Université catholique de Louvain

LELEC2885 - Image processing and computer vision

INTRODUCTION TO IMAGE SEGMENTATION

Students :	Lorenzo Fracchia - 29792300 Marco Filipe Silva Santos - 00202300 Vincent Cammarano - 53912100
Group :	07
Professors :	Laurent Jacques Christophe De Vleeschouwer
TA :	Anne-Sophie Collin Antoine Legrand Edouard Couplet

December 2023

1 Introduction

The project is separated in two parts. The first part is an introduction to binary image segmentation using the 'pytorch' library. This introduction will help us to understand how to practically implement image segmentation, deciding which architecture to use, how to train the network and how to evaluate our implementation. The data set used for the project is the "Oxford-IIIT Pet" Dataset, which consists of images of dogs and cats, and their associated ground truth segmentation masks. The second part of the project is the implementation of "test time data augmentation" (TTA) to estimate the uncertainty of our network on the segmentation task.

2 PART A. A classic segmentation task.

2.1 Network Architecture.

The network we decided to use was based on a specific U-Net architecture available on Git Hub[1]. We kept the main structure of the network as it is, but we adapted the parameters to our specific task: we set the number of input channels to 3, since the images from the Dataset are in RGB format, and the number of output channels to 1, since we needed an output prediction mask of just one channel. We also applied a Leaky ReLu function on the output layer as last step.

For all up-sampling layers, we set the 'bilinear' parameter to False, so that no bilinear interpolation during up-sampling was used.

The resulting network begins with a initial layer of double convolution, followed by layers of normalization and ReLU activation. Subsequently, there's four layers of max pooling and down-convolution. Following this, there are four layers of up-convolution (up-sampling), succeeded by a 2D convolution and a Leaky ReLu function layer, generating our output mask.

The output mask, since its values range from 0 to 1, is then passed to a 'thresholding' function, that applies binarization of the mask, setting all pixels that have higher values than 0.5 to 1 and all other pixels to zero. Below, a diagram of our adapted U-net is provided to highlight the layers:

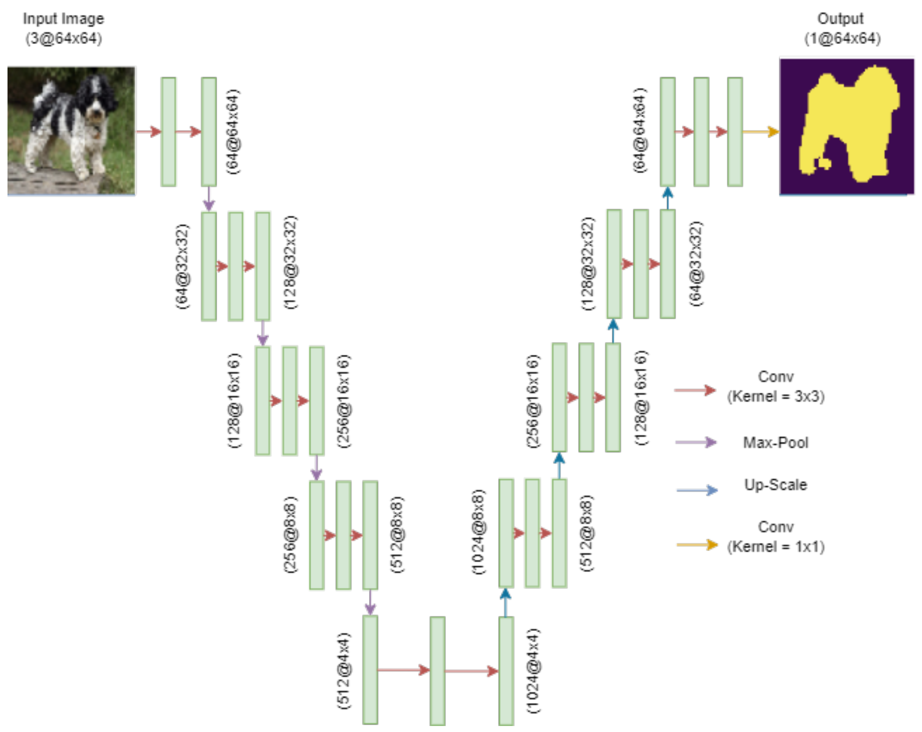

Figure 1: Structure of our adapted network

[1]https://github.com/milesial/Pytorch-UNet/tree/master/unet

2.2 Network Training.

2.2.1 Loss and optimizer

To compute the loss function over the net, we decided to use the BCEWithLogitsLoss implemented in the pytorch library. We chose this function because it is very numerically stable since it combines the BCE with a sigmoid, like described on the documentation[2]:

$$\ell(x, y) = L = \begin{bmatrix} l_1 \\ \vdots \\ l_N \end{bmatrix}^\top, \quad l_n = -w_n \left[y_n \cdot \log(\sigma(x_n)) + (1 - y_n) \cdot \log(1 - \sigma(x_n)) \right]$$

