Image Segmentation using Gaussian Mixture Models and U-Net

Yaseen Haffejee
Computer Science
University of the Witwatersrand
Johannesburg, South Africa
1827555@students.wits.ac.za

Ziyaad Ballim
Computer Science
University of the Witwatersrand
Johannesburg, South Africa
1828251@students.wits.ac.za

Jeremy Crouch
Computer Science
University of the Witwatersrand
Johannesburg, South Africa
1598024@students.wits.ac.za

Fatima Daya
Computer Science
University of the Witwatersrand
Johannesburg, South Africa
1620146@students.wits.ac.za

***Index Terms*—Computer Vision, Image Segmentation, Image Processing**

I. Introduction

The fundamental aim of the project was to build a statistical based model(GMM), and a deep learning model(UNET) in order to perform binary image segmentation. In the process of building these models, consideration had to be made for the data constraints present, as well as the utilisation of the generated masks in the process of building a puzzle solving algorithm. The general dataset is discussed in II, the GMM and U-NET are discussed in Sections III and IV respectively. Finally, a comparative analysis is of the two models is performed in Section V.

II. Dataset

The dataset utilised consists of 48 puzzle pieces. Every puzzle piece has an rgb image with the dimensions 768x1024x, as well as a binary segmented mask which has the dimensions 768x1024. The relatively small size of the dataset may have implications for both models, however these implications may be greater for the Deep learning model since the necessary features to perform segmentation are learnt from the data. However, for the GMM, since we utilise handcrafted features, the sparsity of the dataset will not act as a major challenge to the model since we intend to learn the general statistical properties of the images which are consistent in all the puzzle images.

In order to build models which are able to generalise well, the data had to be split into three sets: the training, validation and testing sets. The recommended split for training, validation and testing was 70%, 15% and 15% respectively. This lead to us having 34 images to train with, 7 images to perform cross validation, and another 7 images in order to test our models. In order to achieve this split, we utilised the in-built train_test_split function which is part of the Scikit-learn package. This enabled us to randomly shuffle the images prior to splitting them up. The random shuffle ensured that different images occurred in every set, thus providing a more stable training procedure and a more general model. Even though we performed this split, to further improve the model, we also performed 6-fold cross validation for each model. This process is discussed in the relevant section for each model.

III. Gaussian Mixture Models

A. Implementation

The Gaussian Mixture Model is a model that aims to capture the information of a dataset utilising multiple Normal/Gaussian distributions. This achieved by defining a latent variable h such that we can represent the Mixture of Gaussians as a marginalisation. Formally, this can be denoted as:

$$P(x|h,\theta) = Norm_x[\mu_h, \Sigma_h] \tag{1}$$

$$P(h|\theta) = Cat_h[\lambda] \tag{2}$$

We can then recover the density by marginalising the joint probability $P(x,h)$. Formally this is denoted as follows:

$$\begin{aligned} P(x|\theta) &= \sum_{k=1}^{K} P(x, h=k|\theta) \\ &= \sum_{k=1}^{K} P(x, h=k, \theta) P(h=k|\theta) \\ &= \sum_{k=1}^{K} \lambda_k Norm_x[\mu_k, \Sigma_k] \end{aligned} \tag{3}$$

The afore mentioned formulation enables two things: 1) The utilisation of Expectation Maximisation in order to learn an optimal set of parameters. 2) Obtaining a closed-form solution in order to update all the parameters.

The expectation maximisation algorithm can be further broken down into 2 steps: The expectation step which will be referred

to as the E-Step and the Maximisation step will be referred to as the M-Step. These two steps are discussed below.

1) E-Step: During this step, we keep the parameters of the model constant and we assign each point in the data to a distribution in the mixture. The assignment is achieved by calculating the responsibility of a point, denoted by r_{ik} of belonging to any of the distributions. The point is assigned to the distribution which yields the highest responsibility. The responsibility is formally defined as:

$$r_{ik} = \frac{\lambda_k Norm_x[\mu_k, \Sigma_k]}{\sum_{j=1}^{K} \lambda_j Norm_{x_j[\mu_j, \Sigma_j]}} \tag{4}$$

