AutoML based Augmentation to improve Synthetic Data

Majd Al Aawar Dhruv Mehra Dheeraj Panneer Selvam

University of Southern California
Los Angeles, California, USA
January 21, 2023

Abstract

Synthetic data is commonly used in data-limited scenarios, but this data usually does not capture real-world conditions to their full extent. In order to address this issue, augmentation is often used in an effort to make data look more realistic. But the process of selecting and tuning for the optimal augmentations along with their best parameters can be difficult and tedious. This work aims to strengthen the capabilities of models to generalize in certain scenarios where it is not possible to get real world data. We propose an AutoML optimized model capable of segmenting real-world images in various climatic conditions by augmenting synthetic data to make it resemble the real world. The augmentation is optimized using feedback received from the segmentation output to increase the accuracy on real-world test data. Both Bayesian Optimization, through SMAC3, and Reinforcement Learning are tested for this feedback. We found that, given our limited training time, the hard coded optimized augmentation hyperparameters outperfromed all other methods and that our AutoML methods did outperform the baseline UNet model but likely still required more training for better convergence.

1 Introduction

Semantic segmentation has many applications in computer vision with its use cases ranging from medical imaging, depth estimation, to autonomous driving. There exists a lot of approaches in semantic segmentation which utilize deep learning models that tend to require a large corpus of data to train them [1, 6, 7, 13]. Gathering or acquiring this data is not always feasible due to privacy concerns, safety conditions, or a lack of available resources [20, 21].

A common way of addressing the above limitations is through the use of synthetic data, this data can be obtained through carefully constructed simulations, generated from AI algorithms, or even obtained from video games. The use of this type of data has shown to improve performance in data-limited settings [20, 21, 8]. To show its use case, an example of how synthetic data can be used in a safety setting could be through mimicking real-world radioactive data so that the personnel's exposure to radioactive material, through regular sensor maintenance or testing, can be minimized.

There are some drawbacks to utilizing synthetic data, one main disadvantage is that it might not accurately represent all the real-world conditions. For example in our particular use case, the GTAV dataset is collected from a video game which was not designed to account for some real world conditions which would take from the user experience, such as poor road conditions or motion blur from high speeds. In this work, we will attempt to address this shortcoming by augmenting the synthetic data to make it more realistic through the use of the Automold library. But one challenge in doing this is finding the optimal augmentations and tuning for their parameters in order to produce more realistic augmentations. Which is why we propose an AutoML based model which uses the validation error, validated on the real-world CityScapes dataset, obtained from a model trained on the augmented synthetic data to continuously tune and test for the best augmentation parameters using feedback until convergence is met.

2 Background

Unet [13] was initially designed for image segmentation tasks in the biomedical field for various purposes like cancer detection, brain tumor detection and so forth. Since then Unet has become very common in Deep learning segmentation tasks, usually requiring only slight modifications in accordance with the dataset in use. Previous works of Unet used on the CityScapes dataset[17], shows that it is able to produce good accuracy in this segmentation task. We will be using Unet as one of the baselines in our testing. Deeplabv1[3] is another commonly used model by others in the literature as it is similar in complexity to Unet but tends

to perform exceptionally better in some segmentation tasks. There are also many works in literature which use Deeplabv1 for segmentation of the CityScapes dataset.

Other works in literature attempt to utilize synthetic data for Unsupervised Domain Adaptation, which transfers what is learned from the source domains with a large amount of annotated training examples to target domains which have only unlabeled data. A few examples are the state of the art models HDRA[8] and CLUDA [20], while they do employ viable methods, One thing they assume is that we have a target domain to learn from which is not the case for us. In addition to this, they use a large amount of unlabeled data that in the target domain and do not discuss how well their methods perform in a data-limited scenario for the target domain.

Image augmentation is an important and commonly used technique in generating useful information, generating more data, and generalizing deep learning models. Segmentation tasks require a large number of pixel level annotation, corresponding to image size, which can be very time consuming and labor intensive. This is one reason why data augmentation methods have been widely used to generate artificial data. Some methods like spatial transformation, color distortion[18], and information dropping[16] have been used in simulating real world conditions and have been found to increase the robustness of the trained models. Spacial transformation and information dropping have been used in the literature to augment the CityScapes and GTA5 datasets[14] so that they could simulate different whether conditions or augment real world data using in-painting objects[9].

