%	95%	95%	94%

Cut:

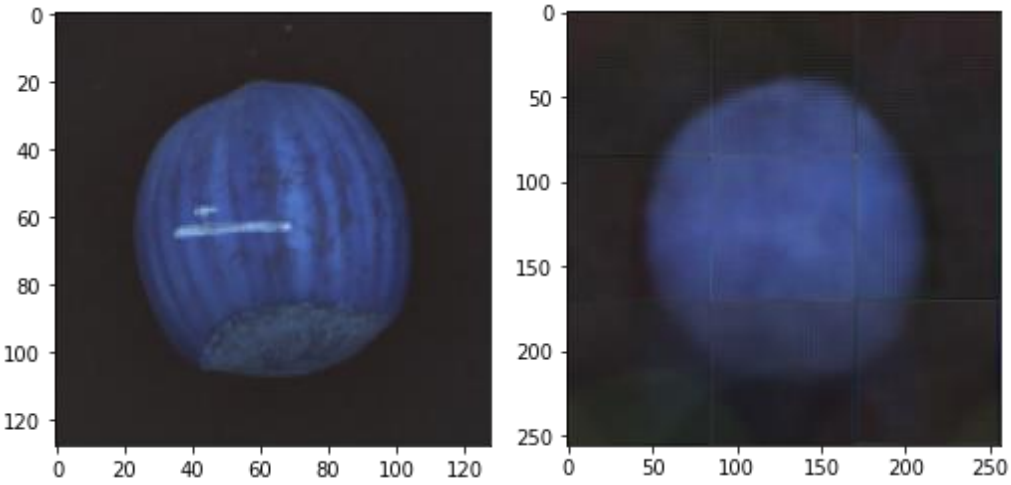

Figure 121

Figures 121 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them

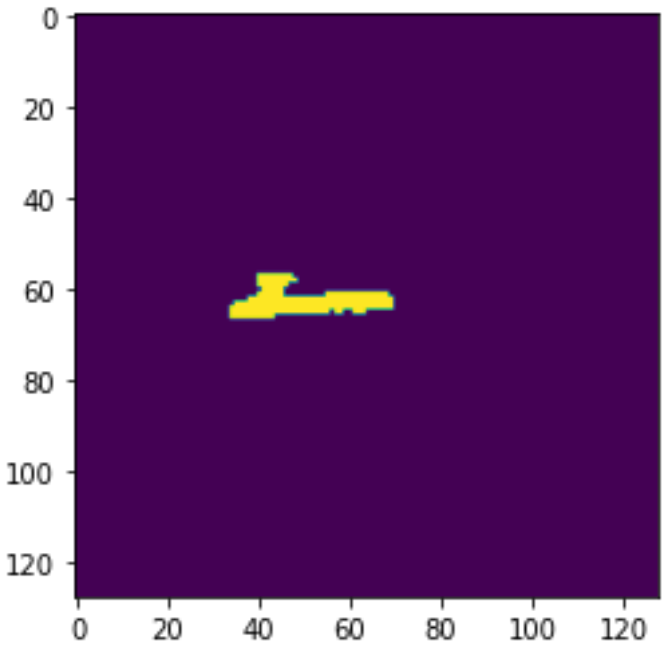

Figure 122

Figure 122 shows the real mask that where the image hast problems and the network is supposed to find them.

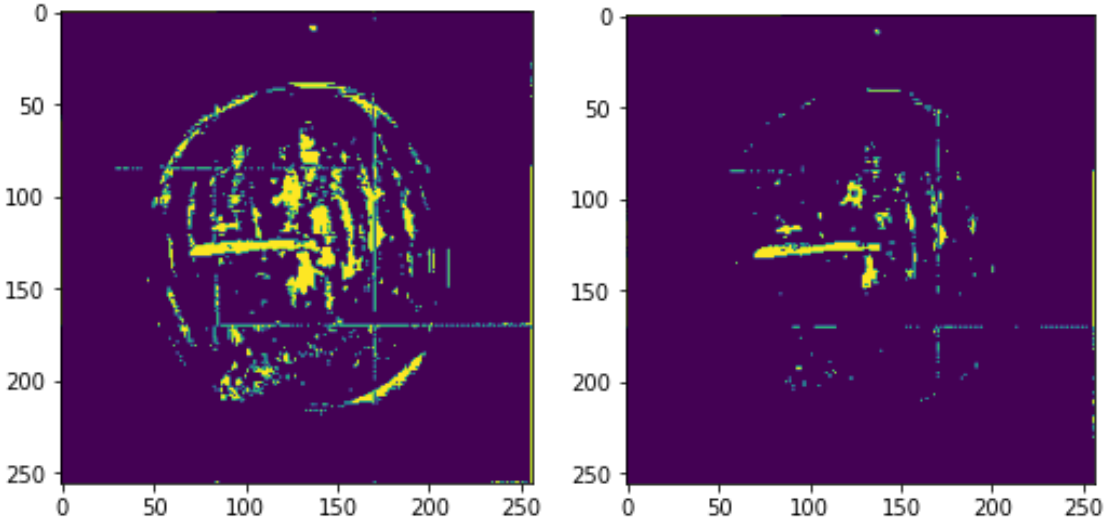

Figure 123

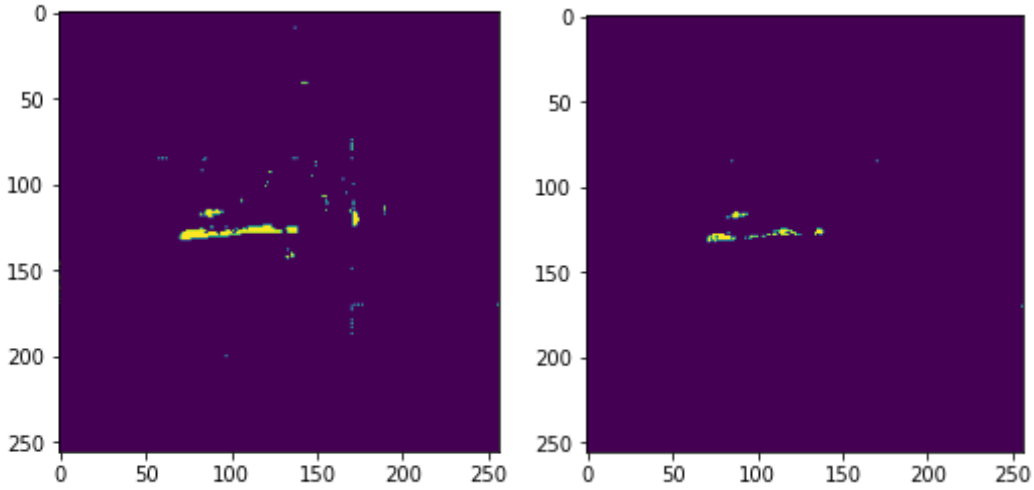

Figure 124

Figure 123 124 from left to right we are using *threshold* 0.1, 0.15, 0.2, 0.3 for them and we can see the result.

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	68%	56%	43%	20%
Method2	10%	26%	71%	98%
Methdo3	93%	98%	99%	99%

Hole:

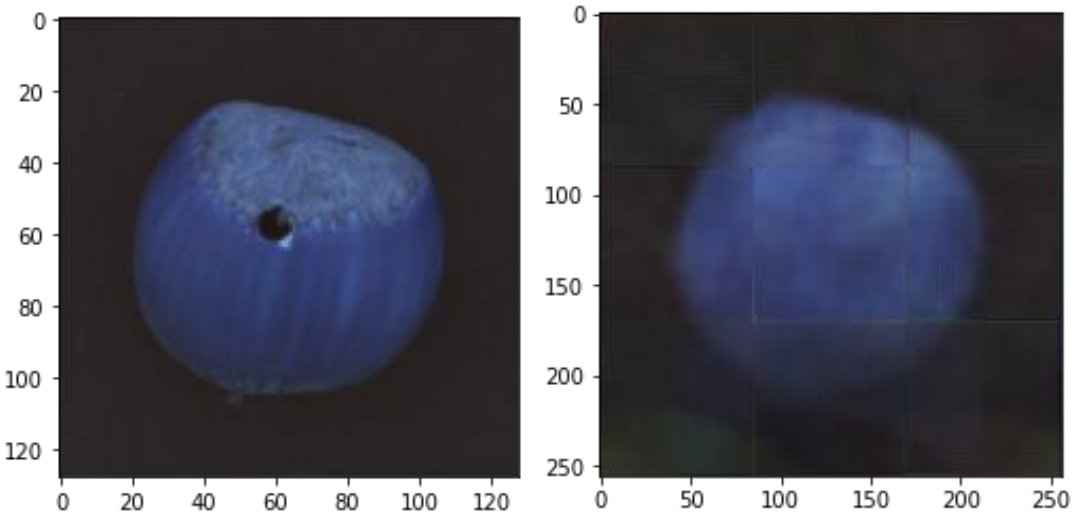

Figure 125

Figures 125 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them.

