Semantic Segmentation: CNN vs Random Forest

Shree Murthy

2024-02-19

Table of Contents

1. Introduction

Semantic segmentation involves classifying every pixel in an image to a specific class. For this report, I used the `CamSeq07` dataset which contains 202 images and masks to learn more about semantic segmentation. The goal of this project is to analyze how well a CNN and Random Forest model can perform on semantic segmentation. The CNN model was trained using the `U-Net` architecture and the Random Forest model was trained using the `RandomForestClassifier` from the `sklearn` library. Before I created the CNN architecture, I used the Random Forest model to decide if deep learning was required.

1.1 Random Forest Analysis

For the Random Forest (RF) model, I created a 70/30 train-test split and trained the model. Ever image and mask was paired together and pixels were flattened to be fed into the model. The model was trained on two estimators and a max_depth of three. When I initally started the training process, I set the estimators to ten and had no max_depth. This took the model 15 hours to train. I reduced the estimators drastically and added a max_depth to reduce the training time. This reduced the training time to 2.77 hours (9880 seconds). To me this was unaccepatable even though it makes sense. An RF model has data fed in one at a time and every pixel is an independent feature. However, the model's accuracy and IoU scores were good:

Train/Test	Accuracy	IoU Score
Train Data	0.8576	0.9172
Test Data	0.8295	0.9140

For both datasets, the IoU scores were above 0.9. This was a good sign that the model was able to classify the pixels well. While the accuracy is lower, those numbers are just provided to show that the model is not overfitting. The IoU score is the most important metric for semantic segmentation and that score provides high pixel-wise overlap. Below is an example output of the model:

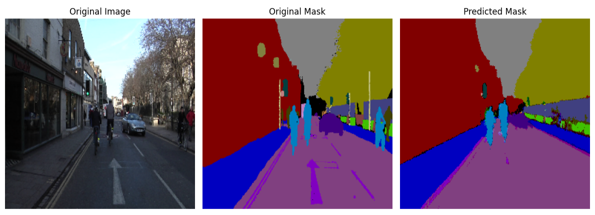


Figure 1: Random Forest Example

The predicted mask shows that the model was able to identify every section of the image. However, the segmentations aren't defined. For instance, the car is not segmented properly to showcase the entire shape of the car and the segmentation is not clear. The entire predicted mask has noise and is not smooth compared to the actual mask. This is due to the pixels being fed in without spatial relationships.

Coupling the training time with the output mask analysis, using an RF model was not sufficient. I would rather use a CNN model to train faster to leverage faster analysis and better spatial understanding of the image. It is not worth using a model that takes 3 hours to train on a GPU and produces noisy masks that don't showcase smooth segmentation of features.

2. Exploratory Data Analysis

There are 202 images and masks (101 each). Each image and mask are 720x960x3. The images with '(name)_L.png' are the masks. Furthermore, there is a `label_colors.txt` file that contains all the color codes for the various segments. There are 32 classes within the dataset: such as, buildings, trees, bicyclists, and etc. Since the `CamSeq07` dataset is derived from a video, the images are not independent of each other. To ensure no overfitting occurs, I will have to shuffle the pairs before splitting them. Below is a sample image and mask:

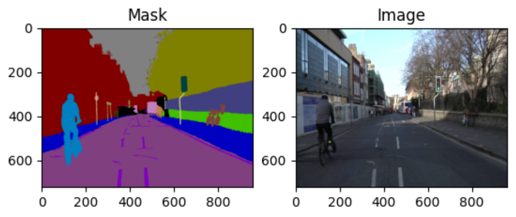


Figure 2: Sample Pair

The masks are smooth and segments everything visible within the original image. This is good to see because this will be needed to ensure the model is trained properly on all the visible classes. Overall, the images will have to be resized to ensure model is not overfitting and can train faster. Also, the masks will have to be encoded with the proper classes such that we can `argmax` the output to get the predicted mask.

3. Methods

3.1 Data Preprocessing

For both the CNN and RF the preprocessing steps for the images are the same. The images are resized to 256x256x3. For the masks, the two models differ. The RF model doesn't encode the masks because doing so creates a 256x256x32 (32 classes) array. This was too large to fit into memory and training shot up to 8 hours before killing the process. Thus, the masks for the RF were resized to 256x256x3. The CNN model was efficiently encoded and resized to be 256x256x32. Both models shuffled the masks and image pairs to avoid overfitting. The CNN model was split into 65/15/20 train-validation-test split. The RF model was split into a 70/30 train-test split.

Augmentation was not used for the RF model aside from zoom to fit the desired size requirements. The CNN model did not use other augmentations; I used zoom to resize the images/masks. My reasoning is that conducting other augmentations could ruin the mask encodings and the augmentations may not be consistent (i.e the same flip may not be applied to the image and mask). Finally, all the data was stored in a `.npz` file to keep data consistent and easily loaded into other files.

