KU Leuven

Computer Vision

Final Project - in the Name of Deep Learning

Swarnalata Patra (r0729319)
Master of Artificial Intelligence (2018-19)

Prof. Dr. Dirk Vandermeulen

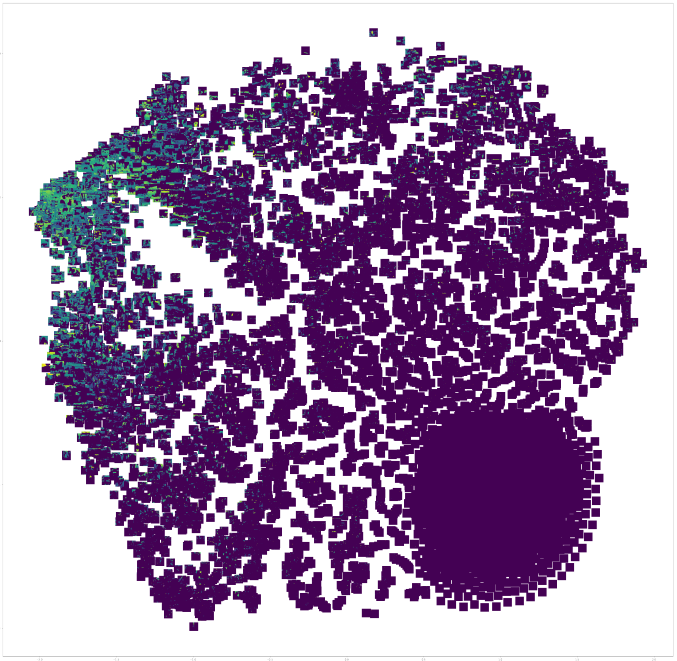
September 2, 2019

1 Introduction

1.1 Data

For this project we have used PASCAL VOC-2009 dataset[1], which consists of color images of various scenes with different object classes. For lower complexity and faster computation on the laptop, I have used 2-3 classes (dog, bird and boat), out of 20 classes available in total. So, in total I have 748 training and 731 validation images to build the model. Further, the validation images were split into 431 validation and 300 test images, so that we can test the performance of the model on unseen(test) data. The images have been resized to 192 by 192 pixels so that we have the uniform size for all the images to build the model.

t-SNE Visualization



For Autoencoders vs PCA, we have used the image size of 128 by 128 pixels and only 2 classes (dog and bird). However for convolution NN, we have built 2 separate models with the image size of 192 by 192 pixels. The major difference between the 2 models of CNN are the number of convolution and max pooling layers. The datasets defined above has been used for all the tasks performed in the project.

1.2 Deep Learning

I have used the keras library for Deep learning techniques. I first used the ADAM optimizer, with different regularization mechanisms and non linear activation functions. However, the accuracy was more with the use of 'rmsprop' optimizer and hence I chose the later one to continue with all my models. The architecture used in the project is as per the requirements like CNN and auto-encoders.

2 Auto-encoders

2.1 PCA vs auto-encoder

In this section, an auto-encoder was constructed to mimic linear PCA.

PCA is basically used for dimensionality reduction. It uses the direction of maximum variance in the data. While, Auto-encoders(AE) can be used for linear as well as non linear transformation depending on the activation functions used in it's encoder-decoder architecture. Auto-encoders can also be stacked with multiple linear layers in order to obtain more detailed local features. However, the training time of an auto-encoder increases drastically as more dense layers are added to the network, whereas PCA is much simpler and more efficient to train in comparison. One can also use pre-trained layers from other models for training an AE in order to speed up learning process and to capture a more efficient representation of the input data. Additionally, with

an increasing amount of features, PCA will result in slower processing compared with AE. To prove the effects experimentally, we used 2 sets of data.

In the first case, we used MNIST data having digit images of size 28 by 28 pixels each, with 60k training and 10k test data. For simplicity we reduced the 784(28x28) dim data to 32 dimensions and tried reconstruction of images from there. We obtained a loss of 9.15%, accuracy of 80.33% and MSE equals 0.96%. Then we tried to reduced the dimension to 32 using stacked AE (784 to 128 to 64 to 32). The new results are as follows: Loss = 8.4%, Accuracy = 81.35%, MSE = 0.76%. So we can see that the performance improved by the use of stacked AE. For the stacked AE, that is meant to mimic a linear PCA, a five layer architecture was used comprising of input layer, output layer, code layer in the middle and a layer on each side of the code layer. Also we could see from figure that the reconstructed images of stacked AE have more clarity as compared to those of single layer AE.