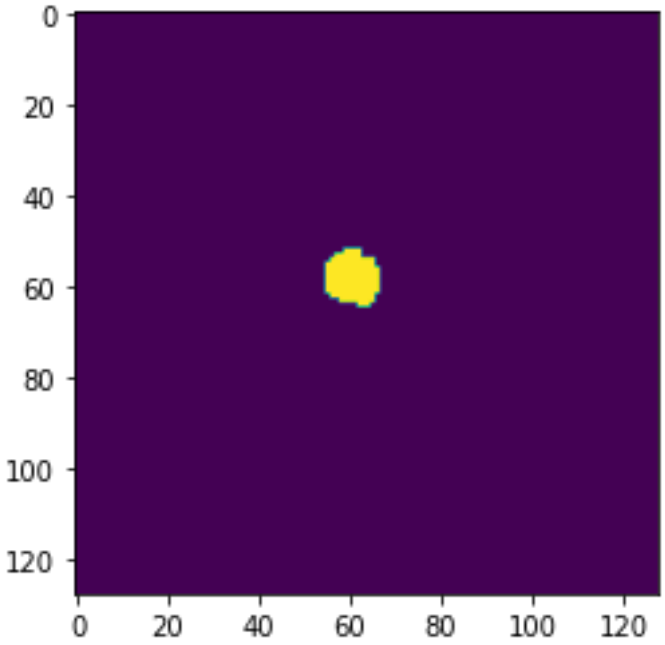

Figure 126

Figure 126 shows the real mask that where the image hast problems and the network is supposed to find them.

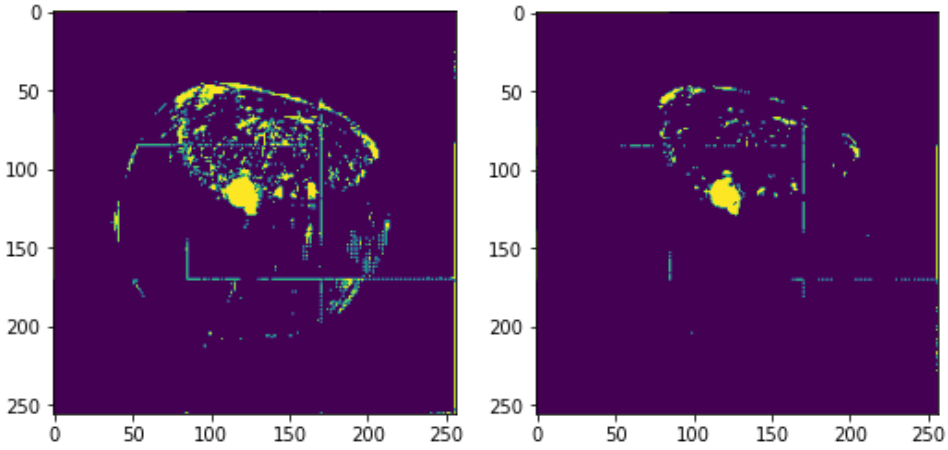

Figure 127

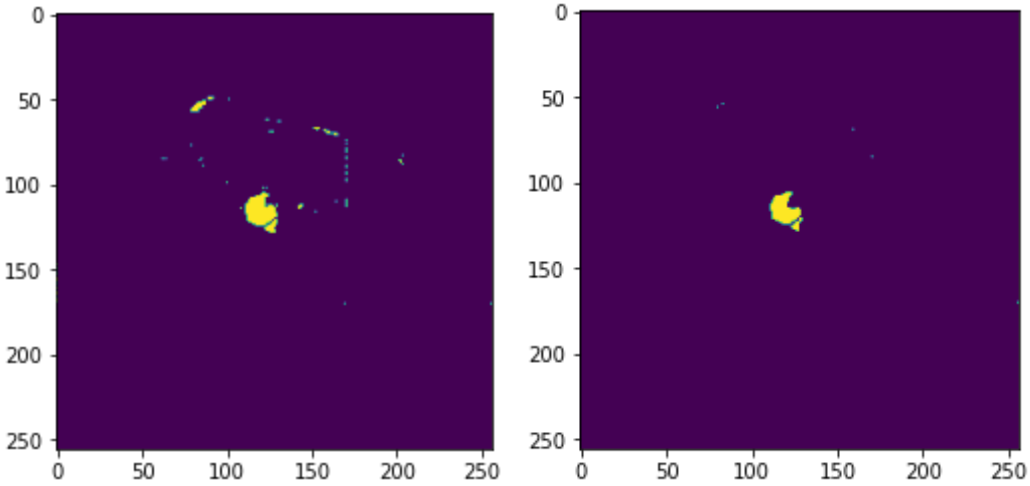

Figure 128

Figure 127 128 from left to right we are using *threshold* 0.1, 0.15, 0.2, 0.3 for them and we can see the result.

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	84%	73%	65%	54%
Method2	12%	30%	70%	98%
Methdo3	96%	99%	99%	99%

Print:

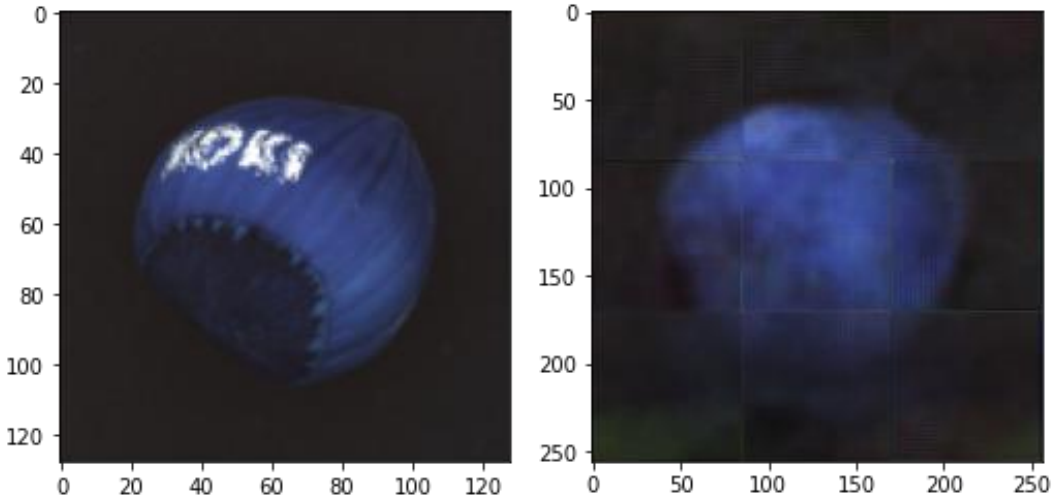

Figure 129

Figures 129 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them.

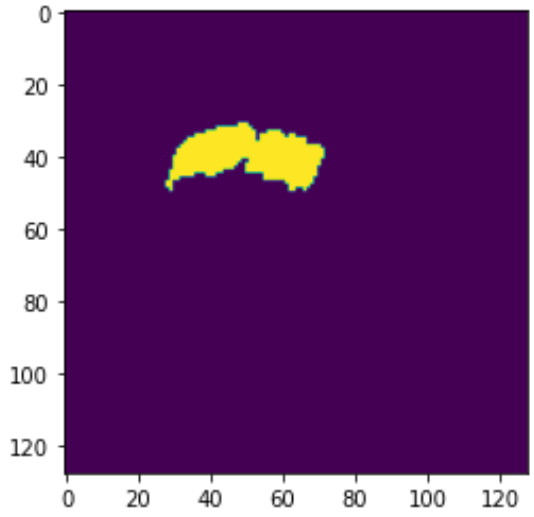

Figure 130

Figure 130 shows the real mask that where the image hast problems and the network is supposed to find them.