2) M-Step: During the M-Step, we keep the responsibilities constant and update the parameters of the model. These parameters include the mean of the distribution in each mixture which is denoted by μ, the covariance of each distribution denoted by Σ, and the probability of each distribution in the mixture which is denoted by λ. As mentioned early, due to the closed form solution, we derive the following update rules:

$$\Sigma_k^{t+1} = \frac{\sum_{i=1}^{I} r_{ik}(x_i - \mu_k^{t+1})(x_i - \mu_k^{t+1})^T}{\sum_{i=1}^{I} r_{ik}} \tag{5}$$

$$\lambda_k^{t+1} = \frac{\sum_{i=1}^{I} r_{ik}}{\sum_{j=1}^{K}\sum_{i=1}^{I} r_{ij}} \tag{6}$$

$$\mu_k^{t+1} = \frac{\sum_{i=1}^{I} r_{ik} x_i}{\sum_{i=1}^{I} r_{ik}} \tag{7}$$

These steps are usually applied until a convergence is reached. In our case, we measured convergence by calculating the log-likelihood after every iteration. If the difference between the log-likelihood of the current iteration and that of the previous iteration is not larger than a threshold of $1e-15$, we assume that the parameters have converged. If convergence was not reached, we allowed a default of 20 maximum iterations before terminating the training procedure.

One major consideration made during training was to ensure that we avoided a singular covariance matrix. A singular covariance matrix usually occurs when the model locks onto a single data-point reducing the variance to 0. Consequently, a 0 variance leads to an infinite likelihood. A singular matrix is thus no longer invertible. In order to combat this, we simply added a regularisation term of 1e-6 to the diagonal elements at every update step. This addition ensures that the covariance matrix remains positive definite, which ensures the matrix is invertible and thus allows us to continue with the minimisation of the likelihood.

Prior to training, the features a separated into background and foreground features based on the masks of the image. In other words, where the mask denotes a puzzle piece, the corresponding features at those indices accounted for foreground features and where the mask denoted the background, those indices were utilised to extract the background features. We then created two models, a foreground and background mixture model respectively. In order to perform inference, we simply evaluate each data-point using the learnt parameters, and assign it to either the foreground or background which yields a mask. Given the foreground prior denoted by λ, the formula is as follows:

$$P(l = fg|x) = \frac{\lambda Norm_x[\mu_{fg}, \Sigma_{fg}]}{\lambda Norm_x[\mu_{fg}, \Sigma_{fg}] + (1-\lambda) Norm_x[\mu_{bg}, \Sigma_{bg}]} \tag{8}$$

We can clearly see that the formula utilises two class conditional models, a background and a foreground conditional model in order to perform inference. In Lab 2 however, we only utilised a background conditional model. This lead to poor results on unseen data, since none of the actual information about the puzzle pieces were considered. This also was the reason we had to choose extremely small thresholds in order to ascertain any kind of segmented image. By utilising Equation 8, we are able to capture the foreground and background information, as well as their respective priors which provide weighting to each model during classification. The result of inference using Equation 8 is a mask of shape 768x1024. The final step was threshold the mask such that any value greater than 0.5 was set to 1 , a foreground pixel, and any value less than or equal to 0.5 was set to 0, a background pixel. Thus resulting in a binary segmented mask.

B. Feature sets

Three feature sets were investigated in order to determine which set enabled the optimal performance of the Gaussian Mixture Model. The three sets were as follows:

1) The RGB pixel values
2) The RGB pixel values coupled with the Difference of Gaussian response
3) The RGB pixel values coupled with the Texton values

1) Difference of Gaussians: The Difference of Gaussians is a feature enhancement algorithm which involves blurring an image using two different degrees of blurriness, and subtracting them from each other resulting in an image with enhanced features. In the case of the puzzle pieces, the enhanced features were that of the edges of the puzzle piece. This is shown in Figure 2. This is a useful feature since the edges play a pivotal role in the segmentation of the puzzle piece and the background.

In order to obtain the Difference of Gaussian, we simply utilised the skimage difference_of_gaussians function and applied it to the grayscale version of the RGB image. We then concatenated this as an additional feature to the original RGB pixels. Thus we now had a 34x768x1024x4 tensor which is separated into foreground and background pixels based on the masks and then utilised to train the respective foreground and background mixture models.

2) Textons: Textons refer to the smaller structures in images. In order to group similar structures, we utilised K-Means clustering. The clustering manages to group similar pixels as shown in Figure 3. Since we expect simlar pixels to form part of the same group, i.e background or

foreground, Textons enables us to ascertain these similarity clusterings which can act as a strong discriminator between the foreground and background.
In order to obtain the Texton values, we simply utilised the OpenCV K_Means clustering function and applied it to the RGB image. We reshaped the assigned clusters into 768x1024 images. We then concatenated this as an additional feature to the original RGB pixels. Thus we now had a 34x768x1024x4 tensor which is separated into foreground and background pixels based on the masks and then utilised to train the respective foreground and background mixture models.

