Revisiting Background Segmentation: Gaussian Mixture Models & Deep Learning

1st Steven Curtis (1657041)
University of the Witwatersrand
Johannesburg, South Africa
1657041@students.wits.ac.za

2nd Christopher Pillay (1362077)
University of the Witwatersrand
Johannesburg, South Africa
1362077@students.wits.ac.za

***Index Terms*—GMM, Unet, Gaussian mixture models, Deep Learning, background segmentation, image masks**

I. Introduction

Over the year, we have trained several different binary classifiers to automate the mask creation of the puzzle pieces. Our classifiers used RGB, HSV, Haar-like features, Local Binary Patterns, Textons, Difference of Gaussians, Laplacian of Gaussian, Histograms of Oriented Gradients and MR8 features to obtain information from the puzzle pieces that would help us differentiate between the different types of pixels. A generative model was then built where we used multivariate normal distributions to represent the class conditional probabilities as well as a Bernoulli prior. Bayes' rule was used to calculate the posterior distribution to tell us the probability that a pixel was part of the puzzle piece (foreground) or the background. Two assumptions that we made were that the extracted features were normally distributed within each class and that the hand-crafted features would contain the correct discriminative information for the model to learn and generalise effectively. To overcome these assumptions, we implement a Gaussian Mixture Model which achieves a main result of 97.84% accuracy on a RGB feature set and a Deep Learning approach which achieves a 99.72% accuracy on the same RGB feature set.

II. Data Segmentation

To implement the two approaches mentioned above, we need to first segment our puzzle piece data. The dataset used consists of 48 RGB images and 48 corresponding masks, where each of the images contain an individual puzzle piece of a complete puzzle. The dimensions of each of the RGB images and masks are 1024*768 and the data is randomly split(using a train-test split function), into the training dataset which contains 34 images, the validation dataset which contains 7 images and the test dataset which contains 7 images. The data segmentation is done within each of the implementations code files as we split the data using the unique image and mask ID's and load the images and masks as required.

III. Gaussian Mixture Models

A. Past Models

Throughout the labs this year we have predicted the background of puzzle piece images to create a binary mask image. Different feature vectors have been used in each of these models. A prior model that was used for this prediction involved a background classifier that used a multivariate normal probability density function to predict the background of a puzzle piece using the mean and covariance from our feature vector. Feature vectors that were used contained the vertical and horizontal Prewitt filtered image, the Laplacian filtered image and the RGB and HSV images. This background classifier leads to a prediction accuracy of around 94%.

In another implementation, we used feature vectors that included the MR8 filter bank, local binary patterns, Haar filter, the vertical and horizontal Prewitt filtered image, the Laplacian filtered image and the RGB and HSV images. We then used this feature vector with the K Nearest Neighbour classifier to predict the Texton with 4 clusters and added this to the feature vector. The multivariate normal probability density function was then used to predict the background of a puzzle piece using the mean and covariance from our feature vector with an accuracy of around 87% in our implementation.

B. GMM Methodology

To address the assumption of unimodal normal distributions over our features in previous classifiers, a Gaussian Mixture Model was implemented. A Gaussian Mixture Model is a marginalisation of the joint probability distribution and is a probabilistic model. This model uses the expectation-maximization (EM) algorithm for fitting Gaussian mixture models. Our implementation of GMM is as follows. We first split the puzzle image into training, validation and testing datasets as described in the above section on data splitting. Three different feature sets were created as follows:

- RGB image only
- RBG and Difference of Gaussian images. Here, DoG is created using the formula $g(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}} - \frac{1}{2\pi K^2\sigma^2}e^{-\frac{x^2+y^2}{2K^2\sigma^2}}$ where K is a hyper-parameter used to ensure that the subtracted Gaussian have a larger variance than the first.

- Vertical and horizontal Prewitt filtered image, the Laplacian filtered image and the RGB and HSV images (we will call it the "Filtered" feature set for short). This feature set was chosen as it was used with a previously implemented classification model that leads to the high accuracy of around 92% so we can see how our implementation of the Gaussian Mixture Model compares to that accuracy.