In the second case, we used the given PASCAL VOC-2009 dataset[1]. As mentioned earlier in the Introduction part, we used the 2 classes data with image size 128x128x3 (3 color channels RGB). Then we reduced the image size to 20x20x3 (by repetitive trial) and we could observe that the loss obtained was around 4%.

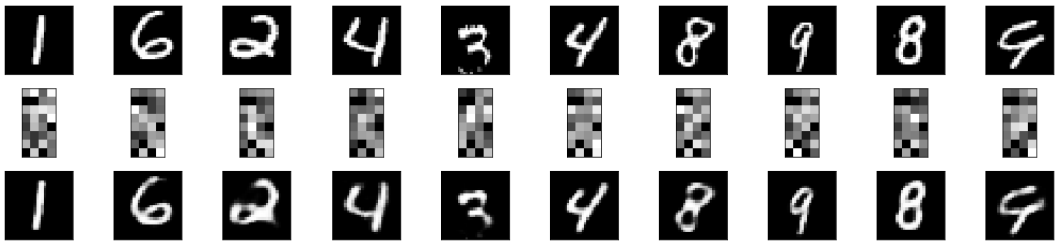
The reconstructed images as well as the summary of the auto-encoder networks for both the dataset is shown in the figure below.

MNIST dataset Summary

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 784)	0
dense_3 (Dense)	(None, 128)	100480
dense_4 (Dense)	(None, 64)	8256
dense_5 (Dense)	(None, 32)	2080

Total params: 110,816
Trainable params: 110,816
Non-trainable params: 0

MNIST dataset Reconstructed Images

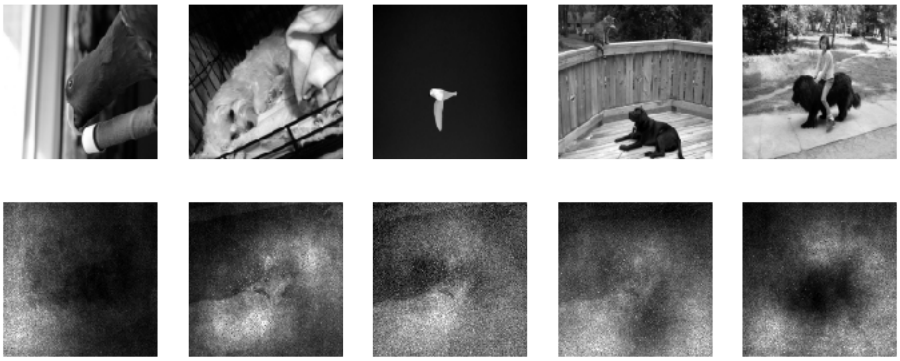


PASCAL dataset Summary

Layer (type)	Output Shape	Param #
input (InputLayer)	(None, 16384)	0
dense_30 (Dense)	(None, 512)	8389120
dense_31 (Dense)	(None, 256)	131328
dense_32 (Dense)	(None, 512)	131584
output (Dense)	(None, 16384)	8404992