In order to determine which feature set provided the best results, we trained 3 Mixture models utilising all these sets whilst keeping the number of components in the mixture fixed at 3, as well as the maximum iterations at 20. All the models performed relatively well on the validation set. The results are captured in Table I.
From the results found that the addition of the D.O.G and the texton features improved the performance of the models. All the feature sets produced similar IOU values, however the addition of the D.O.G features lead to the highest Dice coefficient. Consequently, we chose to utilise the RGB pixel values as well as the D.O.G feature as the optimal feature set for our model. The predicted outputs for the validation set from this model can be seen in Figure 1.

C. 6-fold cross validation

6-fold cross validation is when we split a dataset into a training and testing set. The training set is then split into 6 parts. We fit 6 models, each time holding out a single subset as the validation set, and the remaining 5 are utilised for training. Utilising this method to find models ensures the best model is chosen since we are training on different sets of the data, and in doing so revealing how well the model performs on all subsets of the data. This allows us to build more stable models that are more capable of generalising on unseen data.
We utilised 6-fold cross validation in order find the optimal values for the two hyperparameters of the model: The number of Gaussian components in a mixture, and the maximum iterations allowed before terminating training, provided the parameters have not converged.

1) Number of components: In order to determine the number of components, we kept the number of maximum iterations constant at 20, and trained models with 2,3,4 and 5 components per mixture respectively. The 6-fold results for these models can be found in Tables II, III,IV,V. In order to determine the best performing model, we summarised the results over each of the 6-folds by taking the average value. From the averaged results, we found that utilising more than 3 components provided a negligible increase in the Dice coefficient of the model, a negligible increase in the IOU and a major increase in the training time. Therefore we concluded that utilising 3 components enabled us to decrease the training time by a factor of 1.5, whilst incurring a negligible decrease in accuracy. The averaged results for the number of components can be found in Table VI.

2) Maximum iterations: In order to determine the best number of iterations we set the number of components to 3 and and trained models using 10,20 and 30 maximum iterations. The 6-fold results for these models can be found in Tables VII,VIII and IX respectively. In order to summarise the results into single values, we averaged out the results which can be found in Table X. The summarised results showed that utilising 10 iterations provided a decent base, but the model tends to under-fit the data which lead to a lower IOU and Dice coefficient. If we consider 20 and 30 iterations, we noticed an increase in performance compared to that of 10 iterations, however the difference in performance between 20 and 30 iterations is again negligible. However, the training time increased by a factor of 1.4. Consequently, we concluded that utilising 20 iterations will provide a more stable model with a smaller chance of overfitting the data.
We therefore concluded that using 2 components and a maximum of 20 iterations proved optimal for our dataset.

D. Results

Since we already utilised 6-fold cross validation in order to find the optimal hyperparameters, we utilised the initial training and validation set for training. Therefore we utilised 41 images for training and 7 unseen images for testing. The total training time for the model using 3 components and a maximum of 20 iterations was 4.7 minutes. We then performed inference on the test. The total time taken to perform inference on the 7 test images was 4.6 seconds. The test results can be viewed in Figure 4. The model was able to achieve a testing IOU of 0.953 and a testing Dice score of 0.976. These results are discussed in detail below.

E. Analysis

From Section III-D, we found that the GMM performed remarkably well on the test set. However, if we scrutinise the test set closely, we can see that the puzzle pieces are diverse with a variety of colours and the model is capable of generalising well. If we look at Figure 5, we see that initially the model failed to segment puzzles where the puzzle piece had pixels of the same colour as the background. This is denoted by the first row in the Figure. However, once we trained the model with the optimal hyperparameters and including a similar sample in the training set, we see that the model was capable of differentiating between the puzzle and the background even though the colours were the same, this is denoted by the second row in Figure 5. This speaks to the importance of having a diverse dataset during training which does not contain biases, as this will lead to a more robust model.
If we isolate the metrics, we may see that the average precision from the 6-fold cross-validation and the precision of the test model may differ. The test model achieved a slightly lower precision score, since the in the first mask, some noisy pixels were classified as puzzle pieces. This can be seem in the first

row in Figure 4.
The utilisation of 3 components proved to be appropriate since it managed to capture the multi-modal aspect of the data well. Utilising more than 3 components would have increased computation whilst not providing a major increase in performance. We also notice that allowing a maximum of 20 iterations before terminating training enabled the model to learn the distribution of the data, whilst maintaining the ability to generalise on unseen data. From the results, we feel that the model performed optimally on the given dataset.