For each of the three feature sets, the following was performed to implement the GMM:

- The background and foreground pixels of the puzzle piece in the training feature set are found using their respective mask images. The prior is also calculated using the following formula $prior = len(fg)/float(len(fg) + len(bg))$ where bg and fg are background and foreground pixels respectively. This is done for all images in the training feature set with each image's foreground pixels being stacked to obtain one foreground pixels feature set. The same is done for our background pixels and all the priors are all added together.
- We then use these background and foreground feature sets for training our two models, one for background pixels and a separate one for foreground pixels. We train these models by first creating our background and foreground GMM models by calling our GMM class for each model.
- In our GMM class, our model and the feature set, length of feature set, number of components, lambda, mean, covariance and the responsibility are all initialised with the covariance matrix always being positive definite. To create a positive definite covariance matrix, we create a random n x n matrix (A) and transform it so that it is symmetric (A *= A.T), and then add some positive multiple of the identity matrix (A += n*np.eye(n)) with the identity matrix being multiplied by a large positive number (10000000) so that the initialisation gives the distributions a large covariance to start with. The number of components used controls the number of cluster components that the data will be partitioned into. It is important to select the parameters that lead to the best accuracy as described later on in the Validation section.
- These two models are then trained. Firstly we perform the estimation step by updating the responsibility using the formula in Figure 1. We then perform the maximisation step which updates the lambda, mean and covariance matrix using the last three formula in Figure 1 respectively. To avoid dividing by 0 in this step, a small noise value (0.0000001) is added to the denominator of each formula. We continue to do these two steps until the difference between the old mean and the newly updated mean is less than our tolerance and training of the two models is complete.
- Now that we have our background and foreground trained models, we need to predict our validation images. To do this, we find the likelihood of each model using our validation image and use the prior we calculated before and 1-prior. To predict the foreground and background, we use the formula $\frac{foreground\ likelihood*prior}{(foreground\ likelihood*prior)+(background\ likelihood*(1-prior))}$. In this predicted image, if the pixel value is > 0.5, its pixel value is 1 else it is 0 as we want a binary image.
- Now that we have predicted our validation images, we then find the accuracy of this prediction by comparing it to the real mask for the images. We run many validation tests with different values for our hyperparameters to obtain the parameter values that lead to the best accuracy. Once these values have been found, we train our models on these optimal parameters and predict our test images and obtain the final accuracy for the GMM on the feature set.

$$r_{ik} = \frac{\lambda_k \text{Norm}_{\vec{x_i}}[\vec{\mu_k}, \mathbf{\Sigma}_k]}{\sum_{j=1}^{K} \lambda_j \text{Norm}_{\vec{x_j}}[\vec{\mu_j}, \mathbf{\Sigma}_j]}$$

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

$$\vec{\mu}_k^{[t+1]} = \frac{\sum_{i=1}^{I} r_{ik} \vec{x}_i}{\sum_{i=1}^{I} r_{ik}}$$

$$\mathbf{\Sigma}_k^{[t+1]} = \frac{\sum_{i=1}^{I} r_{ik} (\vec{x}_i - \vec{\mu}_k^{[t+1]})(\vec{x}_i - \vec{\mu}_k^{[t+1]})^T}{\sum_{i=1}^{I} r_{ik}}$$

Fig. 1: Formula used in the GMM training process.

C. GMM Validation Results

The number of components used in our implementation was chosen to be 3 for the RGB feature set and 4 for the others with a tolerance of 0.01. These values were found through validation by trying 2, 3, 4, 5, 6, 7 and 8 number components with a combination of different tolerances 1, 0.5, 0.1, 0.05, 0.01, 0.005 and 0.001. It was observed that for the RGB feature set, the best accuracy was with 3 components and 0.01 tolerance. For the RGB and Difference of Gaussian feature set, the best accuracy was with 4 components and 0.01 tolerance and finally with the last feature set, the best accuracy was with also 4 components and 0.01 tolerance. Some examples of the found accuracies through the validation process can be seen in Table I below. A 6-fold cross-validation was also done on the feature set to obtain the best parameters. To do this, the training dataset was shuffled and split into 6 groups. Then the models were trained on all but one of the groups as that remaining group was then used to test the model. This was then run 6 times so that each of the 6 groups was the testing group once. The accuracies were recorded and the average accuracy of the 6 runs was highest, for example, when the number of components was 3 and tolerance 0.01 for the RGB feature set (94.56%).