To make the use and application of AI more accessible, there has been significant interest in the field of AutoML in designing and training models. Applications of AutoML have been studied for computer vision problems in two separate areas: hyperparameter search[2] and architecture optimization[11] where hyperparameter optimization consists of training related parameters and architecture optimization consists of model related parameters. There are a few open source frameworks which provide users with the AutoML experience, one example would be the SMAC3 tool [10]. SMAC3 is used for Bayesian Optimization when optimizing hyperparameters, configuring algorithms, and solving global optimization problems. It was found to be on par with other state of the art Optimizers, and outperformed the baselines like Random Search. Another technique used for AutoML other than Bayesian Optimzation is Reinforcment Learning. The AutoRL [4] is used in the literature to address the problem of RL algorithims being sensitive to hyperparameter choices since they are non-stationary. They solve this by implementing a hyperparameter optimization population-based framework which meta-optimizes arbitrary off-policy RL algorithms while simultaneously training the agent. Their result show a significant improvement compared to the baseline RandomSearch and to be on par with other state of the art methods while being computationally more efficient.

3 Methodology

Figure 1 below shows us the overall design of our AutoML model's pipeline. We can see that it consists of four main components. The first components is the augmentation block, this block takes in the synthetic data as input and attempts to augment it, using the provided parameters, to look more realistic. The second component is the UNet model which is trained on the augmented data. The third component completes the feed-forward portion of the pipeline, and is the validation block. This block validates the trained UNet against our real world data and outputs an error score. The final component is the feedback which takes the feed-forward's output error and uses it to find the next set of optimized hyperparameters, through Bayesian Optimzation or Reinforcement Learning, to be fed back to our augmentation block for the next iteration. In the next subsections we will discuss each component in detail.

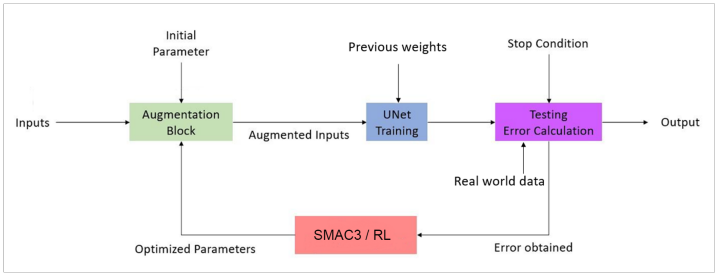

Figure 1: AutoML Model Pipeline

3.1 Data GTA5[12] data set is used as the synthetic data set which is used to train the model and CitySpaces[5] data set is used as the real world data set which is used to test and optimize the augmentation block. Common labels from both the data sets are aligned to the same classes and remaining labels are set

to an ignore class based on the label description from both data sets. Both data sets are first decoded to get the class labels from the color mask, then mapped to indicate the same classes and then resized to 455x256 resolution (maintaining the same aspect ratio). This resizing was done after running the some training tests using a single image on a laptop with i5 7th gen and a gtx 1060 6gb graphic shown in table 1.

Resolution	Time (s)
455x256	2.56
360p	3.83
480p	4.81
720p	9.60
1080p	45.90

Table 1: Computation times per tested resolution

As seen in table 1, our current resolution gives us the lowest time in computation and is a fairly good resolution for our task.

3.1.1 Synthetic Data - GTAV GTA5 is a synthetic data set which is collected from the popular game "Grand Theft Auto 5" which is set in an American style city. The data set contains 25000 screen capture in-game images and fine pixel level annotated labels. The original data set images of resolution (1914,1052) are resize and then the entire data set is used to train the model.

3.1.2 Real World Data - CityScapes CityScapes is a real-world data set which is focused on semantic segmentation task in urban city street scenes. The images are collected from 50 different European cities and contains 5000 fine pixel level annotated images and labels. The data set images are originally of resolution (2048,1024) are resized and only the train and val split are used to test the model, since the test set does not have labels. The 2975 labeled training data is used as our real-world validations set and the 500 labeled validation set is used as a final testing set to evaluate the effectiveness of our methods

3.2 Augmentation Block The augmentation block makes use of the Automold–Road-Augmentation-Library [19] to make our synthetic dataset look more realistic. This library is designed to augment road images to have various weather and road conditions, the augmentations it provides are: brighten, darken, add shadows, add snow, add rain, add fog, add gravel, add sun flare, add speed, add autumn, flip image, add manhole, and add gravel. For the purpose of our project and to save computational resources, we had only considered the brighten, darken, snow, random flip, and rain augmentations, since the rest have heavier inference, as seen in table 2 below, or not as meaningful and finding all the different combinations would be too demanding computationally. But now as we are reducing the size further we can accommodate more augmentations.