F. Conclusion

Given the sparsity of the dataset, the model managed to generalise well on the test dataset. The utilisation of 6-fold cross-validation played a vital role in removing bias from our models by varying the training and validation data. It also enabled us to find a set of optimal hyperparameters which were suitable for the different variations of the data used during the cross-validation. We can therefore conclude that given a sparse dataset, if segmentation is required, Gaussian Mixture Models can be a viable option due to their ability to use minimal data, their fast training and inference speed. The only downfall is that to improve performance, hand-crafted features like the Difference of Gaussian may need to be utilised, and the set of features that will improve the performance is dependent on the dataset. Thus, to unlock the full potential of the GMM's, careful consideration needs to be made about the type of features that will be utilised.

IV. U-NET

A. Implementation

As outlined in the Project brief, we chose our U-Net's backbone to be the VGG16 architecture. This entailed building the encoder side of the U-Net using VGG16's layers and weights which gave us 5 blocks instead of a Vanilla U-Net's 4 blocks. We also opted to use VGG16's pre-trained IMAGENET1K _V1 weights.

1) U-Net architecture: U-Net is a semantic segmentation model which consists of an encoder and a decoder network. The encoder takes in an RGB image and reduces the input image into a latent representation of itself through the process of convolution. The latent representation consists of feature maps which contains the features that the network learnt to be important during the training of the model. The encoder consists of contractional blocks, and within each contractional block there exists two fundamental operations: a 3x3 convolution and 2x2 max-pooling. The convolution enables the network to scan the entire image for the presence of the kernel signals, the response received from kernel is then reduced into a smaller representation utilising max-pooling. Max-pooling summarises the responses into single values, where a higher value is indicative of the presence of a feature and the absence of the feature the low the response is. There are 5 of these contractional blocks, and the further we move down the encoder, the smaller the image becomes but the number of feature maps doubles until we reach the bottleneck.
The bottleneck is the point at which we have the latent representation of the image. The latent representation is then passed through the decoder network. The decoder network consists of 5 expansion blocks. These blocks consist of three operations: concatenating spatial information from the corresponding contractional block, a 3x3 convolution and a 2x2 upsampling. The skip connection essentially transfers the feature maps from the encoder to the decoder. The transfer of these feature maps enables the decoder to gain the spatial information that was lost during the encoding process, and consequently the model is capable of reconstructing the semantic maps with the correct spatial information. The 3x3 convolution ensures the reduction of feature maps as we move up through the network. The 2x2 upsampling ensures we double the size of the image at each block in order to reach the original input size at the end of the decoder. The result is the segmentation mapping. The U-Net's name is naturally derived from the shape of the network. A general architecture of the U-Net model can be seen in Figure **??**.

2) VGG16 Weights:: VGG16 is a specifically designed CNN architecture that has been an award winning vision model for image classification. The model was trained on the imagenet dataset which contains approximately 1.2 million natural images. Since the model achieves a 92.7% accuracy on imagenet, it is fair to say the kernels do a good job at capturing the features within these images. Consequently, we transfer these weights into the kernels of our encoder network. This provides us with reasonable initialisation of the kernel weights and can reduce the amount of time spent fine-tuning the weights during training and improve the overall performance of the model.

3) Training U-Net: Given we have set up the U-Net architecture and initialised the kernel weights using the VGG16 models weights, we trained the model in the following way:

1) The RGB puzzle image is passed into the network.
2) The images passes through the encoder portion in order to get the feature maps.
3) The image then passes through the decoder in order construct the segmentation maps.
4) At the final layer, the ground-truth mask is compared to the models mask.The error is calculated utilising the Binary cross-entropy function.
5) The error is then backpropagated through the network and the weights are adjusted.

This procedure continues until the Binary Cross-Entropy Loss function is minimised. We utilised Binary cross-entropy as the loss function since the output is a pixel-wise binary prediction. In other words, the model needs to predict either 1(foreground) or 0(background) per pixel in the image. The equation for

binary cross-entropy is given below:

$$cross\_entropy = \frac{-1}{N}\sum_{i=1}^{N}(Y_i \cdot log(\hat{Y}_i) + (1 - Y_i) \cdot log(1 - \hat{Y}_i)) \tag{9}$$

The Binary cross-entropy function enables us to calculate the error in the probability of our predictions and make adjustments to the weights accordingly such that the loss function is minimised. In order to optimise the loss function in 9, we utilised the Adam optimiser. Given that the general architecture was decided, we utilised two sets of data to train two models.