Total params: 17,057,024
Trainable params: 17,057,024
Non-trainable params: 0

PASCAL dataset Reconstructed Images

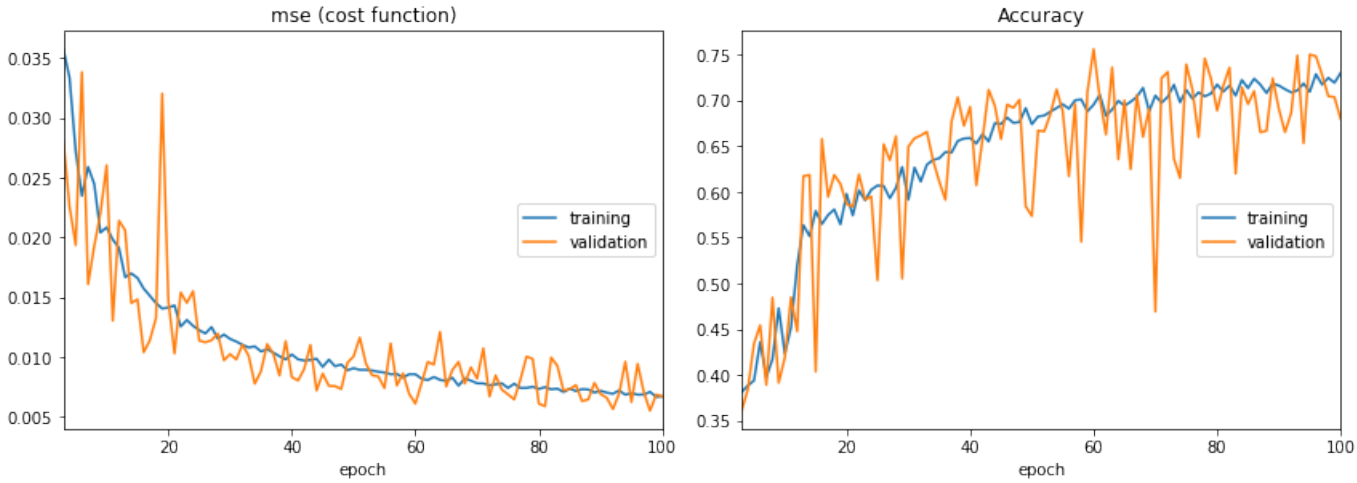


We conclude from the above that using linear PCA (AE mimicking linear PCA) for high resolution images works very poorly. However, further experimentation with the linear AE might improve the results to certain extent, but will be low quality compared to the output obtained by MNIST dataset.

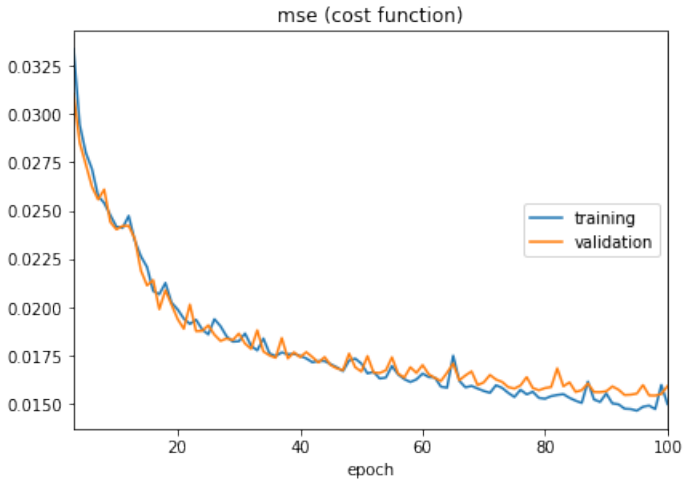
2.2 Non-linear and convolutional

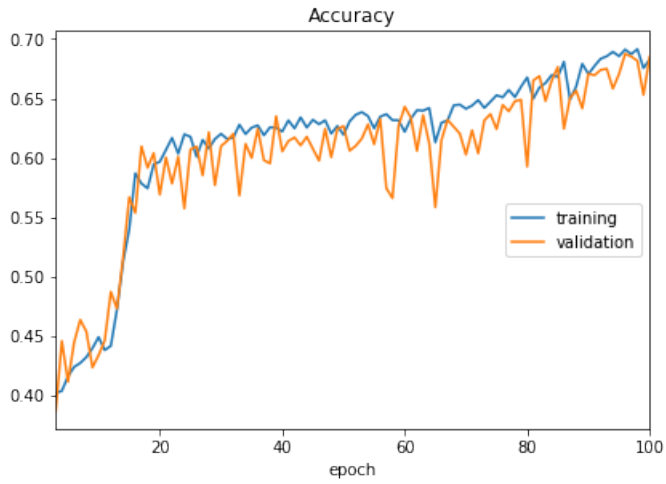
As we know, convolutional neural networks(CNN) have superior performance compared to fully-connected variants as used in the previous section. Hence, in this section, we have implemented 2 different CNN models with varying architecture. In this case, we have considered the size of images to be 192x192 pixels and 3 classes (dog, bird and boat). The activation function used in all the layers except the last softmax layer is ReLU and we used sigmoid activation function in the last decoder layer. For training the CNN models, we experimented with various loss functions: binary_crossentropy for 2 classes and categorical_crossentropy for multi-class cases. However, the accuracy was not as expected. Hence, we decided to go with the MSE (mean square error) loss function as it is easier to interprete. Also for the optimizer, we experimented with 'adam' optimizer first, and then switched to 'rmsprop', where the latter gave better performance overally. We obtained an accuracy of around 70% with the 1st model and around 68% with the second CNN model, as shown in the figures.

