Overview

The title of our project is ‘Road Extraction and Analysis using Preprocessing Postprocessing and Deep Learning techniques’. This project is helpful in various ways to update the maps or to use such data in emergencies such as disaster planning, any military emergency, evacuation planning, or resource allocation. If any new roads are built it will also help to detect those. Considering the state of the art for this project, there have already been numerous studies done on road extraction using satellite images, like ResNet, U-Net, DeepLabV3+, DenseNet, and VGG. Also, numerous image preprocessing being explored which enhanced the way images were trained and also used post-processing methods to enhance the images more and make them more readable to the user.

In the project we have implemented multiple approaches to reach the desired output it is divided into three steps, we have preprocessed the data with existing technologies and implemented a solution by using preprocessing methods such as Data Augmentation, Data Segmentation, and Image Enhancement on the dataset. Secondly, have implemented different CNN models. We have implemented model fusion of Resnet50 and VGG a pretrained deep learning models, where we have concatenated the features extracted from both the models and predicted the output on the test image dataset, and explored 3 different preprocessing while implementing the model for each preprocessing technique, where we can analyze models with different preprocessing based on the predictions they provide. The last step includes the Post-processing techniques we have applied to the predicted images. In this project, we have implemented Edge Detection, Morphological operations, and Conditional Random Fields as our post-processing techniques and explored how they work on the predicted images. For the evaluation purpose since masking images on testing data was not there, we did a qualitative evaluation of the project in addition to that We also plotted loss and accuracy for the training and validation dataset.

To summarize the work, We have implemented 3 preprocessing techniques, Data Augmentation and Image enhancement performed by Bhushan, Image Segmentation done by Deep, 1 fusion model(ResNet and Vgg) by Bhushan and Deep,1 U-Net Model by Deep, and 3 post-processing techniques in Edge detection, Morphologic operations performed by Deep and Conditional Random Field by Bhushan.

Approach

The approach we have implemented for road extraction consisted of important three steps. At first, we preprocessed the training images which consisted of satellite images and the corresponding mask of that image, using three different preprocessing techniques, which are Image Augmentation, Image Enhancement, and Image Segmentation. Then, we used the two different models to train on each preprocessed technique. The models we used are a model fusion of ResNet50 and Vgg16, where we extracted the features from both the transfer deep learning model and concatenated their extracted features. The third important step includes the use of post-processing techniques on predictions of test images, where we have used three different post-processing techniques, which are Edge Detection, Morphological Operations, and Conditional random field. For a better understanding of the flow of approach, refer below diagram. After the image is predicted as an output from the model to convert probabilities provided by the sigmoid function we apply a thresholding of 0.1.