The first set of data utilised was the general training set mentioned in II. However, given the fact that deep learning usually requires more data in order to perform well, we also performed data augmentation in order to curate a second dataset with more data. In order to generate these additional images, each images within the original dataset underwent the following transformations:

1) Scaling
2) Rotation
3) Vertical and Horizontal Flipping
4) Random cropping

All of the augmentations listed come from the Torchvision Augmentations library and each transformation was done with a purpose. The scaling of the images ensures we have images on multiple different scales. Consequently this enables us to inject a certain degree of scale invariance within the network. The rotations enable us to have images that are at different orientations. This allows the network to learn features that are at angles since all the original images have the puzzle pieces at the same angle. As a result, this allowed us to ensure the network is also has rotational invariance. Flipping the images vertically and horizontally ensured we had enough data to train a deep learning model effectively, without duplicating any of the images.Finally, the random cropping removes a random portion of the image, which ensures the model can focus on various other portions of the image which may contain other features in order to perform the semantic segmentation. The combination of these techniques ensured the model did not overfit creating a more general model, whilst also providing sufficient data for the network to train on. A comparison of the results between this model trained on the general set of images and the augmented dataset is discussed in Section IV-C.

B. Results

In this section we outline the results that were achieved by the models utilising the original dataset and the dataset containing the data augmentations.

1) Original Dataset: On the original dataset, we had a training set of 34 images, a validation set of 7 images and a testing set of 7 images. Utilising these sets, the model managed to achieve a testing IOU of 0.832 and Dice coefficient of 0.874. Considering the size of the training set, these results are acceptable considering the data hungry nature of deep learning models. A samples result can be seen in Figure 9.

2) Augmented dataset: On the augmented dataset, we performed 2 rotation transformations, 2 scaling transformations and the vertical and horizontal flip per image. This provided us with a training set 238 images which 7 times more samples than the original dataset. Utilising this set for training, the model managed to achieve an IOU of 0.909 and a Dice coefficient of 0.926. Sample test results for this model can be seen in Figures 7 and 8 respectively. We can see that the model segements the image really well, however around the edges there is some noise which reduced the overall performance metrics of the model.

The results are studied comparatively in Section IV-C.

C. Analysis

Considering the results found in Section IV-B, we can see that the addition of the data augmentations improved the performance of the model. This is due to the fact that the augmentations introduce not only more samples to train on, but a more diverse set of images due to transformations applied. Consequently, the model becomes more robust which leads to an improved performance. In the case of the puzzle pieces, the original dataset does not contain much variation in terms of the orientation or sizing of the images. Thus by introducing these through augmentation, we are manually injecting rotational and scale invariance to an extent without encoding it in the architecture directly. Another advantage of having additional data for training is the reduction in the likelihood of overfitting during training. This allows the model to perform better on unseen data producing a more generalised model. Finally, the addition of the random crops in the training set forces the model to look at various other features that can be utilised in order to segment the image. This ensures that the model does not fixate on a single feature, making it a more general model which can perform well on unseen data. From this we decided to utilise the model trained with the augmented dataset as the final model.

D. 6-fold cross validation

Two pivotal hyperparameters that dictate the speed of training as well as the generalisability of the model are the learning rate and epochs used during training. The learning rate dictates the rate at which move in the direction of descent during the minimisation of the loss function. The epochs dictates how long we train for prior to terminating training and is directly related to the model overfitting or underfitting. In order to determine the best values for these parameters, we utilise 6-fold cross-validation.

1) Learning Rate: We utilised 2 learning rates of 0.0001 and 0.00003 respectively. Utilising these learning rates, we kept the epochs constant at 50 epochs. The 6-fold results for these models can be found in Tables XI and XII respectively. From these results we can clearly see that utilising a smaller learning rate gave us a slightly longer training time since the

steps in the direction descent is much smaller which results in slower convergence. For the learning rate of 0.0001 , we achieved an average Dice coefficient of 0.897 and average IOU of 0.883. For a learning rate of 0.00003 we achieved an average Dice coefficient of 0.926 and an average IOU of 0.906. From these results we concluded that utilising a smaller learning rate provided us with more stable gradient and consequently better performance. Therefore we utilised a learning rate of 0.0003.

