# COMS4036A/COMS7050A Project: GMMs and U-Nets

Matthew Kruger
1669326
*School of Computer Science and Applied Mathematics*
*University of the Witwatersrand*
Johannesburg, South Africa

Jared Harris-Dewey
1846346
*School of Computer Science and Applied Mathematics*
*University of the Witwatersrand*
Johannesburg, South Africa

## I. Gaussian Mixture Models (GMMs)

### A. Problem Domain

Gaussian mixture models (GMMs) lie in the domain of unsupervised machine-learning as it attempts to identify data points of similar characteristics. In the case of classifying puzzle pieces pixels as foreground and the rest of the image that is not the puzzle piece as background, we aim to do supervised learning. The usage of GMMs here is thus a blend of unsupervised and supervised machine learning, as we attempt to find the characteristics of a background piece as well as the foreground piece using GMMs (which is quite similar to fuzzy c-means clustering). Then using these characteristics of background and foreground from our GMMs we hope to be able to tell whether a specific pixel is foreground or background. We will use Bayesian inference. Using this model (GMMs) and our inference technique (Bayesian) we hope to be able to create accurate masks of puzzle pieces, which separates the foreground from the background.

### B. Expectation Maximisation Algorithm

The *Expectation Maximisation* algorithm is an algorithm that uses a given set of data to set a statistical distribution's parameters to be that of the most most likely set to have generated the data [1]. The algorithm does converge but this convergence is not guaranteed to be a global maxima and at times can be a local maxima instead [2]. The algorithm is characterised into two main steps, firstly in an expectation step we assign responsibilities to each of the clusters. These responsibilities in essence represent how "responsible" or rather how "likely" that a specific distribution was in creating a specific data point. This step is characterised by the following equations:

$$r_{ik} = \frac{\lambda_k \operatorname{Norm}_{x_i}[\mu_k, \Sigma_k]}{\sum_{j=1}^{K} \lambda_j \operatorname{Norm}_{x_j}[\mu_j, \Sigma_j]} \tag{1}$$

Where $k$ is the number of distributions (clusters), $i$ is the specific data point, $r_{ik}$ represents the responsibility distribution k takes for data point $i$, $\mu_k$ represents the distribution's mean and $\Sigma_k$ represents the distribution's covariance.

The maximisation step is then done through the use of the following equations:

$$\lambda_k^{t+1} = \frac{\sum_{i=1}^{I} r_{ik}}{\sum_{j=1}^{K} \sum_{i=1}^{I} r_{ij}} \tag{2}$$

Here $\lambda_k^{t+1}$ represents the proportion of data of which is believed to have been generated by distribution $k$ at time step $t+1$.

$$\mu_k^{t+1} = \frac{\sum_{i=1}^{I} r_{ik} x_i}{\sum_{i=1}^{I} r_{ik}} \tag{3}$$

Here $\mu_k^{t+1}$ represents the mean of distribution $k$ at time step $t+1$.

$$\Sigma_k^{t+1} = \frac{\sum_{i=1}^{I} r_{ik}(x_i - \mu_k^{t+1})(x_i - \mu_k^{t+1})^T}{\sum_{i=1}^{I} r_{ik}} \tag{4}$$

Here $\Sigma_k^{t+1}$ represents the covariance of distribution $k$ at time step $t+1$.

These parameters are updated until we reach a point where the Euclidean norm ($L2$ norm) between the current (parameters at time step $t+1$) set of parameters for all distributions minus the previous set of parameters (parameters at time step $t$) is less than a specified threshold given by $\epsilon \in R$.

### C. Methodology

To split our data sets we have used 5-folds. In this case, eight images and their respective masks are chosen from the set of puzzle pieces. The selection of these first eight pieces is random. The remainder of indices possible are stored in an array - the indices not reserved for testing. Here we use 5-folds, by sliding over the remaining indices to select 8 validation images and 32 training images in each slide. For each slide the set of validation images is mutually exclusive and the union of the set of validation images would be the set of the remaining indices left for training. It should also be noted that in the case of creating a DoG feature we made use of Open-CV and in the case of converting images to grayscale or HSV we made use of Skimage.