Augmentation	Time (s)
rain	0.02
fog	0.37
gravel	0.04
snow	0.06
flip	0
shadow	0.01
autumn	0.31
manhole	0.01
brightness	0.05
speed	0.2
Sun flare	0.09

Table 2: The above inference times are computed on a single 1280x720 image.

The augmentation block is designed to have four inputs:

1. images: List of images from our dataset
2. labels: List of corresponding labels to the images. These will also need to be modified in some cases (such as when flipping our image).
3. aug: list of distribution of augmentation types, for example [0.025, 0.05] would augment 2.5% of images with the first augmentation type and 5% of the images with the second augmentation type (no overlap). These will be tunable parameters for our AutoML pipeline to optimize.
4. params: List of parameter coefficients for each augmentation type (each is between 0 and 1). By default, params = [0.5,0.5,0.5,None,None]. This sets None to the random_flip and add_rain augmentation's parameters so that the default values are activated. This is because these values are randomly assigned by the function and should vary, (ex: rain droplet's length or slant). These will also be tunable parameters for our AutoML pipeline to optimize (except for the None types being fixed)

We can see an example of the output from the augmentation block in figure 2 below with the following input: 20 images, augmentation distributions of 10% allocated to each type, and using the default parameters for each augmentation.

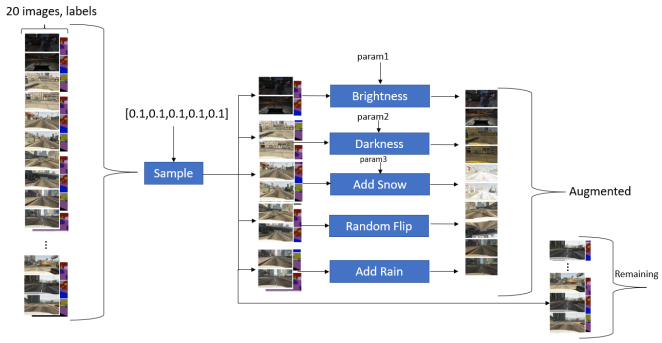

Figure 2: AugBlock's Anatomy Example

From the output we see that 50% of the 20 images (10 images) are augmented and are shown below in figure 3 in the same order as shown above (2 pictures per augmentation).

Figure 3: AugBlock example output

3.3 UNet UNet bases its architecture upon a combination of CNNs with autoencoder (AE) architecture. AEs are neural networks that aim to find a latent space that represents the data in a smaller space as shown in the left portion of figure 4 below. UNet uses the autoencoder architecture for its encoder and decoder parts. A defining feature of UNet is the U shape connection between this encoder-decoder as seen in right figure 4.

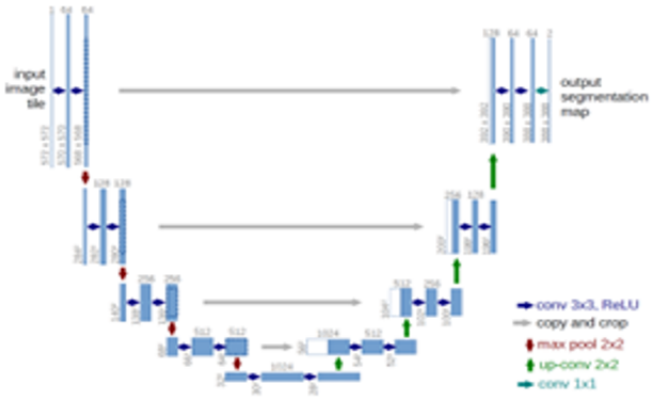

Figure 4: UNet Stucture

The UNet architecture we are using contains 4 encoder layers and 4 decoder layers with respective skip connections, each encoder layer is made up of 2 convolution, batchnorm and ReLu layers followed by a max-pool layer and decoder layers are constructed using a convolution transpose layer followed by a series of 2 convolution, batchnorm and ReLu layers.We have chosen this type of network because of restricted resources and time constraints, this network can be swapped for a more complex network like a deeplab v3, for better accuracies, but would cost more computing resources and time.