For the optimizer of the network, we chose the Adam[3] one. It is a stochastic gradient descent algorithm on first and second order derivation. It is widely used in the literature and provides good results for a lot of deep learning problems.

2.2.2 Training Parameters

- LEARNING_RATE = 0.001: We have keep the original learning rate since the losses is decreasing smoothly with this one
- EPOCH = 100 : We have used a number of 100, epochs which trains the network enough for our binary segmentation task
- BATCH_SIZE = 16 : A batch size of 16 is a good choice for our network since we have 260 training images and only 65 images for the validation batch.

Here is the learning curve after the first training of the network:

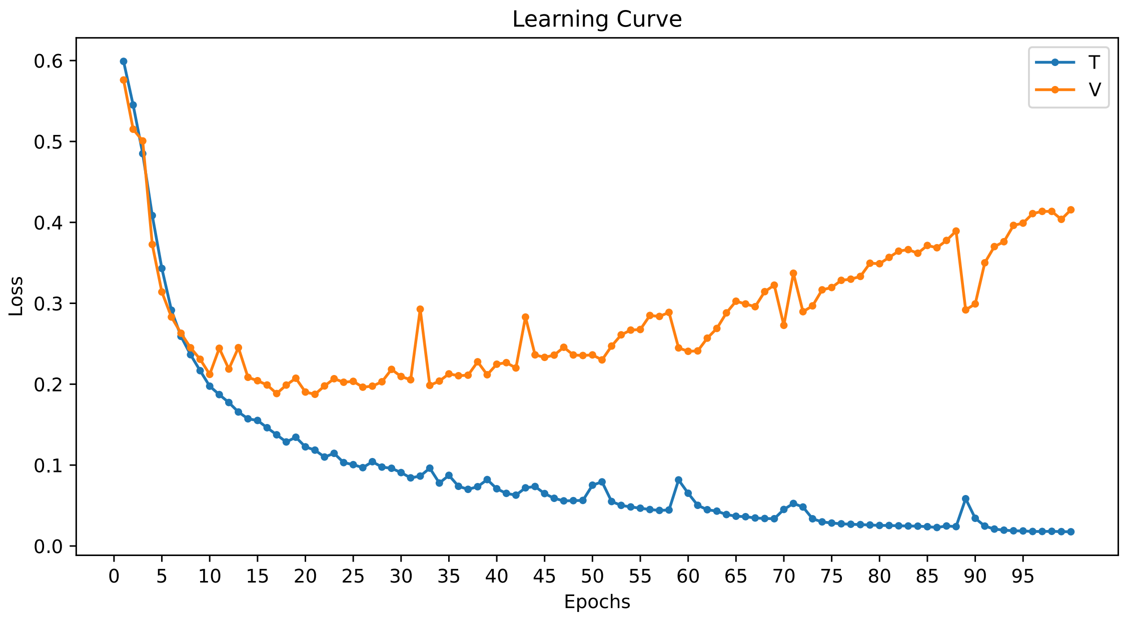

Figure 2: V is for validation curve / T for training curve

[2]https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html
[3]https://pytorch.org/docs/stable/generated/torch.optim.Adam.html#adam

As we can see in the graph above, our first training of the network led to clear over-fitting. After roughly the 10^{th} epoch, the more the net trains, the better it gets at segmenting the training set images, but gets worse results on the ones from the validation set.