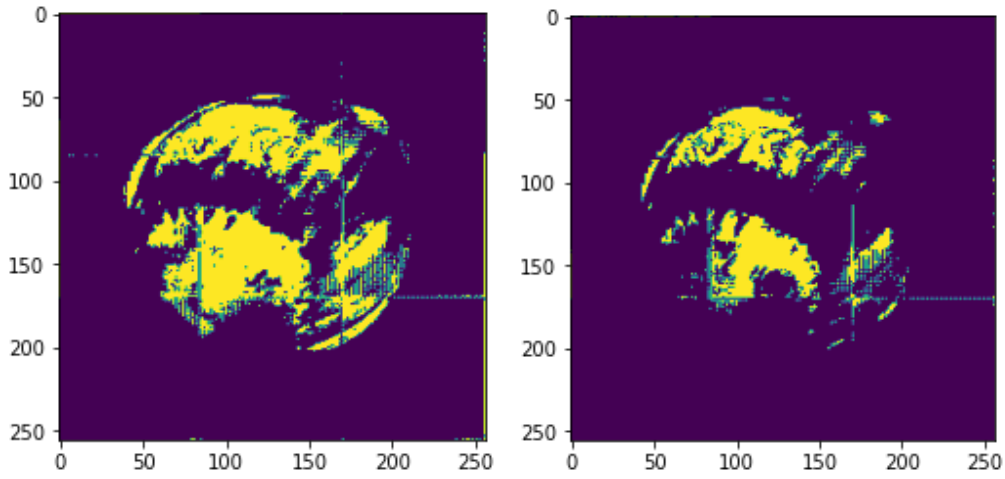

Figure 131

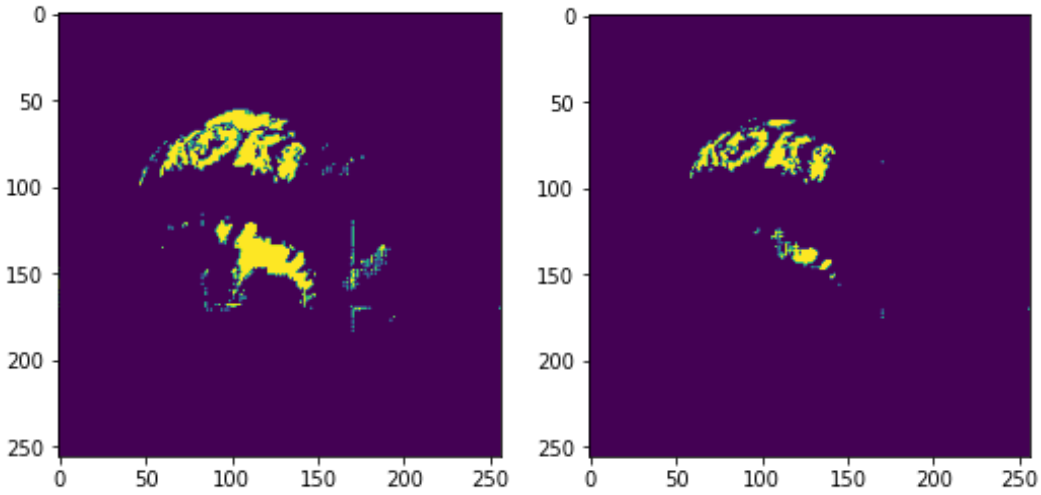

Figure 132

Figure 131 132 from left to right we are using *threshold* 0.1, 0.15, 0.2, 0.3 for them and we can see the result.

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	83%	69%	62%	49%
Method2	15%	25%	43%	76%
Methdo3	87%	93%	97%	98%

Part 4) In this part we use augmentation to increase the number of data's that we have and we are going to use it for our best model that we have reached so far.

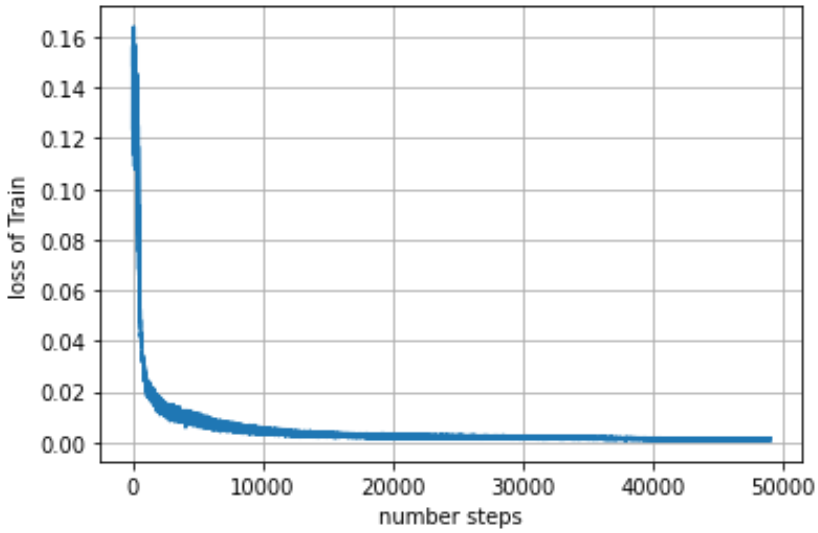

Figure 133

Figure 104 shows the loss of the model we can see that as we expect the loss is decreasing by epochs and because we are using batches the number of steps is $\frac{DataSetSize}{batchSiz} \times Epochs$ that why we see a lot of steps in it.