3.4 Feedforward The feedforward network consists of the augmentation block and the Unet, later it is evaluated on real data. The input images (synthetic data) are loaded using custom data loaders, which have been created to match the labels and color codings of the real world data (cityscapes) and synthetic data (GTA 5).

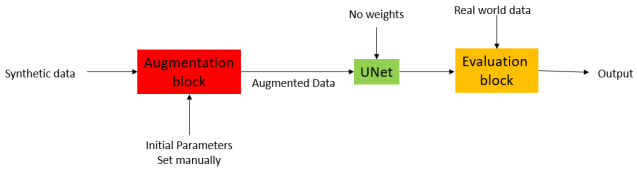

Figure 5: Feedforward Pipeline

The input synthetic data is then loaded into the augmentation block with some initial parameters (not tuned). The augmentation block performs various augmentations based on the parameters set and gives out a new augmented data loader. The augmented data loader is passed into the UNet for training and the model is saved. With the trained weights, the model predicts the output with the real world (GTA 5) data and an accuracy measure is used to compare the predictions and labels.

3.5 SMAC3 Feedback As mentioned above, Sequential Model Algorithm Configuration (SMAC) is an open-source package which is commonly used in the hyperparameter optimization of Machine Learning algorithms, or even arbitrary algorithms. It aims to optimize the hyperparameters by finding the best set which minimizes a validation dataset's loss as shown below:

$$(A,\ \boldsymbol{\lambda}^*) \in \underset{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}_i}{\arg\min}\ c(A,\ \boldsymbol{\lambda}) = \underset{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}_i}{\arg\min}\ \mathcal{L}(\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}; A\ (\boldsymbol{\lambda}))$$

SMAC3 [10] starts by doing a random sampling following the Sobol Sequencing, it then begins training a RandomForest surrogate function which estimates how well the model can perform if trained with a certain set of hyperparameters, and finally it select new hyperparameters based on the trained surrogate function and selection function, which is a logEI (expected improvement), function. It does this iteratively until our hyperparameter choices near convergence.

By simply setting up a train() function (which is our feedforward portion of the pipeline) that outputs the error, we define the ConfigurationSpace (which takes a dictionary of the parameters we want to tune and their limits) and create a SMAC object for hyperparameter optimization using the provided configuration space and the settings we desire. This outputs the optimal values for our hyperparameters and their corresponding validation scores.

3.6 RL Feedback In this section we will discuss our feedback experimentation through RL. This was not presented initially as it was not yet finalized due to it make the overall pipeline computationally heavier.

3.6.1 Action and State Space The Action Space was a Gym Box, with 8 values ranging from 0 to 1. Gym is an open source library commonly used in reinforcement learning that provides APIs which allow for communication between learning algorithms and environments. A Gym Box is a datatype used to define observation and action spaces.The first 5 actions were

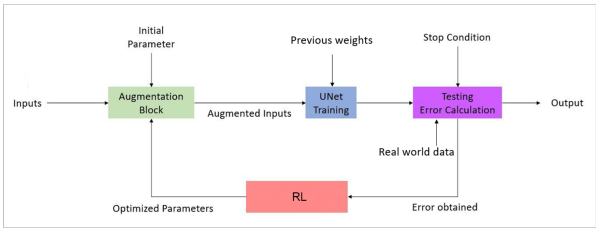

Figure 6: RL Feedback

the augmentation percentage and the last 3 were intensities, each action corresponds to the hyperparameters we are attempting to tune as seen in figure 7.

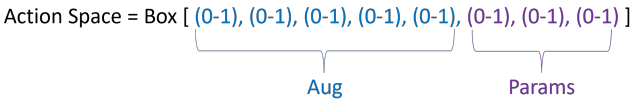

Figure 7: Action Space

The State space should technically be composed of the entire image validation dataset, which is 2975 images with dimensions 455*256. But this huge state space is not feasible given our resources. So, we produced a measure which will be used as surrogate for this by discretizing each batch of images.