To train our model, we will create both a background and a foreground set of GMMs. These GMMs will be trained exclusively on background or on foreground. The

way that we extract background/foreground will be through the use of the given masks. Foreground data will be flattened from a set of training images with their features concatenated along the last axis, with the first axis being every individual feature, like-wise this will similarly be done for the background. The shape of this result is the following: number of foreground/background pixels in subset of data by the number of features. The GMMs will be initialised randomly using *numpy.random.randn* which samples from a univariate normal distribution with a mean of 0 and a variance of 1 [3]. Training will be done using the multivariate normal distribution which makes use of the Scipy Stats package [4]. We will randomly initialise our covariance and mean using *numpy.random.randn*. Training will continue until parameters for the lambda values, means and covariances for each centroid in the GMM converge within a distance of $0.001$ (thus $\epsilon = 0.001$) between two consecutive iterations. This distance will be calculated using the $L2$ norm.

Similar to how an image's pixels are classified in training we will do the same for validation. Using the set of trained foreground and background GMMs we will perform Bayesian inference to classify whether a piece is foreground or background. The probabilities from the inference will be thresholded by $\theta \in R$. The optimal $\theta$ value will be chosen for a given model by searching through a linear space of range $[0, 1)$ with step increments of $\frac{1}{1000}$, therefore, $\theta \in \{0, \frac{1}{1000}, \frac{2}{1000}, \frac{3}{1000}, \cdot, \frac{999}{1000}\}$. Once an optimal $\theta$ has been found for the model, it will be returned alongside the accuracy of the model. The accuracy of the model will be the average accuracy across all pixels in the validation set.

Testing will be done similar to how validation is done. The key difference is that we will select the model which has the highest accuracy to be used for testing. The model will also make use of the optimal $\theta$ value as determined during validation. The accuracy of the training will be recorded.

For training we will recreate the model ten times, timing the amount duration it takes to create and train the model. Training here includes the time it takes for the GMMs (both background and foreground) to settle on the optimal hyper-parameters - when they are within 0.001 distance of each other using the $L2$ norm. The time taken to train will then be averaged over these ten runs and this will be recorded. In regards to the time taken to perform inference we will average this over ten runs as well, however it should be noted that this is not necessary given that it is not stochastic. The time taken to perform inference will be averaged over ten runs and this time will be recorded.

For our extra feature set, we chose to include HSV. This is because HSV did exceedingly well in the labs and hence it was of interest to see how well it would perform here with our GMM-based models. The following breaks down the five models we trained and validated in order to determine the best model. In breaking the models down we will specify their characteristics as follows; number of background distributions, number of foreground distributions and features chosen. Furthermore, we note that below RGB is Red Green Blue colour representation for pixels, DoG is the difference of Gaussians between of a grayscale image where the kernel sizes are 3 and 6 with the Gaussian of kernel size 6 has the Gaussian of kernel size 3 subtracted from it, and finally HSV which represents the Hue Saturation Value representation of the image.

- 2, 4, RGB and DoG
- 4, 8, RGB
- 6, 12, RGB, DoG and HSV
- 8, 16, RGB
- 8, 16, RGB, DoG and HSV

### *D. Experimental Setup*

The experiment was run on a computer with the following notable specs: Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, 16 GB of DDR3 RAM and Windows 10 64-bit Operating System. Furthermore, the code was written in Python.

### *E. Evaluation of Results & Discussion*

The outputs and results for this section can be found in the appendix.

TABLE I: Accuracy and Theta Values of Models

| | Model 1 | Model 2 | Model 3 | Model 4 | Model 5 |
|---|---|---|---|---|---|
| Accuracy | 98.57% | 96.20% | 96.44% | 98.28% | 95.94% |
| Optimal $\theta$ | 0.4484 | 0.0581 | 0.2022 | 0.7457 | 0.01001 |

The best model is the first model which is the simplest. It should however be noted that the initialisation in the *Expectation Maximisation* algorithm plays a key role. Although a local maxima is discovered finding a global maxima which provides the best hyper-parameters is not guaranteed and hence multiple initialisations are often required to find the optimal hyper-parameters. In our case we have noticed that typically the models have comparable performance and finding the true best model may require a large average number of runs. The lowest accuracy achieved by any of the five models was 94.41%, with the greatest accuracy recorded being 98.57%. We note that over five runs the best model was continually model 1 followed by model 4 with model 5 consistently performing the worst. Anecdotally we might consider that this result is due to the "curse of dimensionality", which would imply that adding more features in general makes for worse models in clustering algorithms.