TABLE I: Validation Accuracies for different feature sets

Feature Set	# components	Tolerance	Accuracy
RGB	2	0.1	96.58
RGB	5	0.001	97.94
RGB	**3**	**0.01**	**98.63**
RGB + DoG	2	0.01	98.76
RGB + DoG	3	0.001	99.09
RGB + DoG	**4**	**0.01**	**99.14**
Filtered	2	0.01	89.94
Filtered	5	0.001	90.32
Filtered	**4**	**0.01**	**91.28**

D. *GMM Testing Results*

Now that we have our optimal parameters, the models for each feature set are tested. The accuracies are as follows:

- For our RGB feature set, we obtained an accuracy of 97.82%. This means that our implementation of GMM has higher accuracy on the puzzle pieces than in our previous classification models that we have done this year as stated in the "Past Models" section.
- For our RGB and Difference of Gaussian feature set, we obtained an accuracy of 98.06%. This means that our implementation of GMM has higher accuracy on the puzzle pieces than in our previous classification models as well as our RGB only feature set.
- Our "Filtered" feature set obtained an accuracy of 91.14%. This is the lowest accuracy of the three feature sets but still improves upon the classification model that involved KNN as stated in the "Past Models" section and is very close in accuracy to the previous binary classification model that used this same feature set in its classification. This model struggled most with classifying foreground pixels where colours on the puzzle piece were close to that of the background.

  Examples of some of the predicted masks along with their ground truth can be seen in Appendix B.

TABLE II: Test Accuracies for different feature sets

Feature Set	# components	Tolerance	Accuracy
RGB	2	0.01	97.82
RGB + DoG	4	0.01	98.06
Filtered	4	0.01	91.14

E. *Fixed Issues*

Training time was an issue as some for loops could not be avoided and increased the computational time compared to the previously created models. Inference time was relatively fast and was similar to that of the previous models. The images were therefore scaled for computational reasons in the same way as a previous task this year. Scaling the images lead to no major effect on accuracy with a change of around 0.05% between accuracies with images of a 0.5 scale and a 0.4 scale. We chose our final scaling factor to be 0.4 (the higher of the resulting accuracies). To avoid dividing by 0 in the training step, a small noise value (0.0000001) was added to the denominator of each formula. To avoid singular covariance, we checked to see if the determinant of the newly calculated covariance matrix is 0 which means it is singular and if so, add a small number to the covariance diagonal (0.000001). This will cause the determinant to no longer be 0 and so we will avoid any singular matrix issues.

TABLE III: Test Accuracies at 0.5 scale

Feature Set	# components	Tolerance	Accuracy
RGB	2	0.01	97.84
RGB + DoG	4	0.01	98.04
Filtered	4	0.01	91.11

IV. Deep Learning

A. *U-Net Architecture*

The U-net architecture was developed by Olaf Ronneberger with the original purpose of biomedical image segmentation [1] and improves upon FCNs which were first proposed by [2]. The U-Net is comprised of a contraction path (left side of network) and an expansion path (right side of network) which can be seen to give rise to the unique "U" shape of the network in Fig. 3 in Appendix A. The contraction path in our network consists of encoder blocks which are made up convolution layers, batch normalization layers, Relu activation layers and max pooling layers which are responsible for downsampling, this path aims to decrease the spatial information and increase the contextual information. The expansion path consists of decoder blocks in our network are made up of convolution layers, batch normalization layers, Relu layers and transposed convolution layers which are responsible for upsampling, this path serves the purpose of combining contextual and spatial information through upsampling as well as concatenations through the skip connections as seen in the figure. Hence this network is able to achieve impressive results in terms of segmentation, improves on boundary localization, and works well with small datasets.

B. *Deep Learning Methodology*

Before training and initialization of the semantic segmentation network can begin we set our parameters and required variables, the number of channels is set to 3 to be able to handle our RGB input, the stride for testing is set to 32, a label array is initialized and set with "background" and "foreground" as our two labels, the number of classes is set to the length of the label array which is 2, our base learning rate is set at 0.01, the augmentation variable which decides whether the input images undergo augmentation is set to true and the cache variable which decides whether or not our dataset is stored in memory (further explanation below) is also set to true. The dataset is then split as described above and data visualization is done to view a sample of the data. Utility functions, metric functions, the dataset class and the

U-Net model are then created.

The U-Net model is then initialized with the number of classes and pre-trained weights from the VGG16 model are loaded and set for the encoder blocks in the network, while the decoder weights are initialized using the HE policy [3]. The model is then loaded onto the GPU which was a Geforce 1060 6GB. Our decoder blocks are then set to be trained at the base learning rate while the encoder blocks are set to be trained at half the base learning rate due to the VGG16 weight initialization. Stochastic gradient descent was chosen as the optimization algorithm for the optimizer that was then defined and had network parameters are passed to it when initialized.