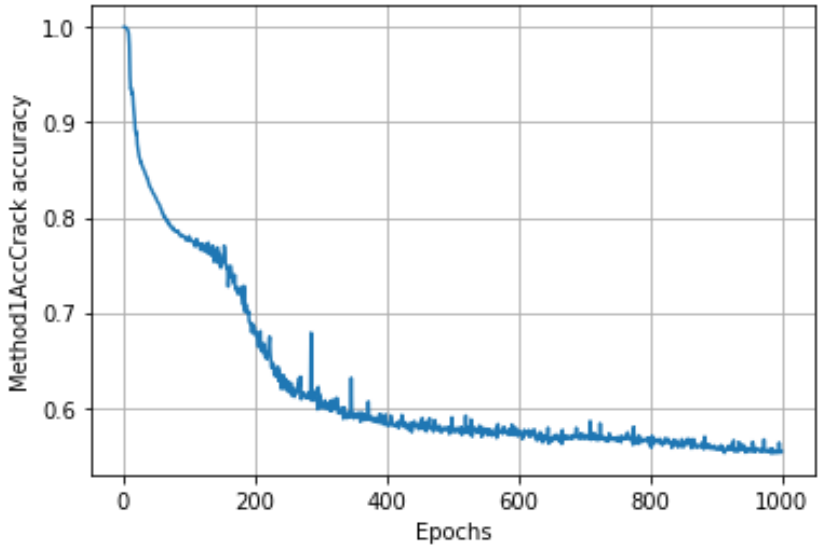

Figure 134

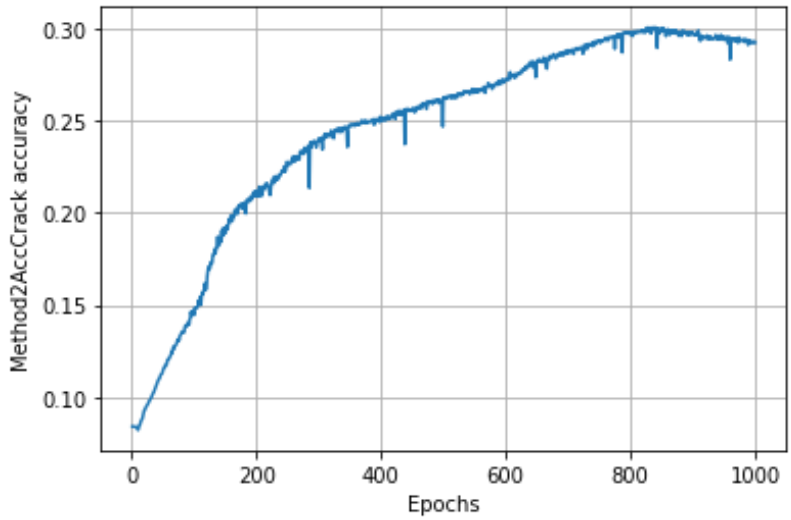

Figure 135

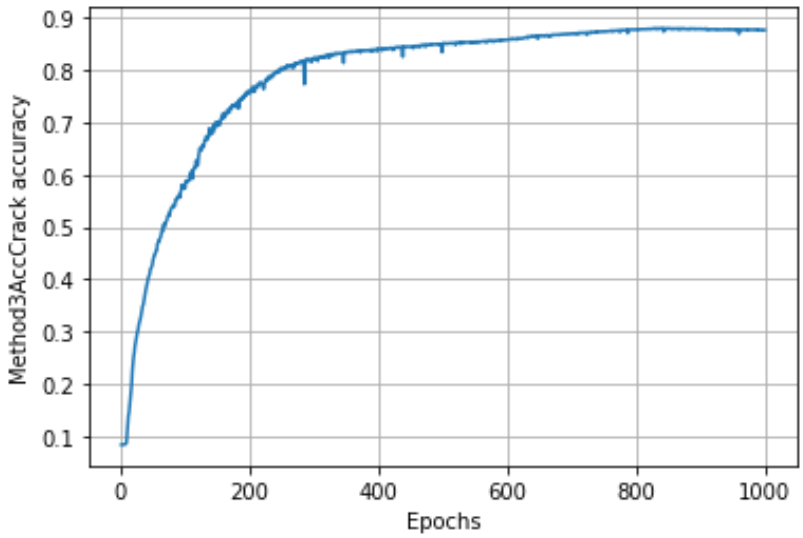

Figure 136

Figure 134 135 136 shows the 3 accuracy that we have and we mentioned every method for the accuracy. The accuracy is for the crack dataset.

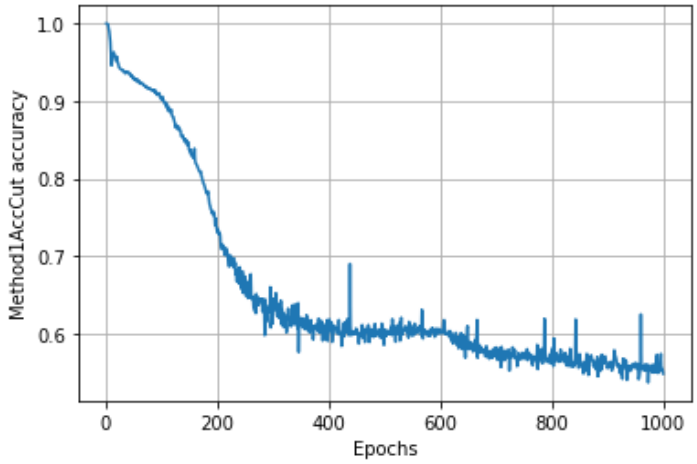

Figure 137

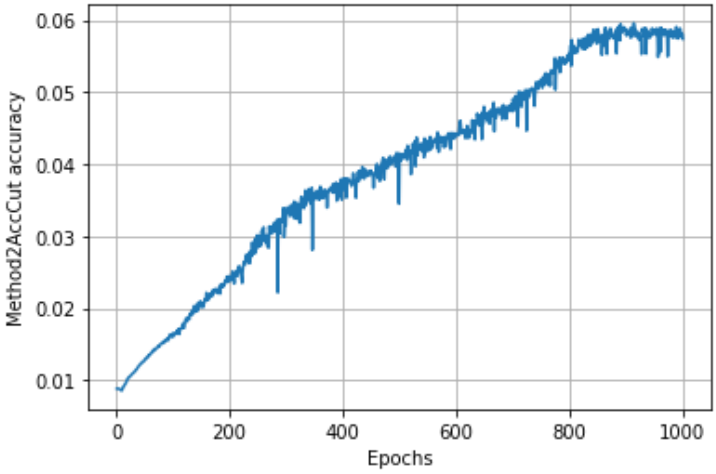

Figure 138

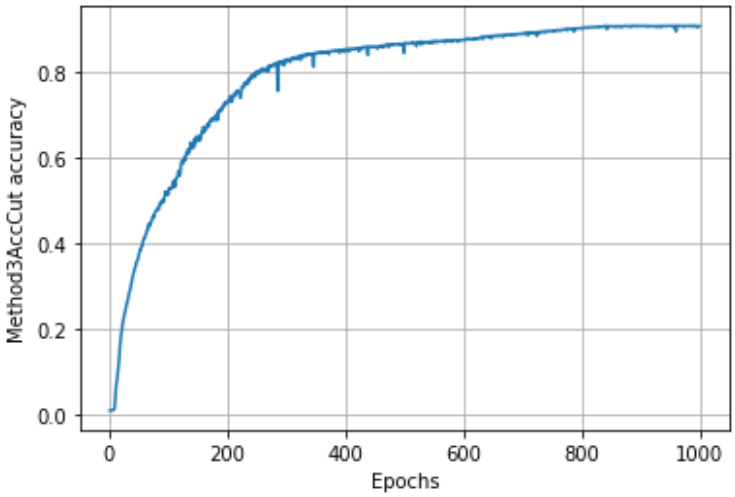

Figure 139

Figures 137 138 139 show the accuracy of the model for the 3 methods that we mentioned, we used the cut dataset

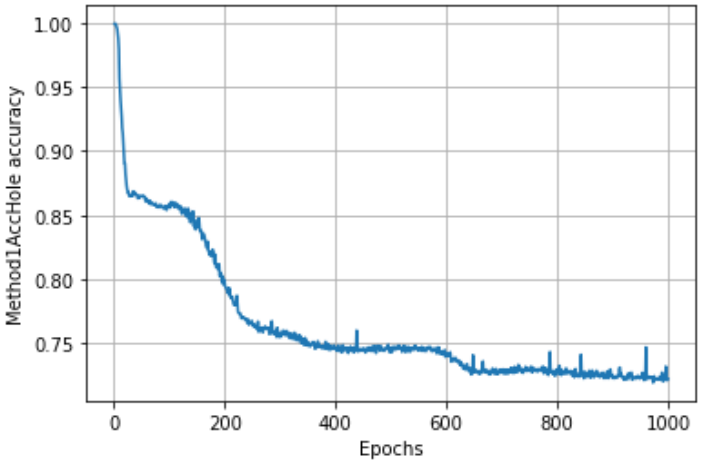

Figure 140

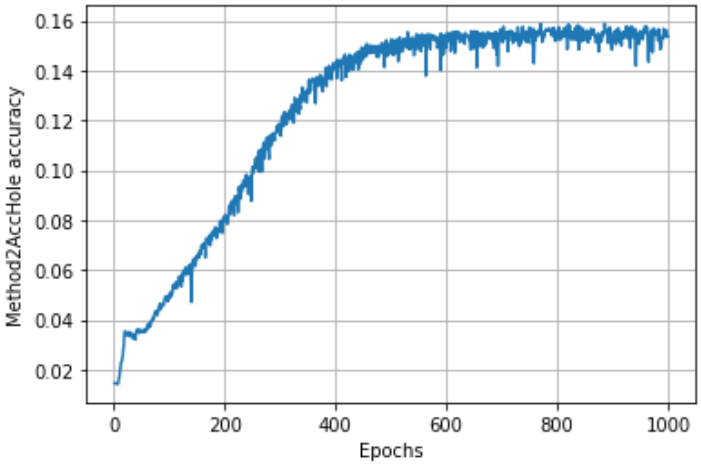

Figure 141

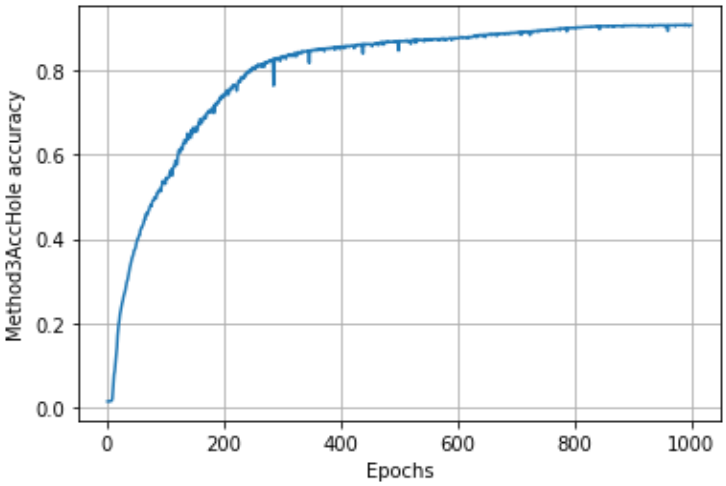

Figure 142

Figures 140 141 142 show the accuracy of the model for the 3 methods that we mentioned, we used the hole dataset