TABLE II: Confusion Matrix on Testing Data

| | Predicted 0 | Predicted 1 |
|---|---|---|
| Actual 0 | 4618374 | 56306 |
| Actual 1 | 87039 | 1529737 |

The Cohen's Kappa ratio we got on our testing data was 0.93996 and we had an Intersection over Union of 0.9143. This showcased that our model had excellent performance.

Thus, we can see that despite all models 2, 3 and 5 providing more information, they were not as competitive as models 1 and 4. These models made use of different setups but showcase some notable issues with "overcomplicating" the model. The first model uses only two distributions for background, and four for the foreground, furthermore it uses RGB and DoG which totals at 4 features. This model is much simpler than that of models three and five which have 7 features each and many more distributions. It seems that the complexity of the models hurt their performance. In regards to model four, we see that adding extra distributions still retained high accuracy (and is nearly the same as model one), this should be assumed it is due to the reduced feature set (as it only has 3 features). In this case we can assume that model one and four did not succumb to the "curse of dimensionality" and did not overcomplicate the model, and hence their performance was better.

*Testing performance:* Using the first model which performed the best we applied it on testing data. In the testing data, the model achieved an accuracy of 97.72%. The average time taken to train the model was found to be 54.00 seconds and it took an average of 2 iterations. Furthermore, the average time taken to perform inference was 4.966 seconds.

### *F. Issues and Resolutions*

There are a few notable problems with GMMs that should always be kept in mind, not all of them have perfect solutions but solving the problems is possible in practice, although, these practical solutions themselves are not perfect and may at times be tedious. Firstly, the GMMs do not guarantee an optimal model and the optimal global parameters for the distribution may not be discovered [2]. This issue's solution is to create multiple GMM models and record the model's initialisation which provides the greatest lower bound for the maximum likelihood expectation of the parameters. This is in practice tedious but provides a better solution. Alternatively, because the GMM model does make use of maximum likelihood expectation in determining the hyper-parameters, there can exist cases where a single point is associated to a single distribution and hence, the model will be highly certain that a specific point is associated to a single distribution. This problem can be solved by taking into account the priors of the distribution's hyper-parameters which prevents the scenario where a single distribution is exclusively associated to a specific point. Another limitation is that the model is reliant on the initialisation of hyper-parameters - specifically the covariance matrices. Typically the covariance matrices can be singular and hence may make sampling from the distribution more difficult using Scipy. A workaround for this is to allow for singular matrices. Another workaround we found was in initialising the parameters slightly differently which circumvented this issue. The initialisation using a matrix of random numbers which come from a univariate normal distribution with a mean of 0 and a variance of 1, but then augmenting the initial values by adding the identity matrix (of same size as the number of features) multiplied by the number of features. This appeared to circumvent the issue of singular matrices.

Our last set of issues deal with the result of Gaussian mixture models as seen in III-A. Here we see there is an issue with regards to context and the nature of the problem. Many puzzle pieces have holes in them. These holes are often caused by the hair of one of the characters (specifically the girl with the brown hair). Some of the blondish segments of her hair caused confusion in the model as it blended in quite well with the background. Thus, the model assumed that her hair in these cases, must be background. This shows that there may be issues in the case where two distributions compete for similar data points. Furthermore, it brings about the issue of context in GMMs. Having holes in a puzzle piece makes no sense, however, the generated mask proposed by the GMM claims that masks should have holes. A simple and easy way to resolve this issue would be to perform a closing morphological operation (using a large kernel) on the masks that were created. Finally, in determining the number of clusters or the amount of data (features) to use in the GMM model can be quite difficult. This is because adding more clusters can lead to overfitting and lack of generalisation as some distributions may end up being responsible for shades of one colour - for example hues of beige or tan. Furthermore, adding more features brings about the "curse of dimensionality" which is ever present in clustering algorithms and thus selecting the optimal number of features is difficult in general and varies from problem to problem, and hence, requires the use of extensive validation with multiple models. Similarly, choosing the optimal number of distributions in each problem is very similar to "choosing k in K-Means" and thus, this requires extensive validation for each model to find a reliable set up, which is dependent on the problem being solved.