The training dataset is then created by passing the 34 RGB image and mask IDs, from the output of the train-test split, to the dataset class which makes use of the data function from the torch utils library. The images are normalized and added to a cache if it has not been encountered before, a random patch of the RGB image and corresponding mask, of size 256 by 256 is then selected and passed to the augmentation function, if desired, which randomly applies a flip or mirror transformation to the patch before returning it. The dataset size is set to 10000 per epoch within the dataset class. The dataset is then passed to a dataloader function from the torch utils library with a set batch size of 10, this object handles the loading of the data during training of the model.

The network and optimizer is then passed to the train function with the number of epochs set to 5 with each epoch running a thousand iterations. Within the train function empty arrays are initialized to store training and validation loss values, the actual training occurs with two for loops, the outer for loop manages the number of epochs which are run. Before the second for loop is entered the network is switched to training mode, in the second for loop the dataloader provides a batch of 10 patches of the RGB images and their corresponding masks, the optimizer is first cleared of any accumulated gradient values and the patches of RGB images are then passed to the network for training. Output of the network and masks are then passed to cross entropy loss function which is then followed by adjustment of weights. Training loss values are then stored and every 100 iterations validation occurs and its loss values are stored. Summary statistics of the training are also then printed out such as loss and accuracy for that iteration which helps monitor convergence. At the end of the epochs the train loss and validation loss is plotted and the model is saved. The loss plot helps us identify if overfitting occurred during the training.

The saved model is then loaded and passed to the test function with the test dataset IDs, the network is then switched to inference mode and each of the RGB images are split into patches and passed to the model for prediction. Once prediction is completed the predicted mask (semantic segmentation map) of the image is reconstructed from the patches. The predicted mask and ground truth mask are then flattened and passed to the metrics function which returns a confusion matrix, the F1 score for each class, the overall accuracy of the prediction and a Cohen kappa score. All predicted masks are then saved and displayed. Lastly, cross validation is then performed with the KFold function from the sklearn library. The tiling training method and helper functions were adapted from [4].

C. Deep Learning Results

1) Effect of Data Augmentation: We first investigated the effect that data augmentation has on the model and accuracy.

TABLE IV: Accuracy of U-Net with and without augmentation for different learning rates

	LR=0.01	LR=0.05
No Augmentation	94.97	85.48
With Augmentation	99.60	99.46

The model was trained for one epoch, which amounts to a 1000 iterations, with augmentation at two different learning rates and without augmentation at two different learning rates. The results of this test can be seen in Table. 4, from these results we are able to come to the conclusion that augmentation does have a positive effect on overall accuracy as the model achieves a significantly higher accuracy at both learning rates when augmentation is used compared to when it is not. Due to these results the remaining results from our deep learning model are based on the learning rate set at 0.01 and augmentation being utilized.

2) Training and Validation Loss: We have a graph in Fig. 2 which represents the training and validation loss that was recorded as the model was trained. At the beginning of training we can see that the validation loss and training loss vary greatly but as training continues the difference between the loss decreases and converges. This helps us confirm that our model is not over-fitting.

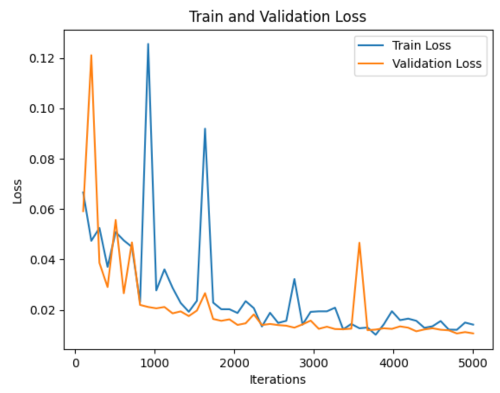

Fig. 2: Training and Validation Loss graph

TABLE V: Confusion Matrix for Test dataset

4063198	7776
7411	1426639

TABLE VI: F1 Score for classes

Background	99.81
Foreground	99.47

TABLE VII: Results

Total Accuracy	99.72
Recall	99.81
Precision	99.81
Kappa Score	99.28

3) Final Model Results: After running the final model the following results were achieved.