This was done by considering the number of unique labels over each batch for the predictions and ground truth, we checked if all the unique labels present in the predictions was also present in ground truth and flagged them as 0 or 1. If labels were not present on both we set the values to 2. Since we use a batch size of 16 for our training, in total we will have a 19x186 2D MultiDiscrete State space, 19 corresponds to the number of classes we have (where each value corresponds to a correct/incorrect/missing discretization discussed above) as shown in figure 8 and 186 is the number of state spaces we end up with, corresponding to 2975/16. This significantly reduces the size of our state space, making it much better than that of 2975x455x256.

3.6.2 RL Feedback Environment For each step the actions are taken, which is the augmentation block's hyperparameters, and the model is trained with a given set of epochs using the augmented data. The model is then validated on the real-world data which outputs the score and the state as mentioned before. This score obtained is our reward and our RL model tries to

State Space = MultiDiscrete [[0 , 1 , 1 , 0, ...]
.
.
[....., 0, 1 ,2]]

Figure 8: State Space

maximize it using the state information available. As shown in figure 9

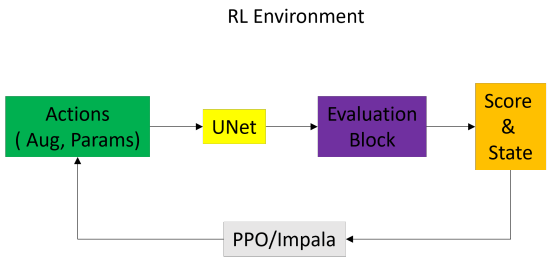

Figure 9: RL Environment

For our RL model, proximal policy optimization (PPO) was used as a policy gradient method [15]. This method alternates between optimizing our surrogate objective function, mentioned previously, using stochastic gradient ascent and sampling data by interacting with the environment. This objective function is updated visa multiple epochs of minibatches. This was implemented using the Gym and StableBaselines3 library with the policy set to an MlpPolicy, which implements actor critic, using a a multilayer perceptron with 2 layers of 64 neurons each.

4 Experiments

We conducted a series of experiments to properly evaluate our results. All the experimentation's UNet models were trained using the default recommended hyperparameters and for 10 epochs each since we were limited by computational resources and time. All final results shown are pixel to pixel accuracies (hit or miss) when the methods are tested against the real-world CityScapes dataset's validation set of 500 images (our test set).

4.1 Baseline: UNet First we used a baseline UNet model, described in Section 3.3, which is simply trained on the synthetic data and then tested on the real-world data. Figure 10 shows a few output predictions and their corresponding ground truth labels on randomly drawn samples.

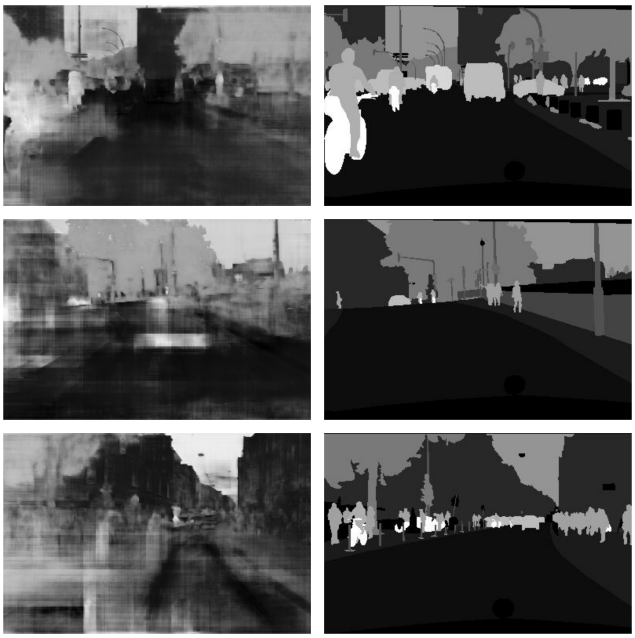

Figure 10: Baseline UNet output (Left), label (Right)

The accuracies calculated on each image are shown in table 3 below

img1	0.164
img2	0.1845
img3	0.2014

Table 3: Baseline UNet sample output accuracies

4.2 AutoML Feedback: SMAC3 This section shows the results from our first implementation of the AutoML model pipeline utilizing SMAC3 for Bayesian Hyperparameter Optimization. Figure 11 shows a few output predictions and their corresponding ground truth labels on randomly drawn samples.