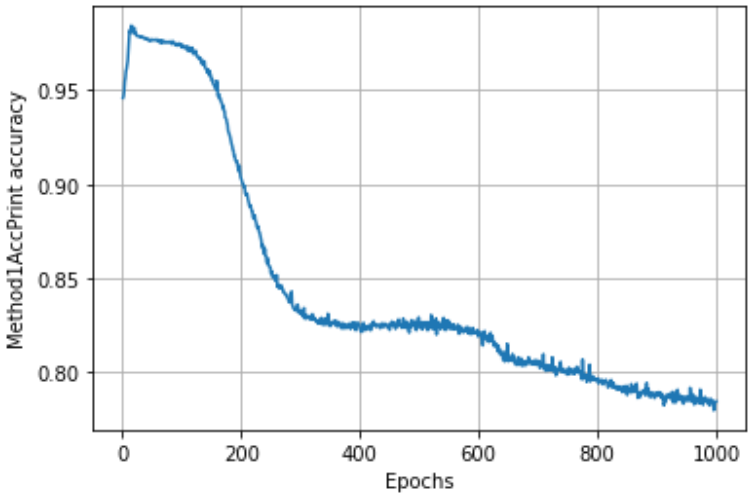

Figure 143

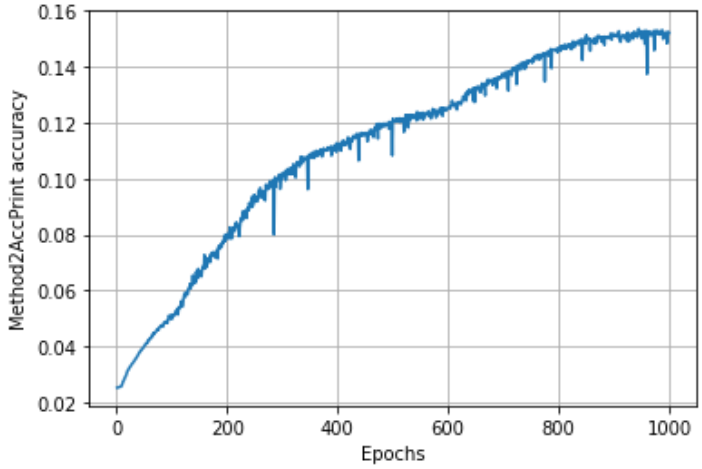

Figure 144

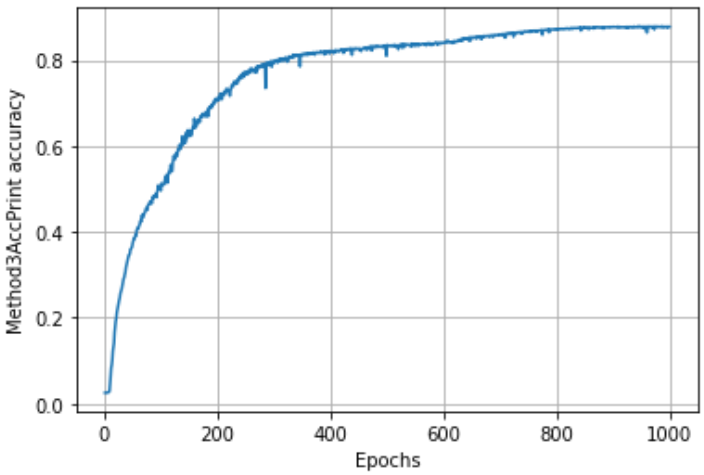

Figure 145

Figure 143 144 145 shows the accuracy with the 3 different methods for the print dataset.

Crack:

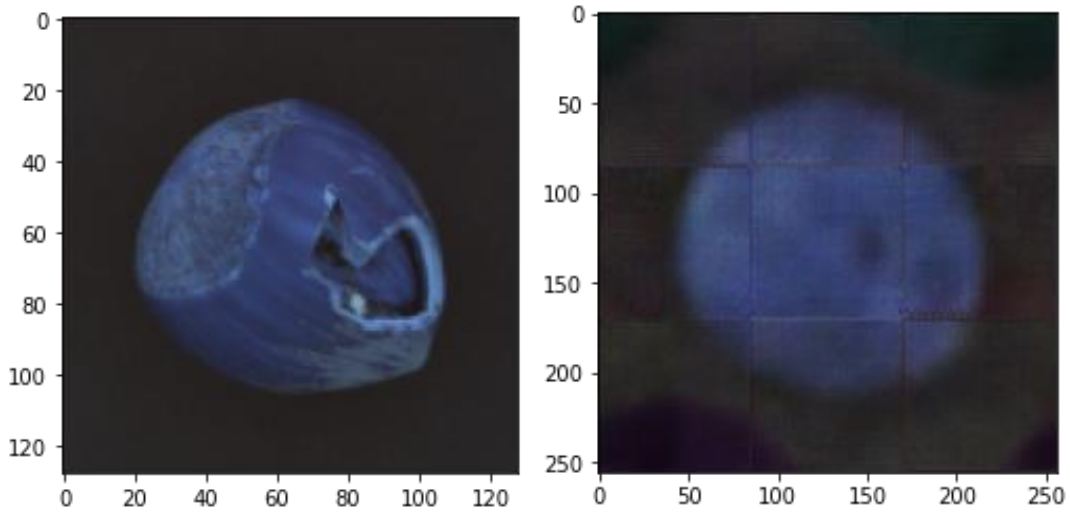

Figure 146

Figures 146 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them

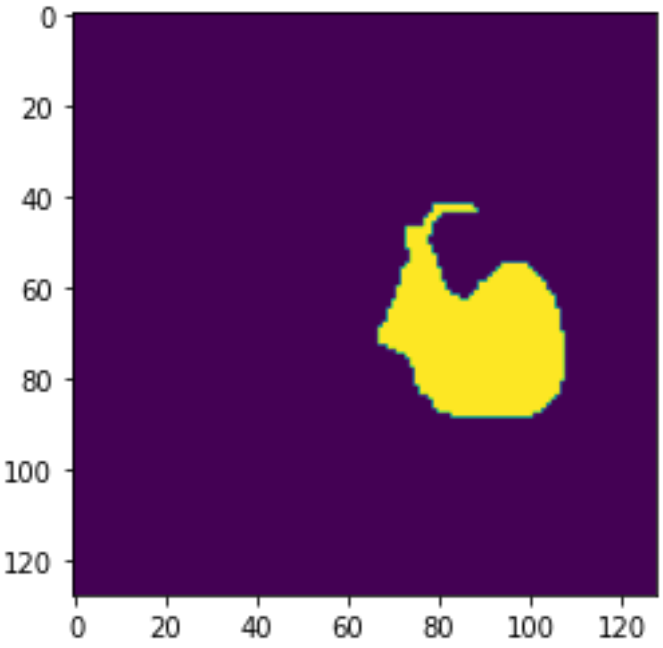

Figure 147

Figure 147 shows the real mask that where the image hast problems and the network is supposed to find them.