Table. 5 contains the confusion matrix for the model on the test dataset, by inspection alone we are able to observe that a large number of the pixels from our test dataset fall within the diagonal of the matrix which is a positive sign. The recall of our model which can be found in Table. 7 is high at 99.81%, thus for all the pixels which are actually background pixels, the recall tells us how many we have correctly identified as being background pixels, furthermore the precision in Table. 7 is also high at 99.81% which implies that the measure of pixels that we correctly identify background pixels out of all the pixels actually being background pixels. Our F1 scores in Table. 6 for both classes are high which indicates that the model is able measure incorrectly classified cases since false negatives and false positives are important, and in this model the percentage is small. The total/overall accuracy of our model is impressively high at 99.72% which can be seen in Table. 7 which implies that the model is a model of good quality for performing background segmentation on this dataset. The kappa score which can also be found in Table. 6 is also high at 99.28% which implies that there is almost perfect agreement between raters for the model and that the model is reliable. Appendix A contains figures of ground truth masks and their corresponding prediction masks from Fig. 4 to Fig. 17.

4) Cross Validation: Six fold cross validation is performed with the model for the purpose of confirming the ability of the model to accurately perform semantic segmentation on the given dataset. To implement this, each fold consists of a train and test dataset but each dataset differs among the folds. Table. 8 represents the accuracy achieved with the train and test set for each fold and we are able to observe that an accuracy of over 99% is achieved across all folds for the train and test sets, therefore the model has been proven to generalize well regardless of which part of the dataset was used to train and test the model.

TABLE VIII: Train and Test accuracy for Folds in Cross Validation

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6
Train accuracy	99.71	99.78	99.79	99.61	99.71	99.77
Test accuracy	99.67	99.81	99.81	99.64	99.73	99.74

V. Comparison of the Gaussian Mixture Model, Deep Learning Model and Models from Labs

If we refer to Table. 3, the GMM achieves an accuracy of 97.84% for the RGB test feature set whereas the deep learning U-Net model in Table. 7 achieves an accuracy of 99.72% and the best performance of our past model was 94%. The deep learning model therefore has higher accuracy of 1.88% than our GMM and 5.72% higher than our past model. The deep learning U-Net model also outperforms the GMM with regards to other feature sets such as the RGB + DoG and Filtered feature sets. We should note that although our past model had an accuracy of 94% it could not generalize well for images that differed greatly to the image it was trained on due to it being modelled on a multivariate normal distribution which is unable to capture and learn a variety of features like our GMM and U-Net models.

VI. Conclusion

The impressive performance of the deep learning model is largely due to the increased research and progress that has been made in the deep learning field over the past few years [5]. Further improvements can be made with regard to our deep learning model such as the use of different networks, optimization functions and the general training methods involved. Although, this project and the various labs that were completed have shown us that we have many other tools to consider when approaching semantic segmentation tasks.

A. Contribution

Steven Curtis, 1657041: **50%** (GMM implementation and write-up)
Christopher Pillay, 1362077: **50%** (Deep Learning implementation and write-up).

References

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[2] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[4] N. Audebert, B. L. Saux, and S. Lefèvre, "Beyond rgb: Very high resolution urban remote sensing with multimodal deep networks," *ISPRS Journal of Photogrammetry and Remote Sensing*, 2017.

[5] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, "A review on deep learning techniques applied to semantic segmentation," *arXiv preprint arXiv:1704.06857*, 2017.

VII. Appendix A

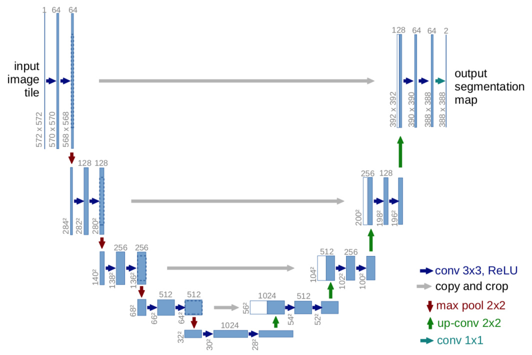

Fig. 3: UNet Architecture (source: [1])