The accuracies calculated on each image are shown in table 4 below

img1	0.2968
img2	0.3180
img3	0.3429

Table 4: AutoML-SMAC3 sample output accuracies

We can also see the validation loss during the training in figure 12 and that it does in fact go towards

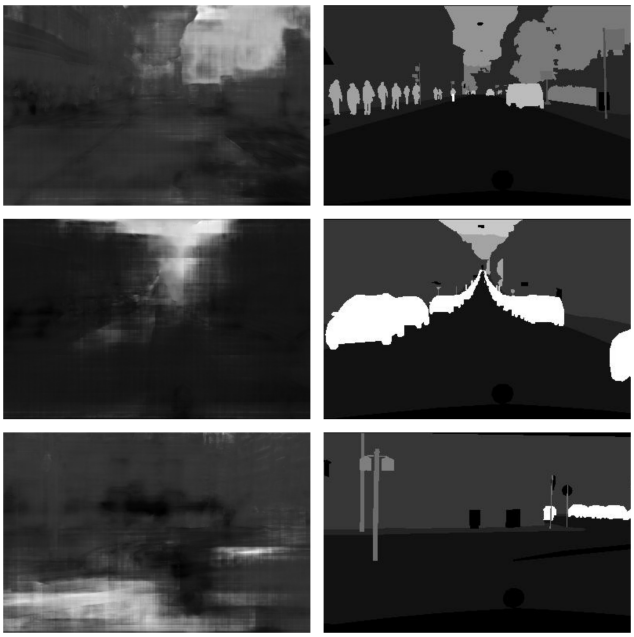

Figure 11: AutoML-SMAC3 (Left), label (Right)

conversion (but likely doesn't reach there).

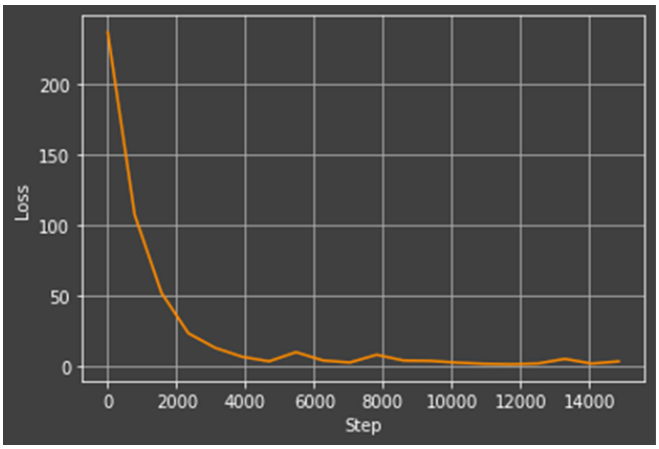

Figure 12: Validation loss during training

4.3 AutoML Feedback: RL This section shwos the result of our second implementation of the AutoML model pipeline which utlizes RL for hyperparameter optimzation rather than smac3's Bayesian Pptimization. Figure 13 shows a few output predictions and their corresponding ground truth labels on randomly drawn samples.

The accuracies calculated on each image are shown in table 6 below

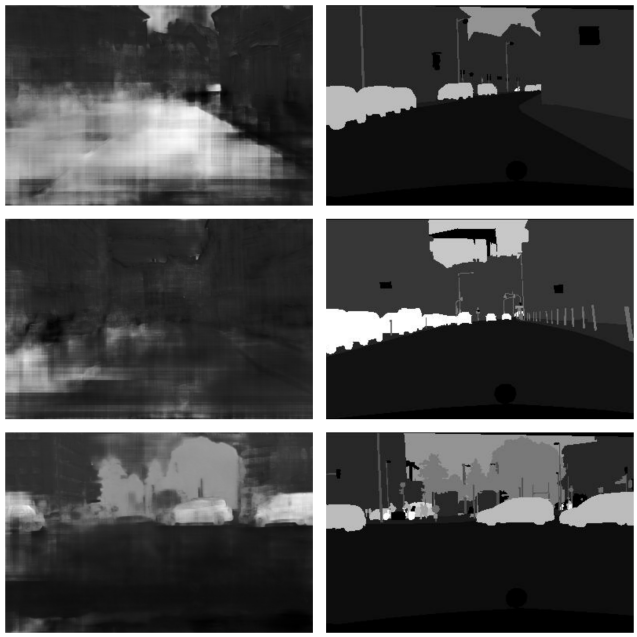

Figure 13: AutoML-RL (Left), label (Right)

img1	0.3334
img2	0.3892
img3	0.3949

Table 5: AutoML-RL sample output accuracies

4.4 Hard Coded Augmentation The final experiment involved taking SMAC3's output optimal hyperparameters and hardcoding that into our augmentation block and simply training our baseline UNet with feedback on this augmented data. Figure 14 shows a few output predictions and their corresponding ground truth labels on randomly drawn samples.