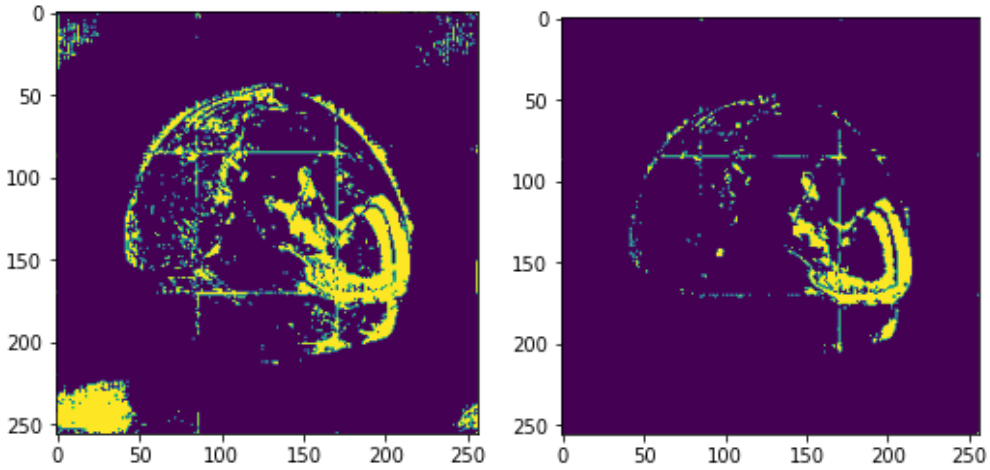

Figure 148

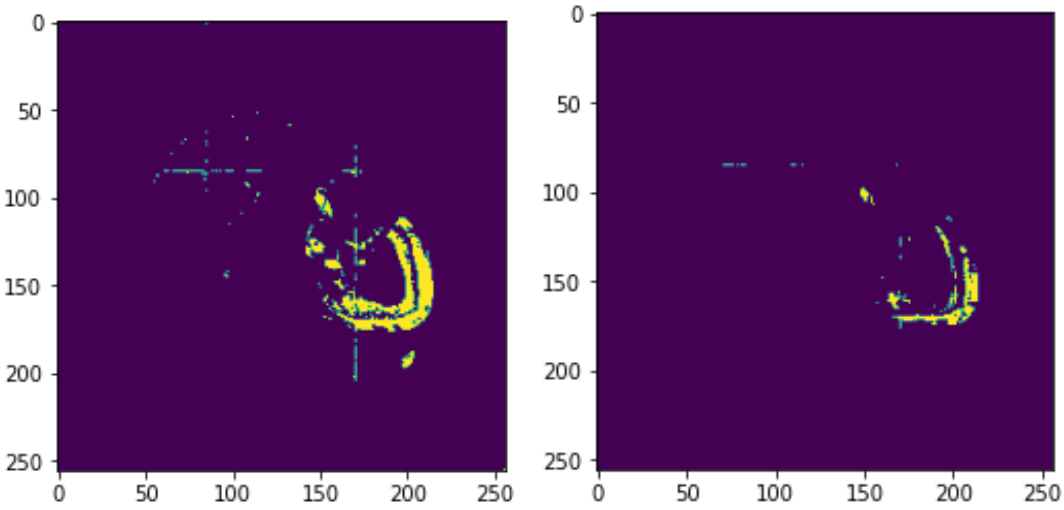

Figure 149

Figure 148 149 from left to right we are using $threshold$ 0.1, 0.15, 0.2, 0.3 for them and we can see the result

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	61%	47%	34%	12%
Method2	34%	71%	90%	97%
Methdo3	90%	95%	96%	94%

Cut:

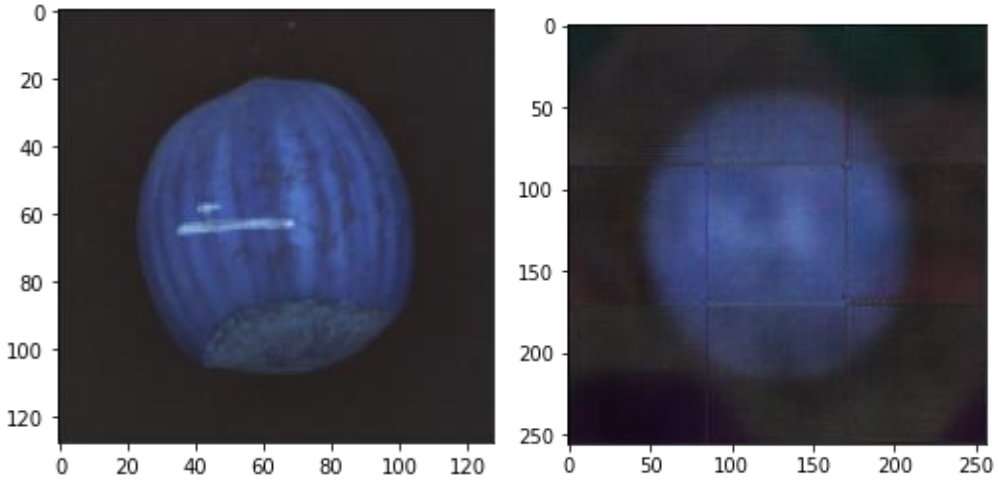

Figure 150

Figures 150 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them

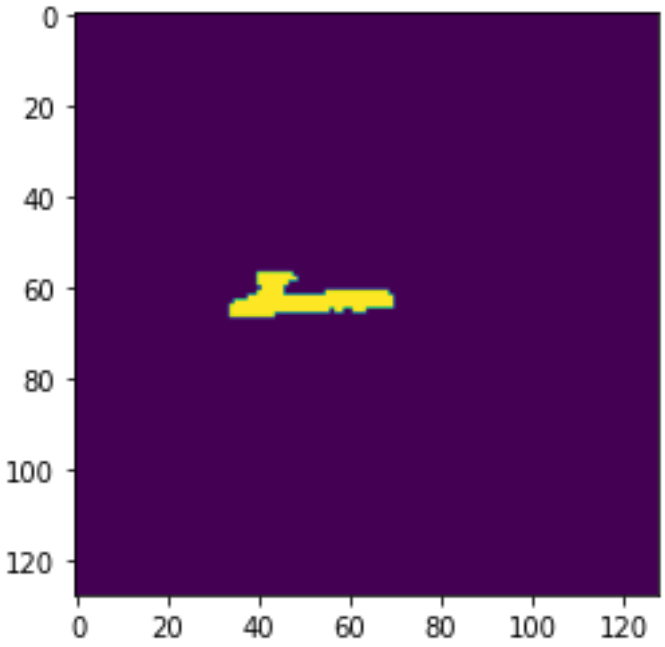

Figure 151

Figure 151 shows the real mask that where the image hast problems and the network is supposed to find them.