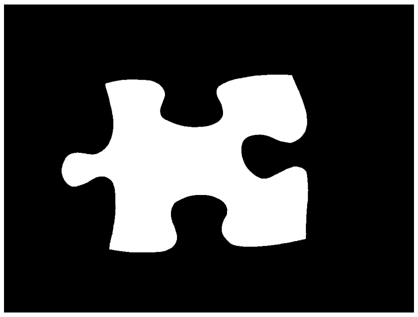

Fig. 4: Ground Truth mask Piece 11

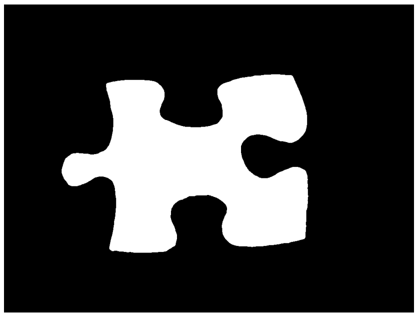

Fig. 5: Prediction mask Piece 11

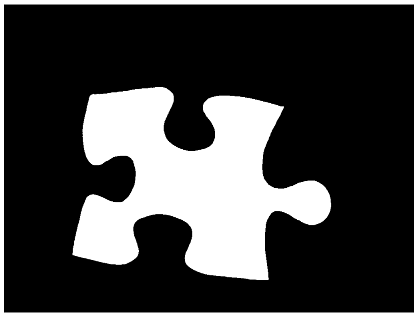

Fig. 6: Ground Truth mask Piece 12

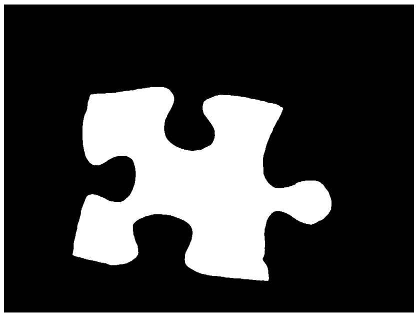

Fig. 7: Prediction mask Piece 12

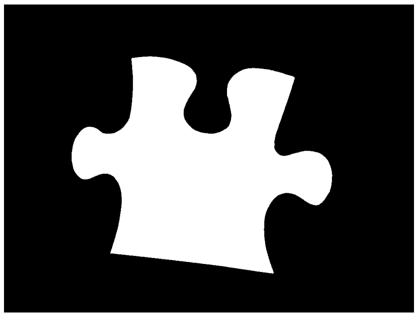

Fig. 8: Ground Truth mask Piece 14

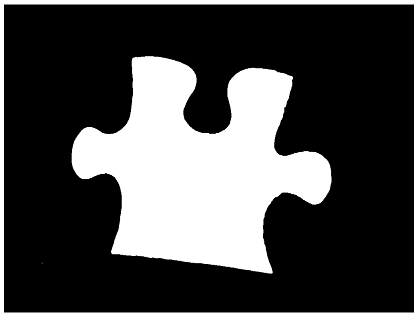

Fig. 9: Prediction mask Piece 14

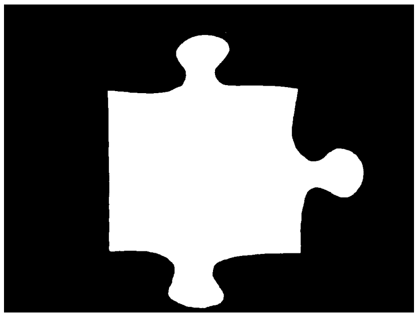
Fig. 10: Ground Truth mask Piece 31

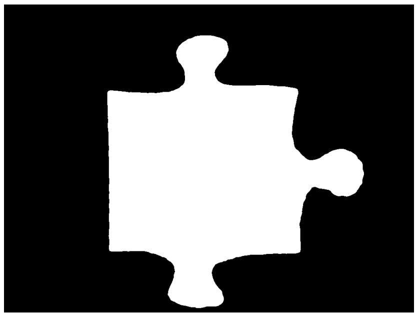
Fig. 11: Prediction mask Piece 31

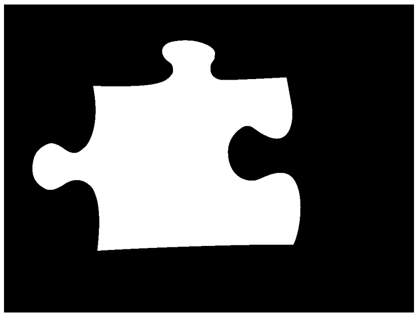
Fig. 12: Ground Truth mask Piece 32