To mitigate this problem, we used the albumentations library to apply transformation on the images of the training batch. Here is how it's done practically:

```
alb.OneOf([
    alb.HorizontalFlip(p=1),
    alb.VerticalFlip(p=1),
    alb.GaussNoise(p=1),
    alb.Rotate(limit=45, p=1),
    alb.Rotate(limit=(-45, 0), p=1),
    alb.ToGray(p=1)
])
```

This portion of code is between the Normalize and pytorch.transforms.ToTensorV2() transformations in the imgTransformsList. So on every images of the training set, it will apply with a probability of 0.5 by default one of these transformations. We decided to apply the "data augmentation techniques" directly on the images without adding new ones, for computational purposes. The loss curves on training and validation set after implementing the transformations are shown in the graph down below:

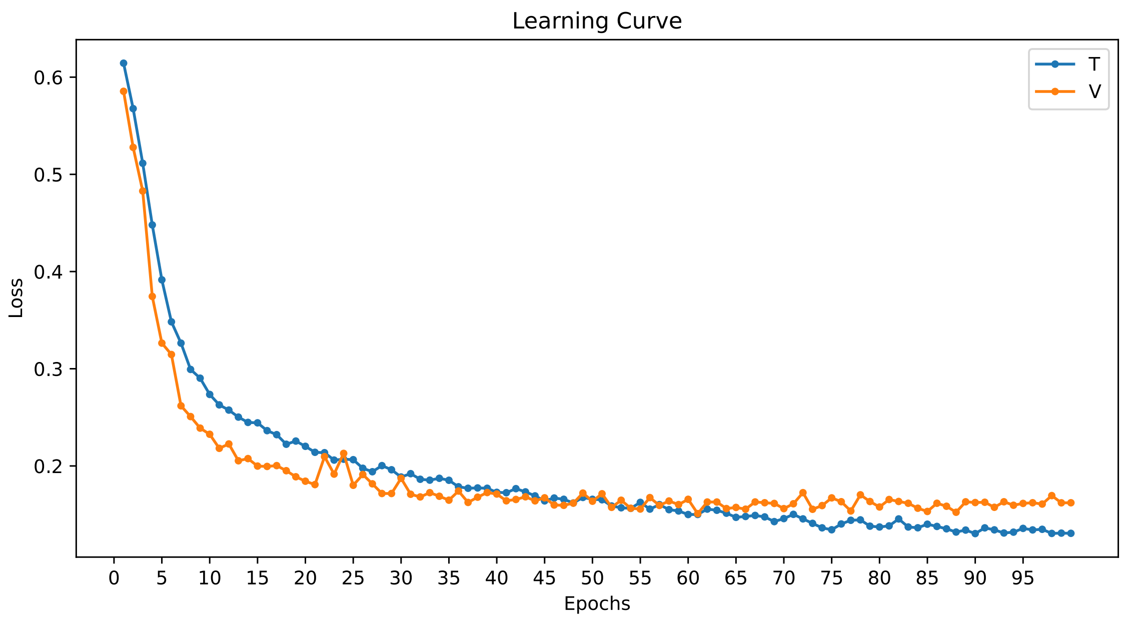

Figure 3: V is for validation curve / T for training curve

By examining the graph, we can observe that there is no more over-fitting! After 70 epochs the network becomes stable and gives nice results.

2.3 Qualitative Evaluation.

2.3.1 Success

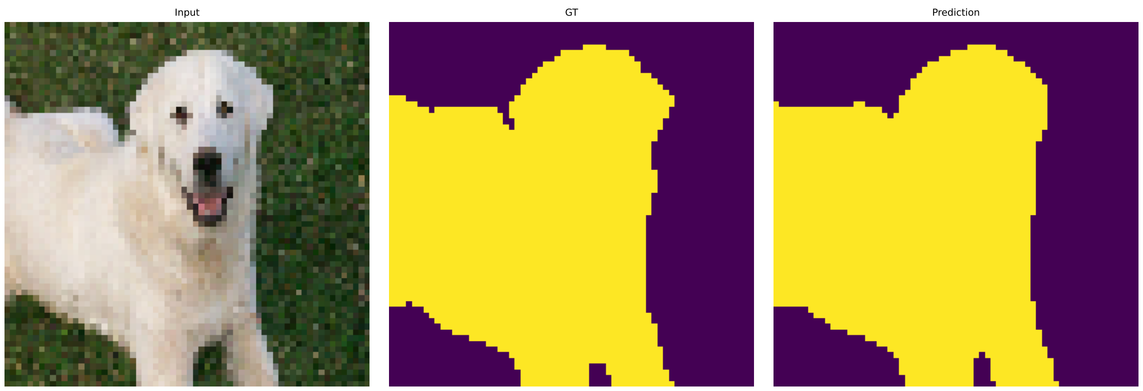

Figure 4: nice example

In general the network provides some good results like the one above. We can see that the predicted mask is really close to the original one, except for few pixels.