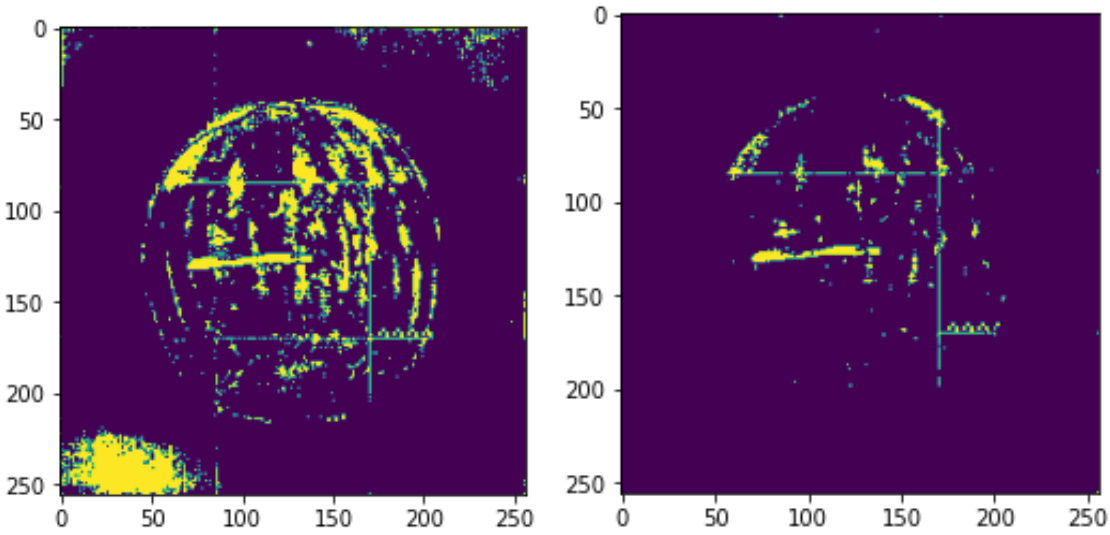

Figure 152

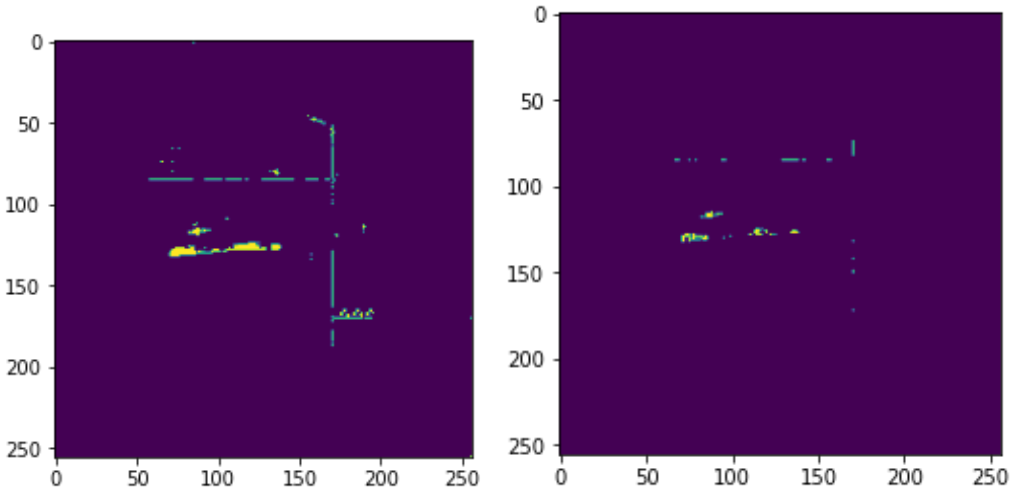

Figure 153

Figure 152 153 from left to right we are using *threshold* 0.1, 0.15, 0.2, 0.3 for them and we can see the result.

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	68%	53%	42%	16%
Method2	5%	23%	47%	72%
Methdo3	89%	98%	99%	99%

Hole:

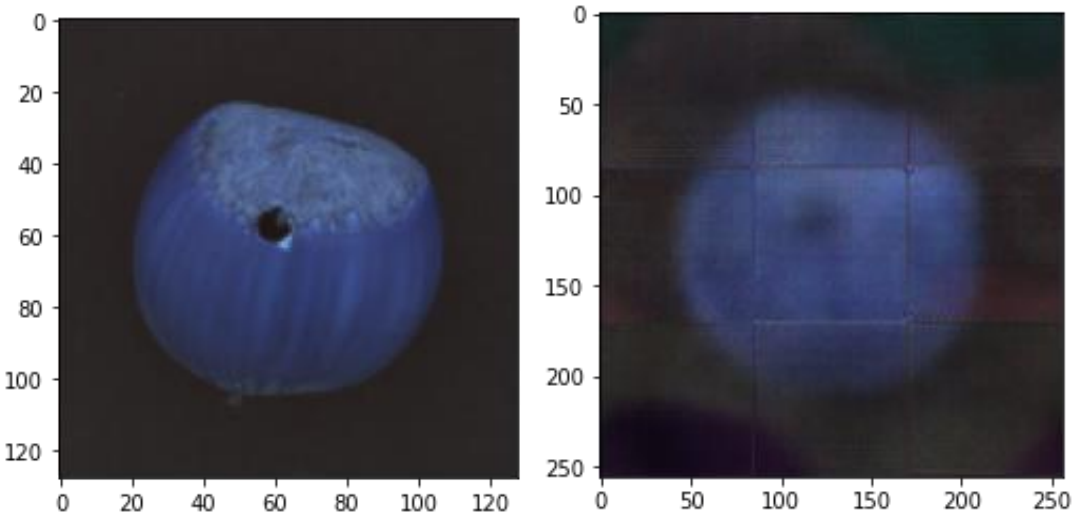

Figure 154

Figures 125 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them.

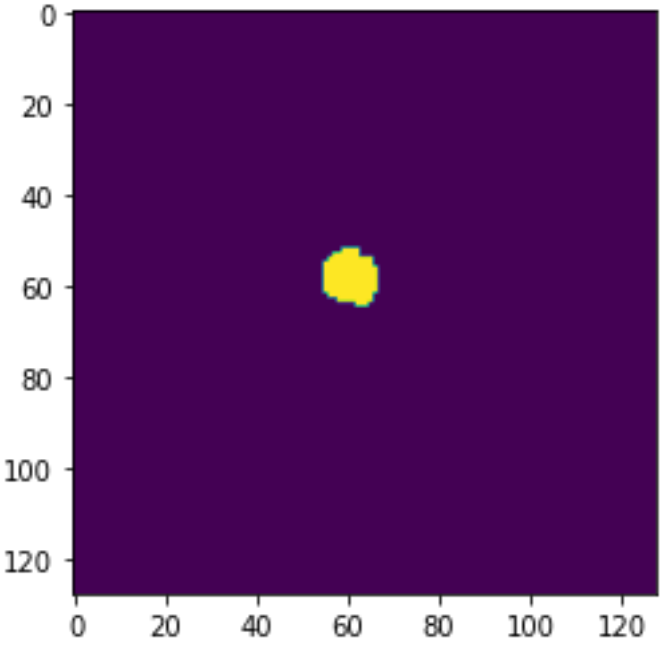

Figure 157

Figure 157 shows the real mask that where the image hast problems and the network is supposed to find them.

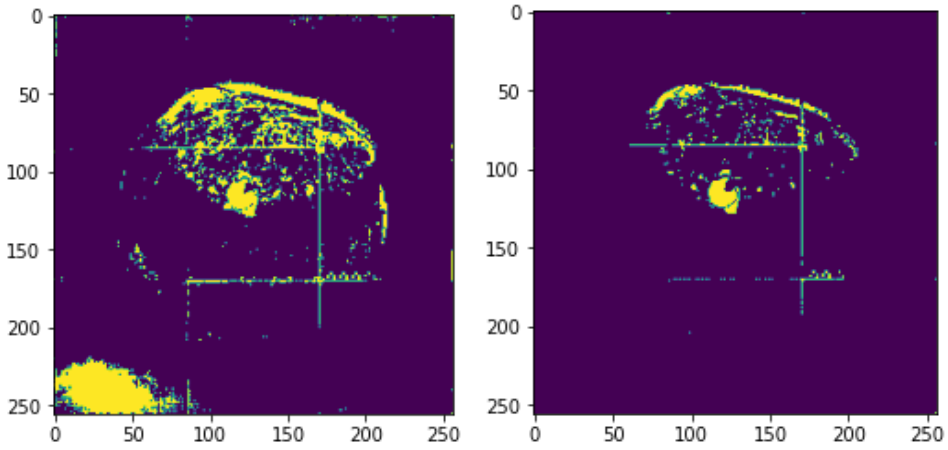

Figure 158

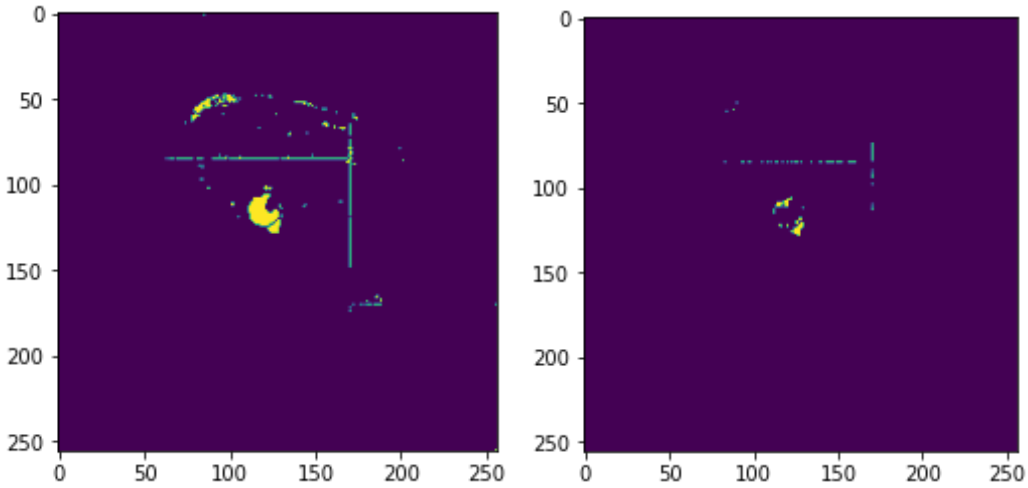

Figure 159

Figure 127 128 from left to right we are using *threshold* 0.1, 0.15, 0.2, 0.3 for them and we can see the result.