### *G. Conclusion*

We have showcased the performance of GMMs with variance hyper-parameters with the focus on classifying a puzzle into foreground (puzzle piece) and background (not puzzle piece). We saw that the GMMs provided decent solutions but were not perfect and that there exist notable limitations due to the clustering nature of the algorithm. Furthermore, we note that the method is quick, however, these results should be refined further if higher accuracy is desired. One possible refinement is to make use of closing operations on the final result.

## II. U-Nets

### *A. Problem Domain*

Our second implementation to solve the problem where we need to separate puzzle piece pixels from background pixels is the U-Net architecture. The U-Net architecture is well suited for the segmentation problem of images. The

U-Net is a convolutional network architecture that consists of three sections. The contraction, the bottleneck and the expansion section. Also known as the encoder, bottleneck and decoder sections. The design of this architecture allows it to extract necessary features to perform the segmentation using a supervised method of learning.

### B. Data Preparation

There are a total of 48 images and masks that get shuffled and split into training, validation, and testing using 70%, 15%, and 15% split respectively. If the data is not perfectly divisible by those percentages, the remainder is given to the test split.

The data is then split whilst creating a data loader which has the option to augment the data in which case we perform N rotations for every image, where N is randomised between 3 and 10 for each new image. This then allows the image to be rotated in varying amounts of degrees to ensure that the U-Net model does not see every image rotated by the same angle. Other data augmentation methods were considered but options like scaling and colour changes were rejected since we wanted images of the same resolution going through the network and changing the colour might cause issues since the model wouldn't be expecting a an entire right pink puzzle piece.

### C. Methodology

*Architecture:* The first half of the network called the encoder makes use of the pre-trained VGG16 network. The encoder side then performs convolutions upon the image as well as pooling. This happens for 5 steps with the image being halved each time. Each step removes spatial information and increases the number of channels until the bottleneck. From the bottleneck, the network goes into the expansion section or decoder section which increases the spatial information and decreases the number of channels in 5 decoder steps. We implemented the decoder section using decoder blocks. Our decoder blocks are created using an up-sample, padding and convolution step rather than a convolution transpose as this has been shown to prevent the checkerboard artefacts in the output image [5]. We also make use of Instance Normalisation as normalisation has been shown to decrease the time needed to train a network [6]. Finally, we use a ReLU as our activation in the decoder blocks, until the very last decoder block which uses a sigmoid activation function to ensure values are between 0 and 1. We also make usage of skip connections to help the architecture with the reconstruction of the spatial information. Connecting all of this together we get our U-Net architecture.

*Training:* The architecture was trained using an I5-8300 CPU, 24GB RAM, and a GTX 1050 4GB. We train the architecture by feeding an image into the network and generating an output. We then make use of our ground truth images to calculate an error using the Binary Cross Entropy with Logits. This is then used to update the network in a supervised manner. We make use of the ADAM optimiser to update the network with a step size of 0.0002 and setting betas = (0.5, 0.999) which was used in the Pix2Pix paper for updating their U-Net [7]. After the training of each epoch, we perform a validation step which saves images with their ground truth for comparison. Finally, after a total of 6 epochs, we perform a test step on which images that the model has never seen are pushed through the model and accuracies are recorded as well as images are saved.

*Limitations and Resolutions:* Due to the large size of the images, some adjustments were made as originally, the full image was not able to go through the entire network as the amount of VRAM needed was too much. We made use of a technique called gradient checkpointing to overcome this issue. Gradient checkpointing works by recalculating some of the values in the backward calculation rather than storing them in the forward calculation. This will come at the cost of computational time, but allow us to fit the full resolution image in the network. We also made use of a mixed model for training using 16-bit and 32-bit precision to train the network as this has been shown to significantly reduce the memory requirement whilst not affecting the network [8]. We also experienced that training for over 6 epochs usually led to the model collapsing.