2) Epochs: The epochs dictate how long training continues for. We kept the learning rate at the earlier optimal rate of 0.0003 and trained two models using 50 and 75 epochs respectively. The results for these models using 6-fold cross validation can be found in tables XIII and XIV respectively. Utilising 50 epochs, we managed an average IOU of 0.901 and an average Dice coefficient of 0.913. Utilising 75 epochs, we managed an average IOU of 0.841 and an average Dice coefficient of 0.867. From these results we can conclude that utilising more epochs did not translate into better performance since the model overfit the training data reducing the ability of the model to generalise on unseen data. Consequently, we opted to utilise 50 epochs as the optimal number of epochs.

E. Conclusion

Given the size of the original dataset, U-Net would not be the best option. However, the power of image augmentations revealed itself through the major improvements in performance when these augmentations were included in the training set. The learning rate and epochs played a vital role in the improvement of the performance of the model by allowing less overfitting and creating a more general model.

V. GMM VS U-NET

Performing a comparative analysis between the GMM and U-Net utilising the optimal models, we can see that the GMM performed slightly better than U-Net. One of the reasons for this could be the fact that in the GMM, we utilised handcrafted features which we were sure would assist the model in segmenting the image. However, in U-Net, the learning of the feature set is left purely up to the network and thus the network may not have learnt an optimal set of features. Another consideration is the fact that U-Net performed better on the augmented dataset since it had more data to train on. However, it should be noted that the transformations applied were not random which could have introduced a bias into a dataset which affected the models ability to generalise. If we were to extend the comparison of performance between these two models and the models we utilised in the labs, we can see that these models have a better ability to generalise on unseen data. This is due to the fact that in the labs, we did not model the foreground in any way and thus any vital information such as the edges of the pixels were negated. Another consequence of negating a foreground model is the fact that the threshold to produce the final mask was extremely small, thus making it difficult to threshold the image effectively whereas for the GMM a threshold of 0.5 and a threshold of 0.3 for U-Net proved to be sufficient. Therefore we can conclude that the GMM and U-Net are better segmentation models than the lab models and finally we can choose whether to use U-Net or a GMM depending on the size of dataset. In general, a GMM will perform better on a smaller dataset and U-Net will perform better on a larger dataset. However, for a smaller dataset, one can opt to use U-Net, however the trade-off would be between the time to perform the correct augmentations versus finding the optimal handcrafted features for the GMM.

APPENDIX

Features	IOU	DICE
RGB	0.891	0.936
RGB + D.O.G	0.897	0.940
RGB + Texton	0.891	0.937

TABLE I
FEATURE SET RESULTS

Fold No.	training_time	inference_time	IOU	DICE
0	3.3391	0.0434	0.896	0.938
1	3.2657	0.0495	0.911	0.952
2	3.2190	0.0501	0.942	0.970
3	3.3303	0.0496	0.825	0.899
4	3.3376	0.0554	0.946	0.972
5	3.3278	0.0530	0.864	0.917

TABLE II
6- FOLD CROSS VALIDATION WITH 2 COMPONENTS

Fold No.	training_time	inference_time	IOU	DICE
0	4.6696	0.0630	0.893	0.936
1	4.5596	0.0719	0.924	0.959
2	4.5649	0.0732	0.946	0.972
3	4.5843	0.0764	0.821	0.896
4	4.5641	0.0738	0.955	0.977
5	4.4840	0.0711	0.870	0.919

TABLE III
6- FOLD CROSS VALIDATION WITH 3 COMPONENTS

Fold No.	training_time	inference_time	IOU	DICE
0	5.9868	0.0813	0.890	0.933
1	5.9018	0.1000	0.923	0.958
2	5.9746	0.1002	0.946	0.972
3	6.0231	0.1000	0.847	0.913
4	6.0737	0.1069	0.957	0.978
5	5.9083	0.1031	0.887	0.931

TABLE IV
6- FOLD CROSS VALIDATION WITH 4 COMPONENTS

Fold No.	training_time	inference_time	IOU	DICE
0	7.5587	0.1087	0.889	0.933
1	7.3736	0.1366	0.927	0.960
2	7.1842	0.1209	0.952	0.975
3	7.0941	0.1214	0.860	0.922
4	7.1959	0.1246	0.963	0.981
5	6.5349	0.1243	0.884	0.931

TABLE V
6- FOLD CROSS VALIDATION WITH 5 COMPONENTS

N_Components.	training_time	inference_time	IOU	DICE
5	7.1569	0.1228	0.9125	0.9503
4	5.9780	0.0986	0.9083	0.9475
3	4.5711	0.0716	0.9015	0.9432
2	3.3032	0.0502	0.8973	0.9413