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	77%	68%	61%	17%
Method2	5%	18%	39%	56%
Methdo3	91%	98%	99%	99%

Print:

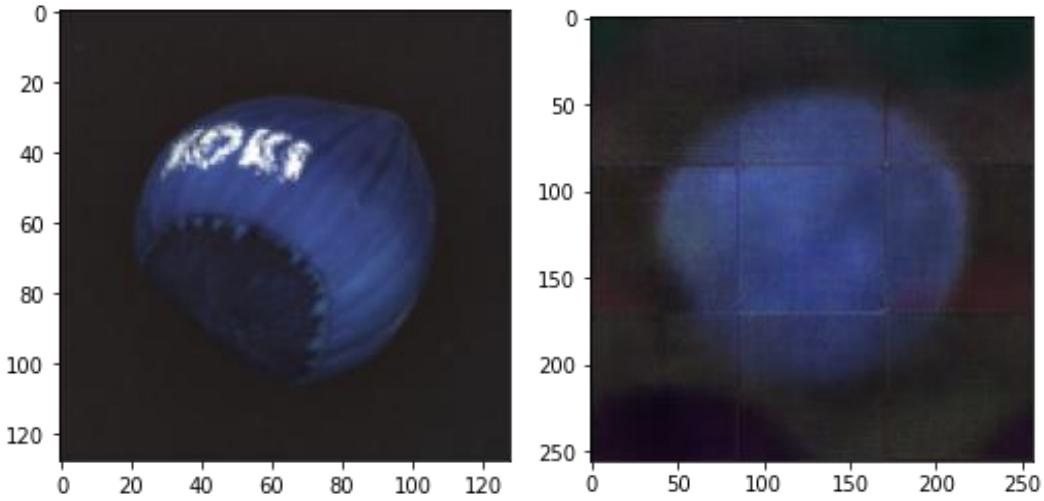

Figure 160

Figures 129 show the input and output of the model we can see that it predicts it will be round but it still has problems with the edges and we need to try to fix them.

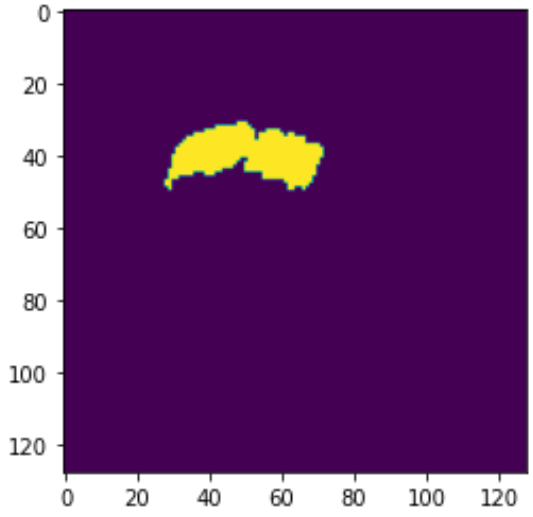

Figure 161

Figure 130 shows the real mask that where the image hast problems and the network is supposed to find them.

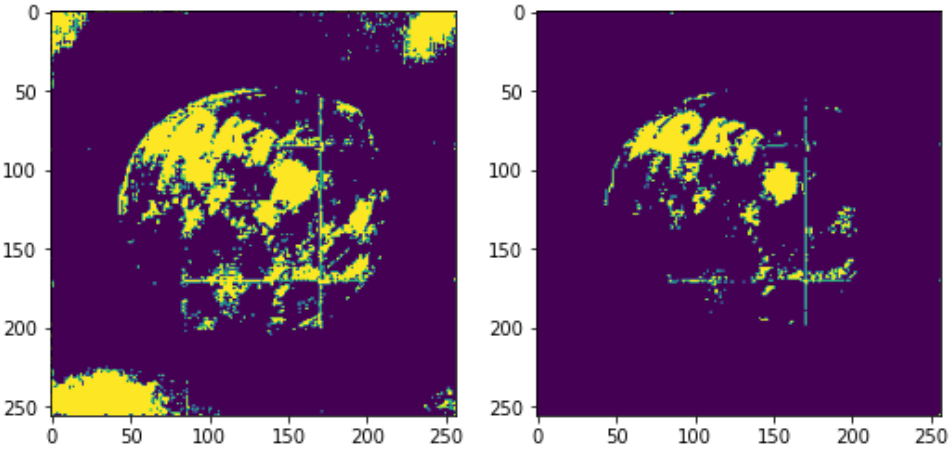

Figure 162

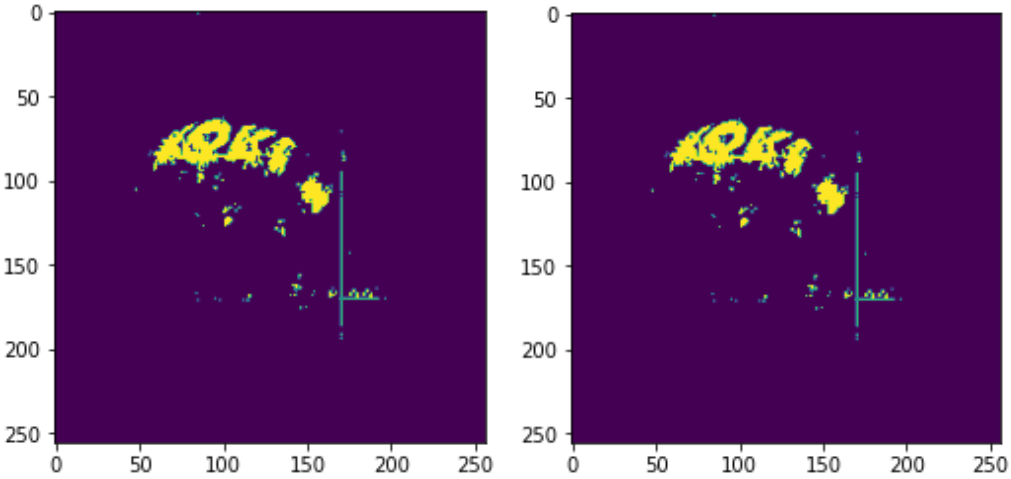

Figure 163

Figure 131 132 from left to right we are using $threshold$ 0.1, 0.15, 0.2, 0.3 for them and we can see the result.

Accuracy/Threshold	0.1	0.15	0.2	0.3
Method1	91%	84%	77%	77%
Method2	15%	42%	43%	67%
Methdo3	87%	96%	97%	98%

Overall we saw the accuracy for the 3 different methods and for all the 4 ways that the question wanted us to do, and by using the tables we can see the accuracy for a single image not all the images which isn't that much accurate but still very good to compare the result and see what happens and in the last there is a not that we increase the number of neurons in the bottle neck we increased it from 100 to 500 and that help our system why? Because when we are using a bigger image size data and we want to convert it to a space with 100 dimension

and that is hard so making it become to 500 is quite better and we got the permission from the TA.

Process

In this HW we learned the structure of the Alex Net DNN and we also found out the equation of input and output size of the data depending of the stride, kernel and padding size and overall saw the with having a small amount of data will result as over fitting and we need to augment the data so we have better generalization and we also add some other features to the network to see if the accuracy of the models increases or not. And the second part of the Homework we learned about the applications of the auto encoders that how they can help to predict the missing or fault parts of an image.

Reference

DeepPose: Human Pose Estimation via Deep Neural Networks, Alexander Toshev, Christian Szegedy

Articulated Pose Estimation by a Graphical Model with Image Dependent Pairwise Relations, Xianjie Chen, Alan Yuille

ImageNet Classification with Deep Convolutional Neural Networks, Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton

2D Articulated Human Pose Estimation and Retrieval in (Almost) Unconstrained Still Images, M. Eichner, M. Marin-Jimenez, A. Zisserman, V. Ferrari

Learning Effective Human Pose Estimation from Inaccurate Annotation, Sam Johnson and Mark Everingham

MVTec AD —A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection, Paul Bergmann, Michael Fauser, David Sattlegger, Carsten Steger

https://medium.com/@quanhua92/alexnet-review-and-implementation-e37a8e4dab54

https://github.com/dansuh17/alexnet-pytorch/blob/master/model.py