### D. Evaluation of Results & Discussion

TABLE III: Accuracy on Images

| | Image 5 | Image 25 | Image 28 | Image 44 | Average |
|---|---|---|---|---|---|
| Not Augmented | 99.56% | 98.84% | 99.43% | 99.49% | 99.34% |
| Augmented | 99.7% | 97.74% | 99.67% | 99.76% | 99.42% |
| Not Pre-trained | 97.52% | 84.31% | 84.78% | 98.34% | 94.03% |

As can be seen from the listed accuracies in Table III, the augmented model performed the best on the test images for all images except image 25. Figure 8 demonstrates the plot of the accuracies over the images for the augmented and non-augmented models. Both the augmented data model and the non-augmented data model struggled on image 25 with the augmented data model performing the worst of the two. It is possible that training for the same amount of epochs but on a larger dataset caused the augmented model to overfit more and hence gave a worse output than the non-augmented output, but overall, the augmented data model was the better of the two on average. Averages were based on the full testing data.

Looking at figure 7, we can see that when using the pre-trained VGG16 model our training converged faster (Blue and Orange plots). The training started to oscillate after having trained over 200 steps and this was true for both the augmented and non-augmented model, but the augmented model trained for more steps as both models ran for 6 epochs and the augmented model had more images that it had to

train over for 6 epochs.

The red plot in figure 7 was trained on the augmented dataset but using a non pre-trained VGG16 model. It is clear that the pre-trained model helped the model to converge faster and more stably than the non pre-trained model.

### *E. Comparison with GMM and previous models*

The U-Net model managed to achieve a higher accuracy on average compared to the GMM. More effort had to be given in the creation of the GMM as handcrafted features had to be created whereas the U-Net was able to self-learn features.

Compared to previous models like the generative model that was created in lab 3, the Unet also managed to outperform this model on average and was significantly more easier to implement whilst being able to generalise to a larger problem since no handcrafted features are required.

### *F. Conclusion*

We have shown that U-Net managed to segment images better than the GMM without the need of having to create handcrafted features. We have also shown that the usage of a pre-trained model using augmented data was on average better than if we had not used either of the two methods, and therefore the U-Net is a good architecture to make usage of in image segmentation problems.

## REFERENCES


[1] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[2] C. F. J. Wu, "On the convergence properties of the em algorithm," *The Annals of Statistics*, vol. 11, no. 1, pp. 95–103, 1983.

[3] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[4] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[5] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and checkerboard artifacts," *Distill*, 2016.

[6] Y. Wu and K. He, "Group normalization," 2018.

[7] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," 2018.

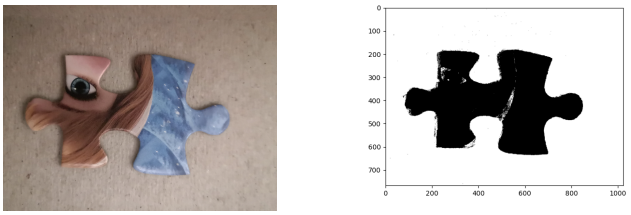
[8] C. C. Case and M. Carilli, "Nvidia apex: Tools for easy mixed-precision training in pytorch," Dec 2018.


## III. APPENDIX

The work was divided in the following way:

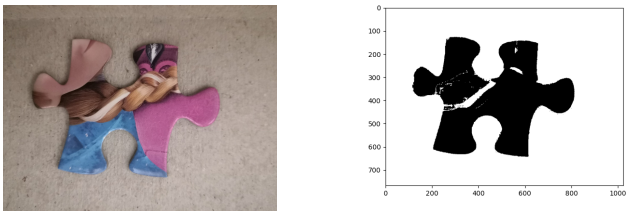
- 1669326: Gaussian Mixture Models
- 1846346: U-Net