3.2 Model Creation

For the RF model that model was created using the `RandomForestClassifier` from the `sklearn` library. The model was trained on 2 estimators and a max_depth of 3.

For the CNN model, I leveraged the `U-Net` architecture learned in class. The model was trained using a VGG16 encoder and a decoder I built from scratch. The encoder model created the feature maps and the decoder model upsampled and used skip connections to build the predicted mask. The output mask was 256x256x32. The model was trained using `Adam` optimizer and `categorical_dice_loss` as the loss function. At every epoch, the model calcuates accuracy, dice score, and Jaccard (IoU) score. The model was trained for 75 epochs with a batch size of 32.

The CNN model used two callbacks: ModelCheckpoint and ReduceLROnPlateau. The ModelCheckpoint saved the best model weights and the ReduceLROnPlateau reduced the learning rate if the model's validation loss did not improve after 5 epochs. This ensured I was getting the best possible weights and model performance.

4. Results

The CNN model had the following results and history plot:

Train/Validation/Test	Loss	Accuracy	Dice Score	IoU Score
Train Data	0.0832	0.9186	0.9121	0.8390
Validation Data	0.0815	0.9236	0.9185	0.8500
Test Data	0.086	0.919	0.914	0.842

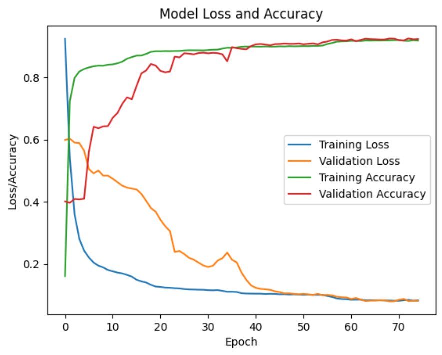
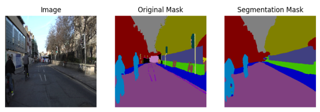
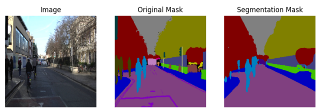
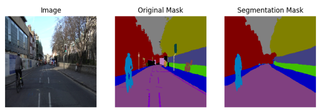
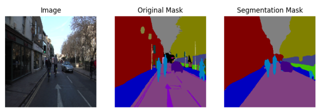
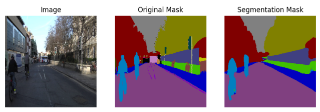
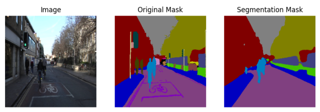


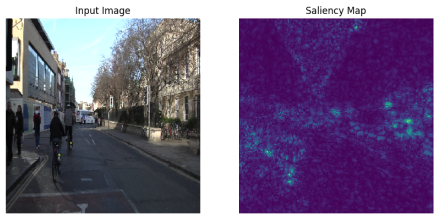
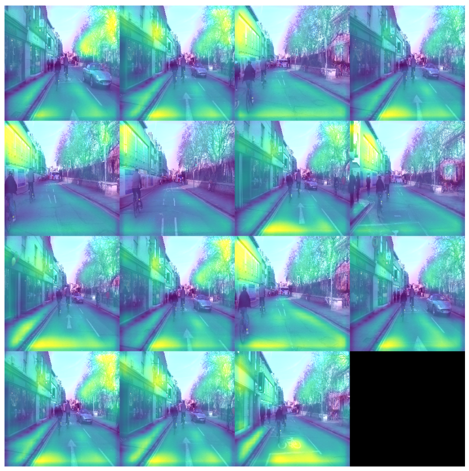
Figure 3: CNN History

Loss and accuracy for train and val improved over time and had no overfitting. The history plot proves that the model converged properly and there were no issues during the training stage. Furthermore, the accuracy scores were better than the RF model proving that the CNN was able to produce better masks. However, the key indicators were the Dice and IoU score. However, the difference between the dice and IoU scores was concerning. Both metrics are used to evaluate overlaps between original and predicted masks. The high dice score indcated the entire original and predicted mask had high similarity. However, the IoU score indicated that the pixel-wise overlap wasn't as high.

To analyze the best and worst image outputs, I created a table to hold the 3 best and 3 worst images along with their corresponding masks. The table provides insights into the model's performance on specific images.

Best Images	Worst Images
	
	
	

Each image in the table is ordered as follows: original image, original mask, predicted mask. The best and worst images all had one thing in common, it wasn't as detailed as the original masks. This could be attributed to the model focusing on the larger, profound, segments and not the smaller, intricate, segments. This would've led to a higher IoU score to match the high dice score. To dive deeper I looked at a saliency and GradCAM map to see what the model was focusing on.