TABLE VI
6- FOLD NUMBER OF COMPONENTS CROSS VALIDATION SUMMARY

Fold No.	training_time	inference_time	IOU	DICE
0	2.4270	0.0636	0.847	0.908
1	2.3603	0.0762	0.863	0.925
2	2.4109	0.0829	0.905	0.950
3	2.3905	0.0849	0.769	0.861
4	2.4059	0.0756	0.908	0.952
5	2.3611	0.0775	0.815	0.884

TABLE VII
6- FOLD CROSS VALIDATION WITH 10 MAX ITERATIONS

Fold No.	training_time	inference_time	IOU	DICE
0	4.7386	0.0660	0.893	0.936
1	4.6593	0.0756	0.924	0.959
2	4.7347	0.0802	0.946	0.972
3	4.6566	0.0808	0.821	0.896
4	4.7307	0.0812	0.955	0.977
5	4.6886	0.0760	0.870	0.919

TABLE VIII
6- FOLD CROSS VALIDATION WITH 20 MAX ITERATIONS

Fold No.	training_time	inference_time	IOU	DICE
0	7.1858	0.0622	0.898	0.940
1	6.6384	0.0798	0.925	0.960
2	6.9260	0.0773	0.929	0.963
3	6.7689	0.0755	0.861	0.922
4	6.8805	0.0752	0.951	0.974
5	6.6470	0.0752	0.867	0.911

TABLE IX
6- FOLD CROSS VALIDATION WITH 30 MAX ITERATIONS

Max_iterations.	training_time	inference_time	IOU	DICE
30	6.8411	0.0742	0.9052	0.9450
20	4.7014	0.0766	0.9015	0.9432
10	2.3926	0.0768	0.8512	0.9133

TABLE X
6-FOLD MAX ITERATIONS CROSS VALIDATION SUMMARY

Fold No.	training_time	inference_time	IOU	DICE
0	49.37	0.0321	0.887	0.919
1	52.14	0.0541	0.901	0.923
2	53.33	0.0491	0.922	0.894
3	50.18	0.0499	0.833	0.859
4	55.74	0.0554	0.911	0.901
5	48.11	0.0480	0.844	0.897

TABLE XI
6- FOLD CROSS VALIDATION USING 0.0001 LEARNING RATE

Fold No.	training_time	inference_time	IOU	DICE
0	61.89	0.0441	0.921	0.928
1	64.45	0.0358	0.894	0.914
2	68.33	0.0514	0.903	0.969
3	67.12	0.0339	0.873	0.899
4	61.11	0.0291	0.936	0.945
5	69.99	0.0415	0.910	0.901

TABLE XII
6- FOLD CROSS VALIDATION USING 0.00003 LEARNING RATE

Fold No.	training_time	inference_time	IOU	DICE
0	52.25	0.0341	0.914	0.931
1	51.81	0.0498	0.891	0.910
2	50.39	0.0474	0.929	0.916
3	55.45	0.0559	0.889	0.904
4	57.91	0.0289	0.876	0.892
5	53.63	0.0310	0.909	0.924

TABLE XIII
6- FOLD CROSS VALIDATION USING 50 EPOCHS

Fold No.	training_time	inference_time	IOU	DICE
0	65.48	0.0311	0.841	0.878
1	67.78	0.0544	0.866	0.894
2	66.23	0.0495	0.894	0.902
3	61.33	0.0555	0.811	0.841
4	68.31	0.0614	0.822	0.858
5	61.12	0.0335	0.810	0.830