The accuracies calculated on each image are shown in table 6 below

img1	0.4048
img2	0.4371
img3	0.4198

Table 6: Hard coding augmentation sample output accuracies

4.5 Comparisons In this section, we compare all the above experiments' results when tested on the whole 500 images. The final results in table 7 are the averages accuracies over the 500 images:

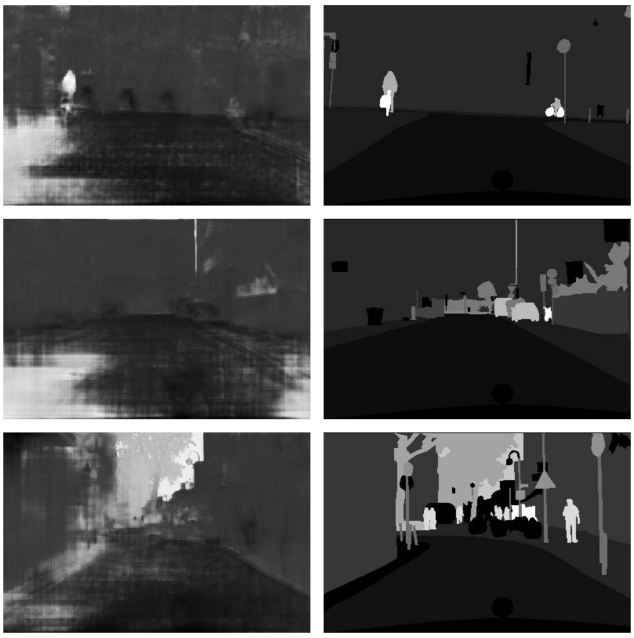

Figure 14: Hard coded augmentatiom (Left), label (Right)

Method	Base	SMAC3	RL	**Aug**
Avg_Accuracy	0.1986	0.225	0.242	**0.292**

Table 7: Hard coding augmentation sample output accuracies

These results show that the optimized hyperpameters hard coded into the augmentation block gave the best performance and that both AutoML implenntations were able to outperform the baseline UNet model.

5 Discussion

From the results in the previous sections, we can see that all our experimentations did yield better results than the baseline model. We can also see that training our AutoML model with both SMAC3 and RL both outperformed the baselines given that these techniques were only trained on a few iterations compared to what is required. Fruthermore, we can see that RL is likely the better method for two reasons, the first being that we do not have to treat it as a blackbox model and can visualize how the optimzation is happening and the second is apparent from the results. Taking the optimally generated hyperparameters and hard coding these into our augmentation gave the best performance, this is indicative that the hyperparameters being tuned are converging towards values which make the images more realistic. While hard coding the optimized hyperparameters did have the best performance, some more testing would be needed against multiple iterations of randomly selected hyperparameters to confirm that the hyperparameters are actually converging towards more optimal values, as we were limited in time we could not do this.

6 Milestones

Below we can see the list of milestones we have accomplished in this project with their corresponding timelines.

- Download datasets and align them to eachother. Nov1-Nov14
- Setup necessary environments (Github, anaconda, etc..). Nov1-Nov14
- Create augmentation block. Nov1-Nov14.
- Setup UNet architecture. Nov1-Nov14.
- Integrate components into feedforward pipeline. Nov14-Nov20.
- Integrate all components into final MLOps pipeline with AutoML. Nov 20-Dec7
- Setup & integrate SMAC3 for feedback. Nov 22-Nov30.
- Setup & integrate RL for feedback. Dec1-Dec10.
- Run multiple test iterations and expirmentations of overall pipeline. Dec1-Dec10