2.3.2 Failures

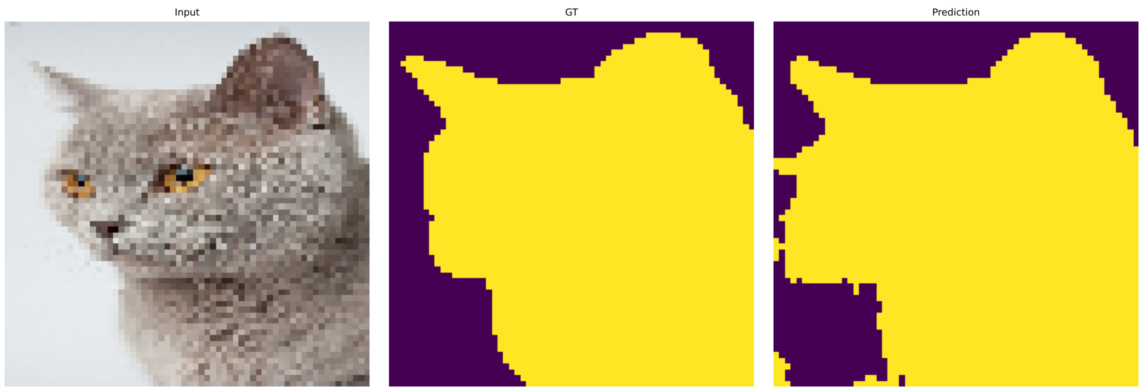

Figure 5: bad example

On the bad result presented above, we can see that at the level of the whiskers of the cat there are a lot of noisy pixels. The same thing happens on the low left corner of the image; these are some bad predictions of pixels by our network.

2.4 Quantitative Evaluation.

As a measure to evaluate our network, we use the following metrics:

1. Intersection over Union (IOU): It's a metric used to evaluate the accuracy of an object detection algorithm. It is calculated by taking the ratio of the area of overlap between the predicted and ground truth regions to the area of their union.

$$IOU = \frac{Intersection\ Area}{Union\ Area}$$

   It provides a measure of how well the predicted segmentation aligns with the actual segmentation, with values ranging from 0 (no overlap) to 1 (perfect overlap).

2. Pixel Accuracy: It's another metric used to measure the overall accuracy of pixel-wise predictions. It is calculated by the ratio of the number of pixels correctly classified to the total number of pixels in the image.

$$PixelAccuracy = \frac{Number\ of\ Correctly\ Classified\ Pixels}{Total\ Number\ of\ Pixels}$$

This metric provides a general indication of how well the model performs at the pixel level, without considering the specific boundaries of the object. The values range from 0 to 1 (perfect classification accuracy).

In the graphs down below we display two bar plots, one for each metric, in which the bars are proportional to the amount of images from the test set that, after feeding them to the model, returned a metric value within a certain range. As we can see, we obtained very high results for both metrics: most of the images led to IoU values falling in the range of 0.8-0.9 (mean of 0.86) and to pixel Accuracy values falling in the range of 0.9-1 (mean of 0.94).