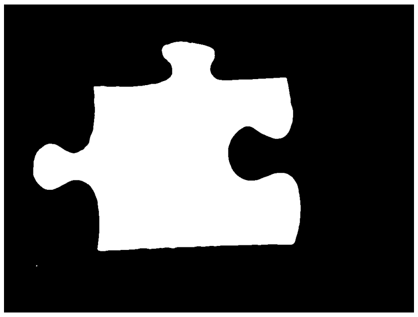
Fig. 13: Prediction mask Piece 32

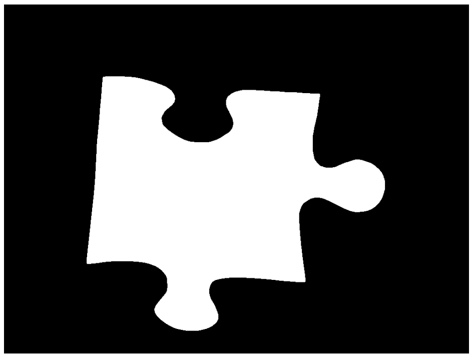
Fig. 14: Ground Truth mask Piece 33

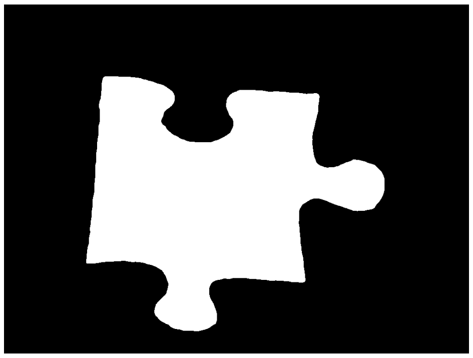
Fig. 15: Prediction mask Piece 33

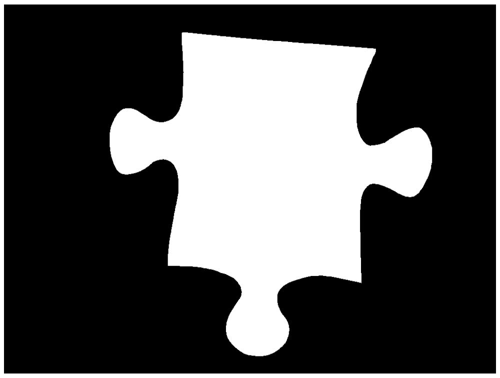
Fig. 16: Ground Truth mask Piece 42

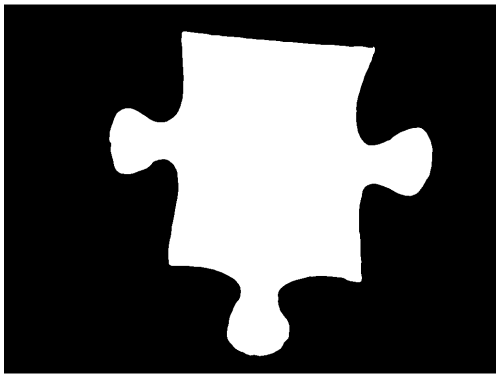
Fig. 17: Prediction mask Piece 42

VIII. Appendix B

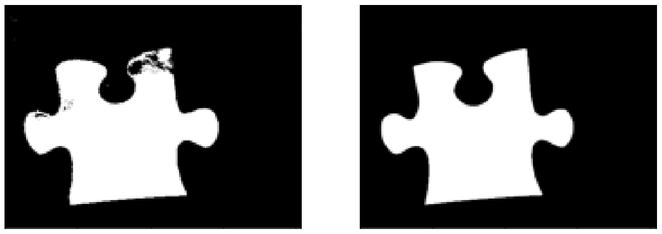

Fig. 18: RGB feature set, Puzzle piece 26, Predicted vs Ground Truth

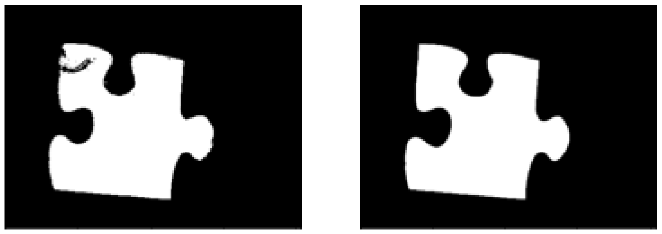

Fig. 19: RGB feature set, Puzzle piece 16, Predicted vs Ground Truth

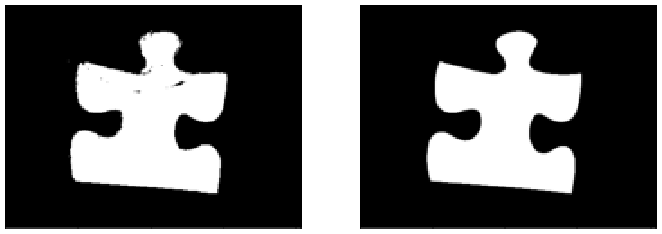

Fig. 20: RGB feature set, Puzzle piece 5, Predicted vs Ground Truth