7 Challenges

There were a few challenges involved in this project. The first being poor documentation of some of the packages we used and the second being implementing the various components into a single pipeline. But the main problem we faced were that of computation time and resource limitation. As a result, we had to do some tradeoffs:

- Simple UNet was used instead of complex segmentation networks
- Resolution was reduced by a factor 20.
- Iterations of training were lowered.
- Types of Augmentation were restricted.
- Utilized surrogate state-spaces rather than the full state-space

8 Future Work

While we did get some interesting results and were able to get some insights on this technique, this work could be extended further by:

- Experimentation with better complex segmentation networks.
- Use of different loss functions in model training.
- Use of different evaluation metrics for feedback network.
- Using a bigger set of augmentation parameters.
- Implementation of other types of feedback like RL based, Gradient based.
- Train for number of Epochs.
- Interpreting what is happening in the RL block in every epoch
- More testing and experimintation to confirm some results
- Testing this approach on other synthetic datasets

9 Conclusions

In conclusion, we have successfully developed different implementations of our AutoML model which are both outputting augmentation hyperparameters, which are converging towards optimal values, that will augment the synthetic data in such a way that it better represents the distribution of the real world data. The accuracies obtained provide proof for even further improvements. This could potentially solve scarcity of real world data by substituting with synthetic data. Given better resources, it would be interesting to further study the impact on performance of training for longer epochs and also checking whether this is a viable method by testing it against other datasets with different kinds of augmentations.

10 Contributions

- Dhruv Mehra: Prepared data
- Majd Al Aawar: Prepared augmentation block, integrated SMAC3 feedback, aided in feedforward pipeline.
- Dheeraj Panneer Selvam: Setup UNet, integrated feedforward components, setup RL feedback

References

[1] V. Badrinarayanan, A. Handa, and R. Cipolla, *Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling*, CoRR, abs/1505.07293 (2015).

[2] J. Bergstra, D. Yamins, and D. D. Cox, *Making a science of model search*, (2012).

[3] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, *Semantic image segmentation with deep convolutional nets and fully connected CRFs*, (2014).

[4] J. K. Franke, G. Koehler, A. Biedenkapp, and F. Hutter, *Sample-efficient automated deep reinforcement learning*, in International Conference on Learning Representations, 2021.

[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*, (2017).

[6] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, *Mask R-CNN*, CoRR, abs/1703.06870 (2017).

[7] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, CoRR, abs/1512.03385 (2015).

[8] L. Hoyer, D. Dai, and L. Van Gool, *HRDA: Context-aware high-resolution domain-adaptive semantic segmentation*, (2022).

[9] D. Lee, S. Liu, J. Gu, M.-Y. Liu, M.-H. Yang, and J. Kautz, *Context-aware synthesis and placement of object instances*, (2018).

[10] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter, *Smac3: A versatile bayesian optimization package for hyperparameter optimization*, Journal of Machine Learning Research, 23 (2022), pp. 1–9.

[11] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. Yuille, and L. Fei-Fei, *Auto-DeepLab: Hierarchical neural architecture search for semantic image segmentation*, (2019).

[12] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, *Playing for data: Ground truth from computer games*, (2016).

[13] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, CoRR, abs/1505.04597 (2015).

[14] C. Sakaridis, D. Dai, and L. Van Gool, *Semantic foggy scene understanding with synthetic data*, (2017).

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, CoRR, abs/1707.06347 (2017).

[16] K. K. Singh, H. Yu, A. Sarmasi, G. Pradeep, and Y. J. Lee, *Hide-and-seek: A data augmentation technique for weakly-supervised localization and beyond*, (2018).

[17] Srihari-Humbarwadi, *Srihari-humbarwadi/cityscapes-segmentation-with-unet: Multiclass image segmentation in keras.*

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed,

D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going deeper with convolutions*, in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, June 2015.

[19] R. G. Ujjwal Saxena, *Ujjwalsaxena / automold–road-augmentation-library*.

[20] M. Vayyat, J. Kasi, A. Bhattacharya, S. Ahmed, and R. Tallamraju, *CLUDA : Contrastive learning in unsupervised domain adaptation for semantic segmentation*, (2022).

[21] B. Xie, S. Li, M. Li, C. H. Liu, G. Huang, and G. Wang, *SePiCo: Semantic-Guided pixel contrast for domain adaptive semantic segmentation*, (2022).