### *A. GMM Figures:*



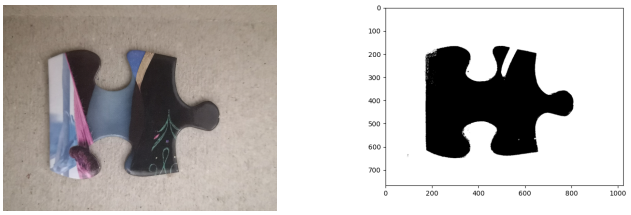
(a) Puzzle Piece (b) Model's Mask

Fig. 1: First Model



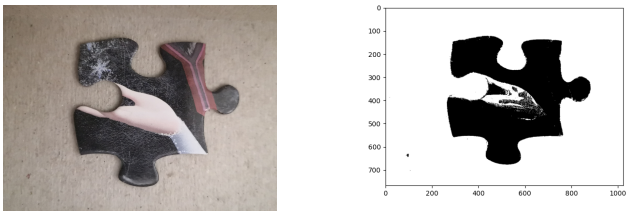
(a) Puzzle Piece (b) Model's Mask

Fig. 2: Second Model



(a) Puzzle Piece (b) Model's Mask

Fig. 3: Third Model

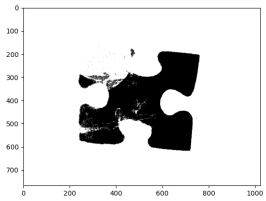


(a) Puzzle Piece (b) Model's Mask

Fig. 4: Fourth Model

### *B. U-Net Figures:*

(a) Puzzle Piece



(b) Model's Mask

Fig. 5: Fifth Model

(a) Puzzle Piece

(b) Model's Mask

Fig. 6: Test Model

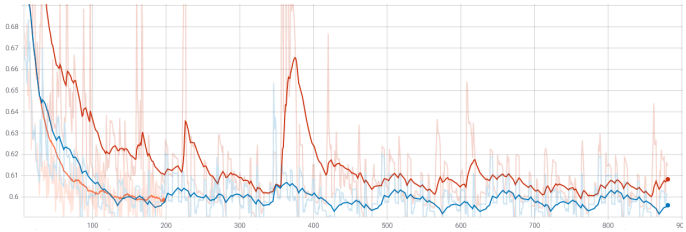


Fig. 7: BCE Training Loss Graph

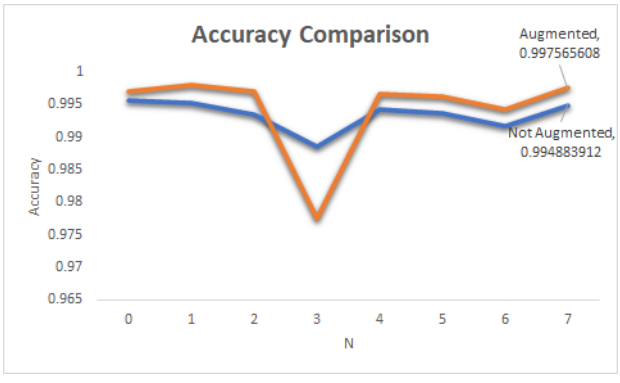



Fig. 8: Accuracy Graph

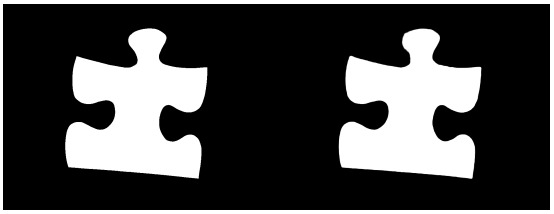


Fig. 9: Image 5 - Left: Ground Truth and Right: U-Net

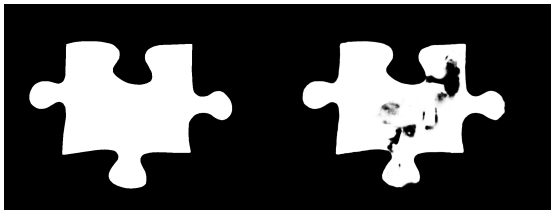


Fig. 10: Image 25 - Left: Ground Truth and Right: U-Net

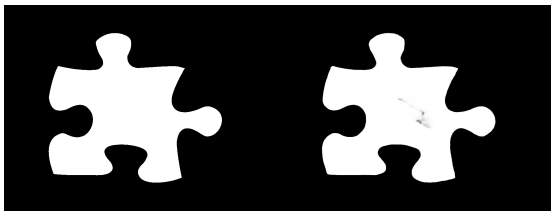


Fig. 11: Image 28 - Left: Ground Truth and Right: U-Net

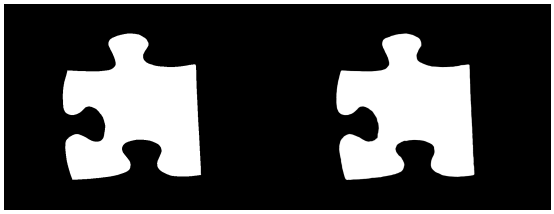


Fig. 12: Image 44 - Left: Ground Truth and Right: U-Net