CNN Model1 (MSE and Accuracy vs #Epochs)

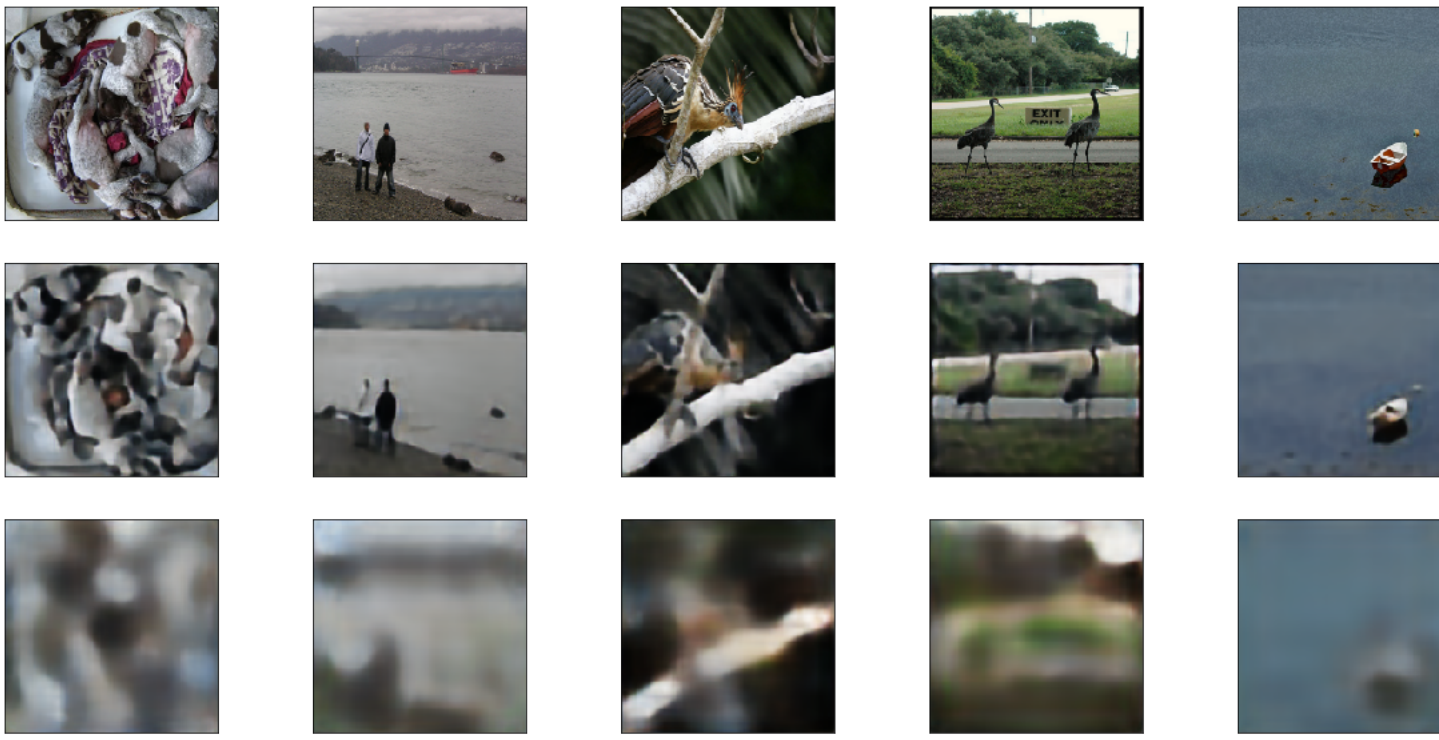


CNN Model2 (MSE and Accuracy vs #Epochs)





Reconstructed Images by CNN Model1 and Model2:



From the above images, we see that the 1st model gives better reconstructed images, also has better performance w.r.t MSE cost function as well as Accuracy. The curves also indicate that the error and accuracy graphs are more stable for training data, however more fluctuating for validation data. And finally we also used the test data for reconstruction of images. Apart from the obtained results, we also experimented with other models by adding and/or removing layers in it, which sometimes reduced the performance as well. Additional layers resulted in huge time complexity. Hence, we mostly used Google colab to run the code. One interesting find was that instead of using the max pooling layer, we could also use convolution layers with varying filter size which could help reduce the dimension as well and performed equally well. However, in the 2 models shown in the figures, we decided to go with the standard form of using the max pooling layers for dimensionality reduction and convolution layer alternatively for extracting local features.