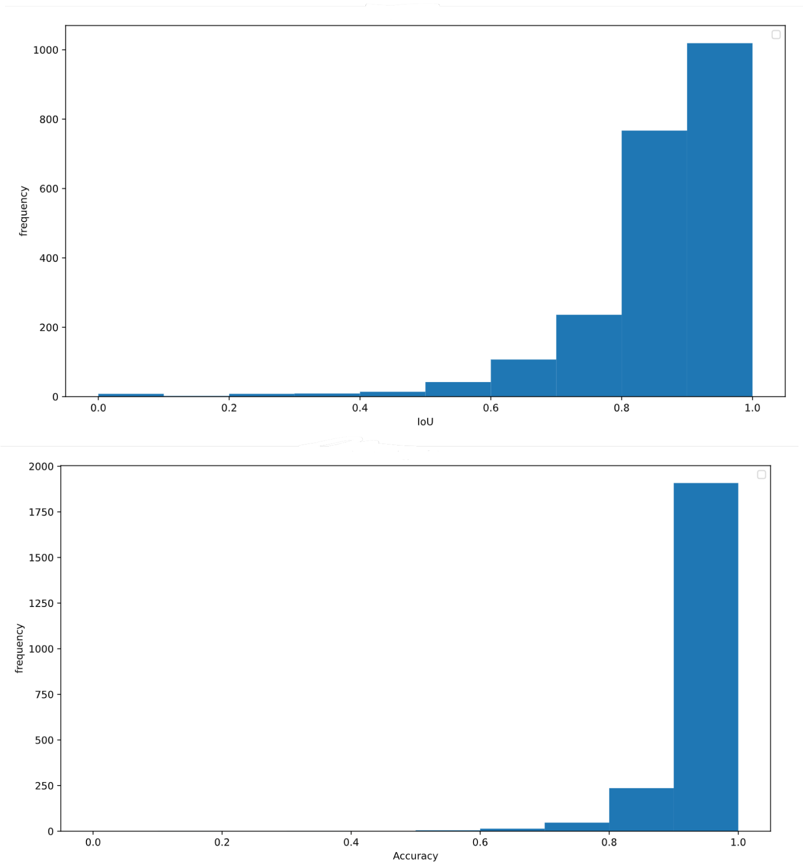

Figure 6: Histogram of IoU/Accuracy metrics on the test set

3 PART B. Uncertainty Estimation

3.1 Objectives

Like we said in the Introduction, this part of the project consists in implementing a method called "test time data augmentation" (TTA). We computed the entropy on multiple predictions of the network at testing time, where these predictions are based on transformations of the original image. This helped us to understand the uncertainty of our trained networks on test predictions and interpret the results.

3.2 Algorithm

For the implementation part of the uncertainty estimation, we followed the algorithm describe in the guidelines.

- 1: For each test image, we generated 5 augmented versions.

  The list of augmentation used is [VerticalFlip,HorizontalFlip,45°Rotation,Gray,Brightened]
- 2: We inferred each augmented image through the network and got 5 predicted segmentation masks.
- 3: We transformed the Vertical, Horizontal and 45°Rotation predictions so that all the predicted masks were aligned
- 4: Computation of the entropy on the 5 masks

3.2.1 Technical Implementation of the TTA

We computed the pixel-wise entropy over the 5 masks following the implementation proposed in the paper[4] suggested in the guidelines:

$$H(Y_i|X) \sim -\sum_{m=1}^{M} \hat{p}_m^i \ln(\hat{p}_m^i)$$

This is the last derivation on the paper where they show that we can compute the entropy for a specific pixel (Y_i) based in our case on the 5 predicted mask [Y_i1,Y_i2,Y_i3,Y_i4,Y_i5]. You can find the python implementation under the name 'entropy' in our code (model_augmentation.py).

[4]Guotai Wang, Wenqi Li, Michael Aertsen, Jan Deprest, Sébastien Ourselin, and Tom Vercauteren. Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks. Neurocomputing, 338:34–45, 2019.

3.3 Evaluation

Thanks to the pixel-wise entropy we were able to compute the uncertainty masks for each image of the test set. The uncertainty mask is zero for all the pixels that have the same value (either 0 or 1) across all the aligned prediction masks, while the more the prediction on a given pixel differs across the aligned masks, the higher the entropy on that pixel is, addressing a bigger uncertainty. As shown in the figure below, where we represented some images and their associated uncertainty masks, the contour of the cats and dogs have the highest values (in green or yellow), because the border between the animal and the background is where the 5 prediction masks very the most. On the contrary, inside and outside the contour the uncertainty is zero (in purple) or close to zero, where all the masks tell the animal and the background apart in the same way. We still can observe some areas outside and