TABLE XIV
6- FOLD CROSS VALIDATION USING 75 EPOCHS

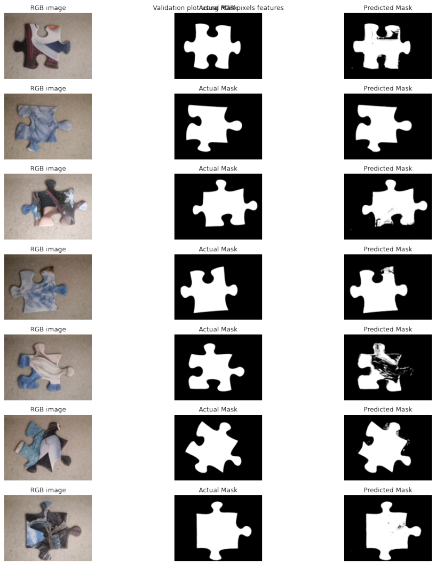

Fig. 1. RGB+D.O.G Validation

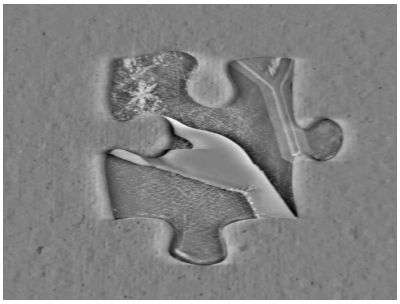

Fig. 2. D.O.G Result

Fig. 3. Texton

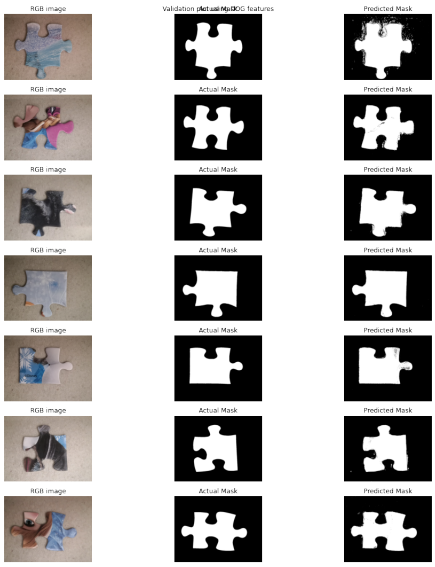

Fig. 4. Test Results

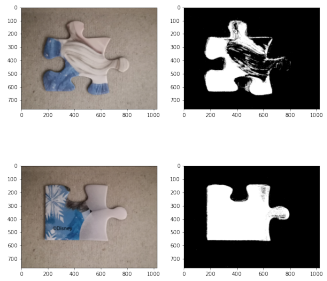

Fig. 5. Struggled

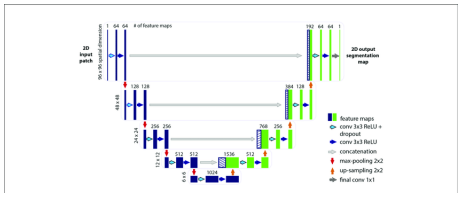

Fig. 6. U-Net Architecture

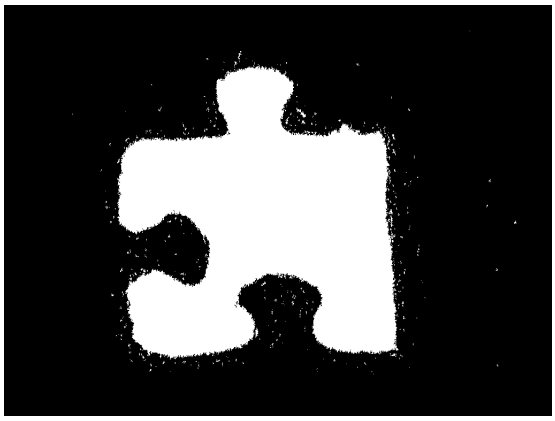

Fig. 7. U-Net result 1

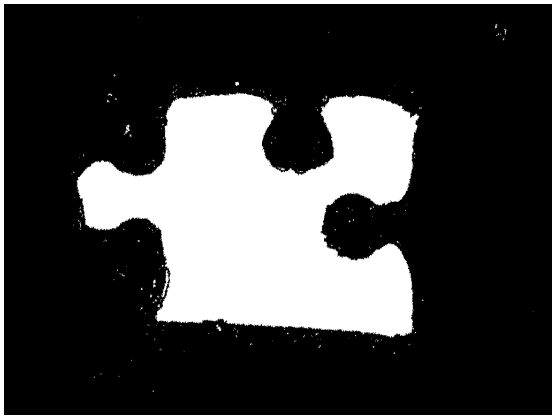

Fig. 8. U-Net result 2

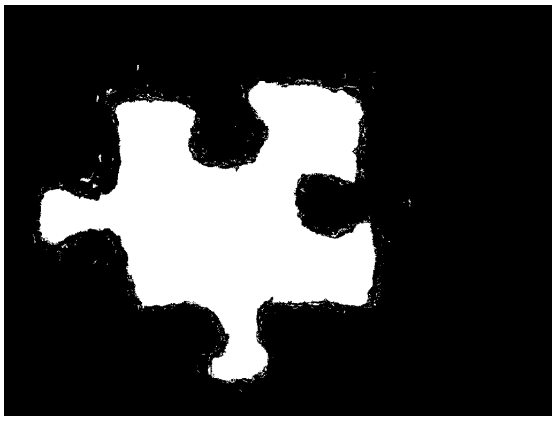

Fig. 9. U-Net No Augmentation result