Saliency Map	GradCAM Map
	

The Saliency map and GradCAM map proved the assumptions I made from the 3 best and worst images. The model was focusing on the larger segments and not the smaller segments. The two maps indicate that the model focused on the large profound objects on the sides of the road and then worked its way to the center where the bicyclists are located and where smaller segments can be found.

Thus, the model was not able to segment the images properly.Capturing dice and IoU scores helped flesh out why the predicted masks might have high similarity to the original masks (indicated by the high Dice scores), but not high pixel-wise overlap because smaller segments were not captured (indicated by the lower IoU score). Nevertheless, compared to the RF model, the CNN model had no noise and segmented the key objects (like car, building, and etc) much better than the RF model.

5. Discussion

Comparing the two models, I would state that the CNN model performed better than the RF model. Even though the RF model created more pronounced and detailed segmentations, the model took a long time to train and noise was introduced into the RF model. The CNN model trained faster and produced smoother masks. However, the CNN model was not able to capture the smaller segments and the pixel-wise overlap was not high. I would recommend using the CNN model over the RF model because the CNN model was faster and produced smoother masks. From this assignment I learned that using classic models like RF for semantic segmentation is not efficient and deep learning models are better suited for this task. Classic models are not worth using because of the high training time. If were to conduct this assignment again but with a larger dataset, I would've wasted more than ~3 hours to train the RF model which is not ideal if there are mission specific deadlines.

5.1 Hyperparameter Tuning and Future Considerations

In terms of hyperparameter tuning for the CNN model, I conducted various approaches before settling on the hyperparameters mentioned in the report. I tried to use the off-shelf Adam optimizer, but it didn't make a good impact on the steps taken to achieve best loss. During the training I noted 5 straight epochs that didn't have an improved val_loss (val_loss worsened during these 5 epochs). Thus I added ReduceLRonPlateau to reduce the learning rate if the val_loss didn't improve after 5 epochs. Next, I tried to use no regularization techniques. This led to overfitting (Train Accuracy was 94.5% but Val Accuracy was 84.7%). To combat this, I added L2 regularizers to every Conv2D layer in the decoder and added BatchNorm before and after every Conv2D layer. This made the model even worse (Train Accuracy was 92% and Val Accuracy was 80%). Finally, I achieved the best results by removing the L2 regularizers and left the BatchNorm layers. This led to the current results seen in this report. I did try unfreezing the top two layers of the model as well as all the layers. This led to the model not converging within the 75 epochs and led to low metrics across the board. I also monitored different epoch values. I ended up with 75 epochs because the model kept showing improvements after the first 20, 30, and 50 epochs I tested. After 75 epochs the model started to worsen in all categories; for example, loss increased from 0.08 to 0.20 for the train set after I tested the model with 100 epochs.

Before I conducted the hyperparameter tuning, I created two different U-Net architectures using two different transfer learning encoders: VGG16 and MobileNetV2. Both models used the same structure for the decoder: four sections to upsample the feature maps. These models were trained as simple as possible to assess the raw impact they had in analyzing the dataset. The VGG16 model greatly outperformed the MobileNetV2 model. VGG16 was 5% more accurate and loss was 0.12 points better than the MobileNetV2 model. This led me to use the VGG16 model for the final CNN model. I did not use dice coefficient and Jaccard score during this comparison process; however, I used the categorical_dice_loss and accuracy to priortize similarity between the overall masks instead of pixel-wise overlap. This was to reduce computational resources and time to find the best model; hence, I focused on the key metrics to provide overall mask similarity.

For the loss functions, I could've used Jaccard Loss instead of Categorical Dice Loss. Jaccard Loss is a better metric to use for semantic segmentation because it focuses on the pixel-wise overlap. Categorical Dice Loss is a good metric to use for overall mask similarity. I wanted to prioritize similarity between the masks; hence, I used Categorical Dice Loss, and monitored Jaccard score as a metric.

In the future, I want to research augmentation approaches that preserve key mask information to help improve the pixel-wise overlap. During the assignment timeframe, I didn't find many approaches; moreover, I didn't want the models to falter drastically because I didn't know how to keep the mask and image augmentations consistent. Also, a Vision Transformer may be better suited for semantic segmentation because of the attention layers. This could help the model focus on the smaller segments after the attention layers focused on the larger segments. If I continue developing the CNN model created in this report, I would experiment adding more layers to the decoder to see if the model can capture the smaller segments and improve the Jaccard score. For the sake of efficiency and minimal computational resources I didn't pursue this approach for this assignment.

6. Resources

1. Learning Rate Scheduler
2. U-Net Architecture
3. tf-explain
4. Multi-Class Segmentation
5. Categorical Dice Loss