3 Classification

3.1 Object Classification

In this section, we will use the CNN models obtained from earlier to build a classification network. We will build 2 classifier models with and without fine tuning as mentioned. For both the classifier models, all 1479 images(belonging to 3 classes - dog, bird and boat) were used, out of which 748 images were used to train the network. The validation set containing rest images are shuffled and split into 300 test images to test the classifications done by the model and the rest as validation images, used as validation data while training the network.

Classifier1

Setting trainable to FALSE

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 192, 192, 3)	0
conv2d_1 (Conv2D)	(None, 192, 192, 256)	7168
max_pooling2d_1 (MaxPooling2	(None, 96, 96, 256)	0
conv2d_2 (Conv2D)	(None, 96, 96, 128)	295040
max_pooling2d_2 (MaxPooling2	(None, 48, 48, 128)	0
conv2d_3 (Conv2D)	(None, 48, 48, 64)	73792
max_pooling2d_3 (MaxPooling2	(None, 24, 24, 64)	0
conv2d_4 (Conv2D)	(None, 24, 24, 32)	18464
flatten_1 (Flatten)	(None, 18432)	0
dense_1 (Dense)	(None, 3)	55299

Total params: 449,763
Trainable params: 55,299
Non-trainable params: 394,464

Classifier2

Setting trainable to FALSE.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 192, 192, 3)	0
conv2d_1 (Conv2D)	(None, 192, 192, 256)	7168
max_pooling2d_1 (MaxPooling2	(None, 96, 96, 256)	0
conv2d_2 (Conv2D)	(None, 96, 96, 128)	295040
max_pooling2d_2 (MaxPooling2	(None, 48, 48, 128)	0
conv2d_3 (Conv2D)	(None, 48, 48, 64)	73792
max_pooling2d_3 (MaxPooling2	(None, 24, 24, 64)	0
conv2d_4 (Conv2D)	(None, 24, 24, 32)	18464
max_pooling2d_4 (MaxPooling2	(None, 12, 12, 32)	0
conv2d_5 (Conv2D)	(None, 12, 12, 16)	4624
max_pooling2d_5 (MaxPooling2	(None, 6, 6, 16)	0
conv2d_6 (Conv2D)	(None, 6, 6, 8)	1160
flatten_2 (Flatten)	(None, 288)	0
dense_2 (Dense)	(None, 3)	867

Total params: 401,115
Trainable params: 867
Non-trainable params: 400,248

For Classifier 1, we used the pre-trained model and freezed the layers with an additional fully connected layer at the end to build classifier. This is also called Fine Tuning. In this method, it took around 145sec on an average for each epoch and we obtained an accuracy of around 42% with 10 epochs.

For the 2n̂d classifier, we trained all parameters from the scratch and hence each epoch took around 391s (more than double the time taken by Classifier1). However it gave us better performance with binary accuracy as high as 80% with only 10 epochs. We could conclude from here that training the parameters from scratch might give us better performance comparatively, however at the expense of more than double the time. Hence, it is preferable to use the pre-trained models by compromising on the performance to a certain extent. Although, binary accuracy might not be the correct metric of measurement for a multi-class problem. The actual accuracy of the model will be much lower in comparison.

We have computed the accuracy, Precision and recall and F1 score of the predictions by the classifier as well.

We conclude that With fine tuning, the model does not really learned to distinguish between the classes and that's why it classifies all images as one class. When fine tuning was not applied, the model was able to better distinguish between the classes, however the results were still not very reliable to be used as a good classifier.

4 Segmentation

4.1 Data

In this section, the entire Segmentation Class data of the PASCAL VOC-2009 dataset was used. The number of images in the training set is 749 and number of images in the validation set is 750. I am using the image size of 192 x 192 by cropping this specific size of image from the original images. This is done so that the image size

is reduced. Cropping was done instead of resizing as there were issues encountered while reshaping the original images and comparing with their respective segmentation classes.