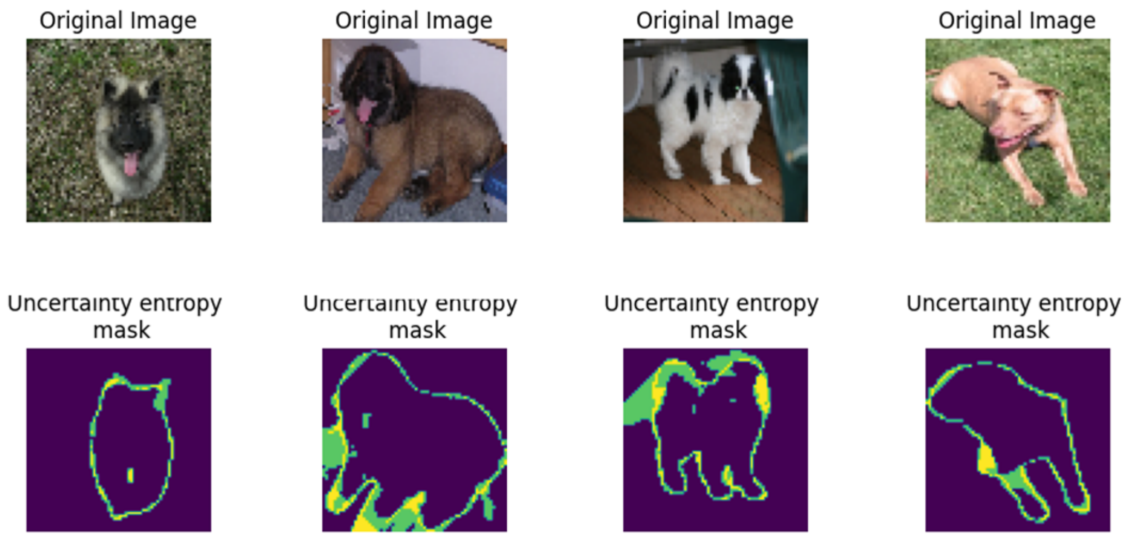

Figure 7: Test set images with corresponding uncertainty mask

inside the contour that look noisy, but those are mostly due to the 45° rotation prediction masks, which have lower performances in predicting the ground truth in respect to the other masks, which show comparable predictions. This is clearly visible in the figure below, showing all the transformations, their prediction masks before and after alignment and the entropy mask. Moreover, during the rotation, the original image is partially cropped, and if in the cropped areas there were parts of the animal, those get lost and are likely labeled as background, leading to higher values of uncertainty.

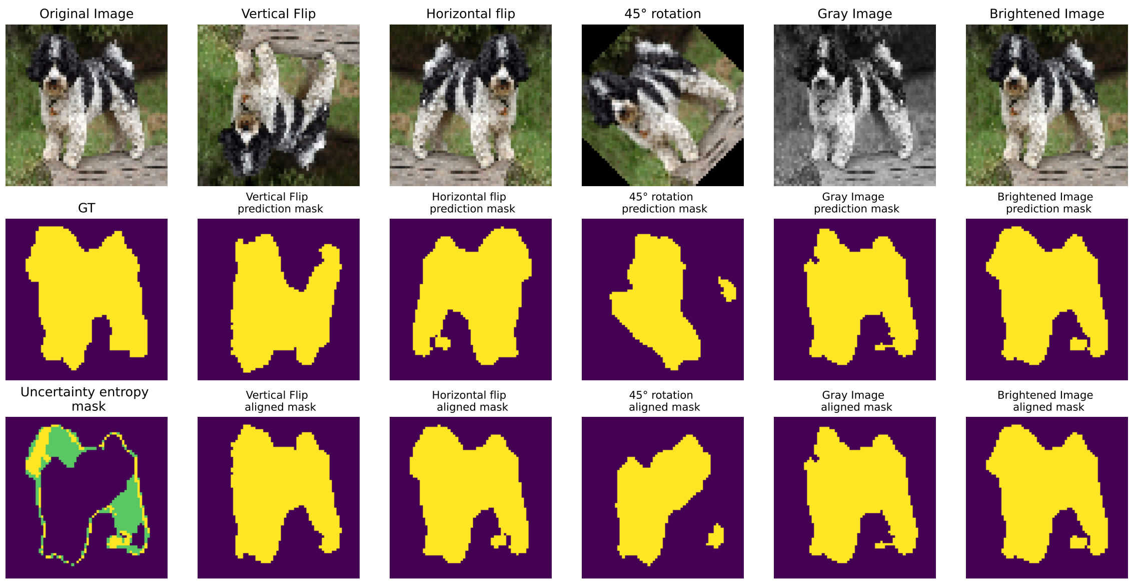
Original Image
Vertical Flip
Horizontal flip
45° rotation
Gray Image
Brightened Image
GT
Vertical Flip prediction mask
Horizontal flip prediction mask
45° rotation prediction mask
Gray Image prediction mask
Brightened Image prediction mask
Uncertainty entropy mask
Vertical Flip aligned mask
Horizontal flip aligned mask
45° rotation aligned mask
Gray Image aligned mask
Brightened Image aligned mask

Figure 8: Example of all the transforms and their predictions. With the entropy of the aligned masks

4 Conclusion

The report outlines a two-part Image Processing and Computer Vision project for the 2023-2024 academic year. In Part A, a U-Net architecture is employed for binary image segmentation using the Oxford-IIIT Pet Dataset.Training involved addressing overfitting through data augmentation. Qualitative and quantitative evaluations showcase successful segmentation with high IoU and Pixel Accuracy.Part B focuses on uncertainty estimation using "test time data augmentation" (TTA) and entropy computation.Evaluation reveals higher uncertainty along object contours, providing insights into the model's confidence.Overall, the project demonstrates effective segmentation and uncertainty analysis.