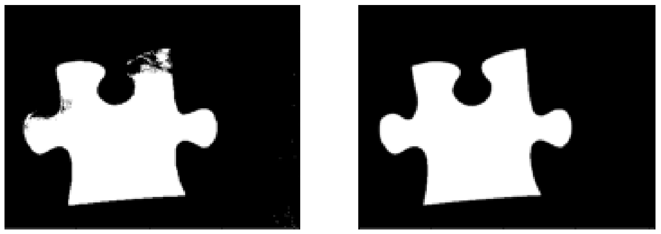

Fig. 21: RGB and DoG feature set, Puzzle piece 26, Predicted vs Ground Truth

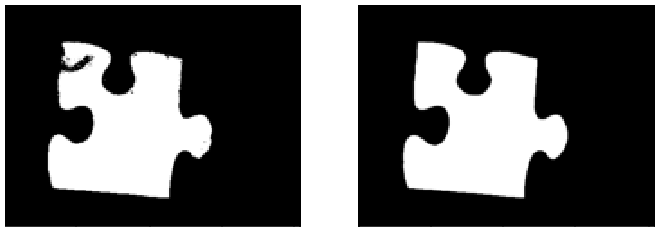

Fig. 22: RGB and DoG feature set, Puzzle piece 16, Predicted vs Ground Truth

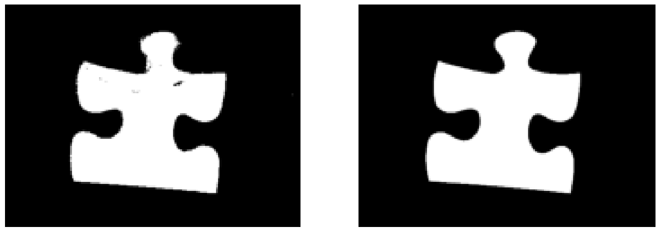

Fig. 23: RGB and DoG feature set, Puzzle piece 5, Predicted vs Ground Truth

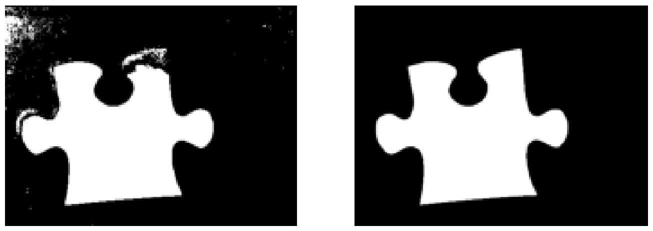

Fig. 24: Filtered feature set, Puzzle piece 26, Predicted vs Ground Truth

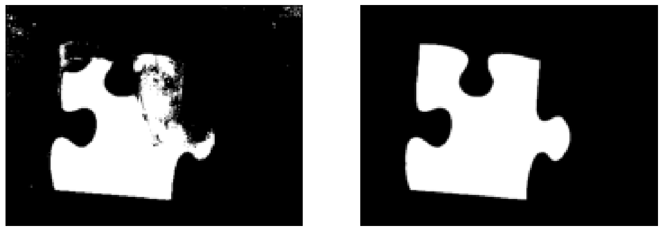

Fig. 25: Filtered feature set, Puzzle piece 16, Predicted vs Ground Truth

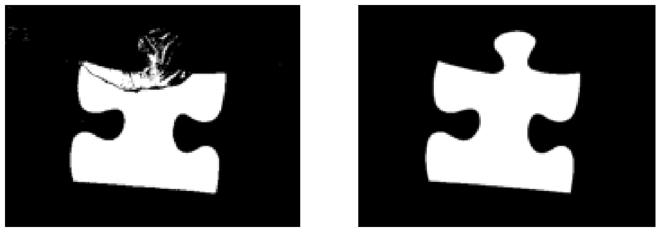

Fig. 26: Filtered feature set, Puzzle piece 5, Predicted vs Ground Truth