I decided to perform binary image segmentation, where background is considered as class 0 and foreground is considered as class 1. This is done by grouping all available objects in an image into one single foreground class.

4.2 Model Architecture

For this, I have used the CNN model that was build in the previous section. In the final layer of this network, the resulting features have been classified linearly using logistic regression (sigmoid activation function). The only difference with the current network architecture is the way training is carried out. For the auto-encoder, the target was the training image itself, but for the network used here, the target is the binary segmented image. The sigmoid activation layer returns the probability for each pixels regarding if the pixel belongs to the background or foreground.

Segmentation Model Architecture:

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 192, 192, 3)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 192, 192, 256)     7168
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 96, 96, 256)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 96, 96, 128)       295040
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 48, 48, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 48, 48, 64)        73792
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 32)        18464
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 24, 24, 32)        9248
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 48, 48, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 48, 48, 64)        18496
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 96, 96, 64)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 96, 96, 128)       73856
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 192, 192, 128)     0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 192, 192, 1)       1153
=================================================================
Total params: 497,217
Trainable params: 497,217
Non-trainable params: 0
```

4.3 Parameter Selection and Loss Functions

While training the network, the number of epochs was set to 30 and the learning rate(0.06) was kept low as with increase in the learning rate, it led to over fitting. The Adam optimizer was used while training, and the loss function used was Dice loss. The loss was selected in order to optimize Dice score. Other loss functions used were Jaccard loss, binary, weighted as well as balanced cross entropy.

In binary cross entropy(Bin CE) loss, the loss is applied to each pixel individually. The final loss of a batch is the sum of all losses of each image in the batch (where loss of each image is sum of all loss of each pixel in that image). Weighted cross entropy (WCE) is a variant of Bin CE where all positive examples get weighted by some coefficient. It is used in the case of class imbalance. Balanced cross entropy (BCE) is similar to WCE. The only difference is that we weight also the negative examples in this case.

Below were the loss, accuracy and Dice scores obtained from using different loss functions.

- binary_crossentropy - loss: 5.9750, dice_score: 0.8146, accuracy: 0.6293.
- weighted_cross_entropy - loss: 11.9501, dice_score: 0.8146 - accuracy: 0.6293.
- combined loss - loss: 12.9006, dice_score: 0.8146, accuracy: 0.6293.

The dice coefficient can also be defined as a loss function defined by below equation:

$$\mathrm{DL}(p, \hat{p}) = 1 - \frac{2p\hat{p} + 1}{p + \hat{p} + 1} \quad \text{where } p \in \{0, 1\} \text{ and } 0 \leq \hat{p} \leq 1 \tag{1}$$

It is also possible to combine multiple loss functions, as implemented in the combined loss function.

4.4 Performance Evaluation

The binary image segmentation will be considered to have a multi-label setting. Hence, accuracy is not an effective measure to analyse the performance of a model. As accuracy does not consider partially correct results. Here, the problem is considered to be a multiple binary classification problems rather than a single categorical classification task. Hence, I decided to go with overlapping metrics such as intersection over union score (IoU) and Dice score (DC).[2]

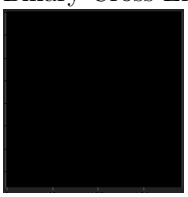
$$\begin{aligned} \text{DC} &= \frac{2TP}{2TP+FP+FN} = \frac{2|X \cap Y|}{|X|+|Y|} \\ \text{IoU} &= \frac{TP}{TP+FP+FN} = \frac{|X \cap Y|}{|X|+|Y|-|X \cap Y|} \end{aligned} \tag{2}$$

where TP are the true positives, FP false positives and FN false negatives.

4.5 Results

True Segmentation Image:

Binary Cross Entropy Resulting Images:



Weighted Cross Entropy Resulting Images:

Combined Loss Resulting Images:

We conclude from the above results that combined loss results in a better segmentation of images. This is in sync with the theory as well.

References

[1] Mark Everingham et al. "The pascal visual object classes (voc) challenge". In: International journal of computer vision 88.2 (2010), pp. 303-338.

[2] https://lars76.github.io/neural-networks/object-detection/losses-for-segmentation/