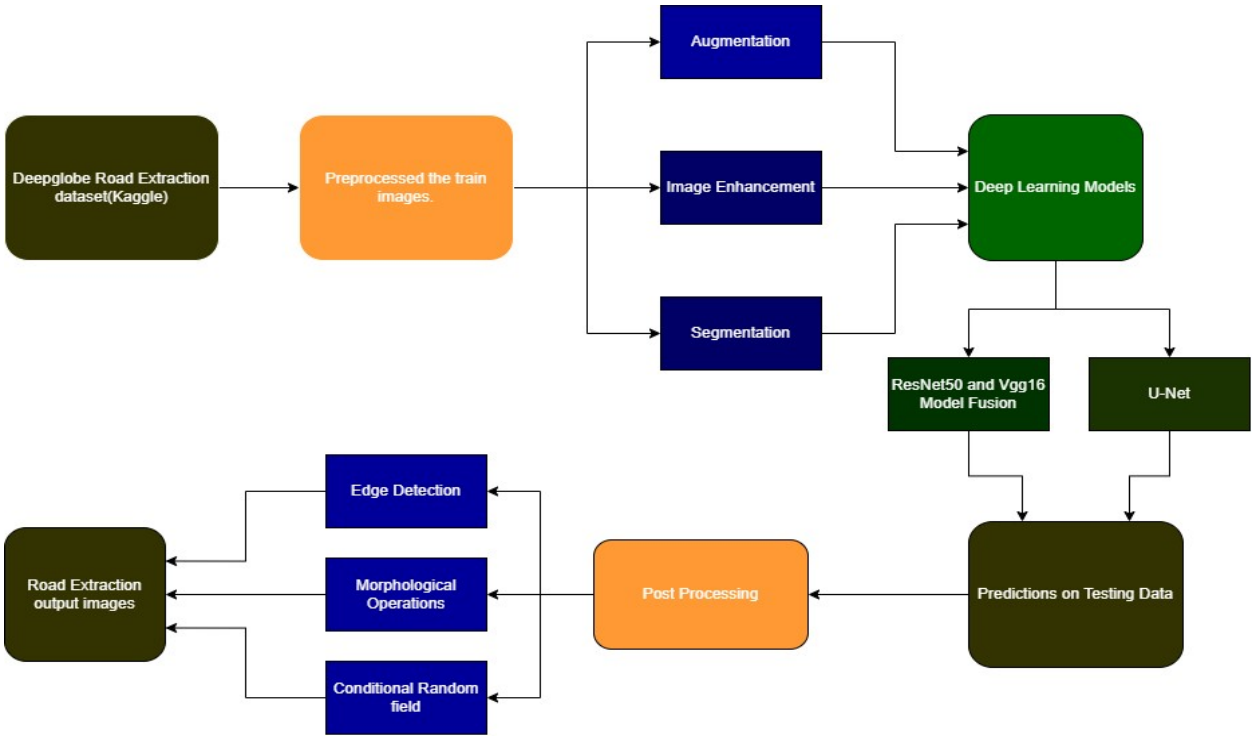


Fig: Block Diagram

We wanted to explore how different preprocessing and post-processing affect the output or predictions of images from models. Also, while reading some research papers we came across the U-net model as an efficient model to train on road extraction problems. The idea behind implementing model fusion of ResNet50 and Vgg16 was to explore how multimodal works on a specific task. ResNet architectures are known for deep architectures with residual connections, which helps in learning from a vast amount of data without overfitting. Whereas, Vgg16 works better in capturing texture and fine details. The fusion of models makes use of the strength of both models and potentially leads to better performance. Initially, we

thought to implement model fusion in such a way that we extract features from ResNet50 and provide the extracted features as input to the Vgg16 model. Then we completed the fusion process by concatenating the features extracted from both models and then training accordingly.

Referred online resources for concatenating the features of pre-trained models fusion, also referred to Kaggle notebook for defining U-net model. Independently coded sections include, augmentation using ImageDataGenerator, training the model, and testing the model, For other parts, have referred to online resources such as geeks for geeks, StackOverflow, Analytics Vidhya, and Medium. In the development of our project, we have independently coded two key components: the model fusion combining ResNet50 and VGG16, and the preprocessing techniques involving image augmentation and enhancement. The model fusion facilitates the prediction of roads from satellite images, forming a crucial part of the overall functionality. This self-developed aspect demonstrates our understanding of combining different neural network architectures for enhanced performance. Additionally, our implementation of edge detection and morphological operations in the post-processing phase further exemplifies our capability to tailor algorithms to specific project needs.

Conversely, we have utilized external resources for certain parts of our project. Specifically, for the segmentation aspect, we referred to existing methodologies without altering their core principles. Furthermore, the Condition Random Field (CRF) component in the post-processing phase was also adopted from external sources. Regarding the U-Net model, we leveraged the implementation detailed at "https://www.kaggle.com/code/yash2410/mapunet". This resource served as a comparative benchmark, enabling us to evaluate our model's performance against established standards.

Let's understand how preprocessing, models, and post-processing were implemented.

Augmentation

We performed augmentation on our satellite photos and their matching outline masks. This helped us create more variety in our training data by slightly altering the images in different ways. We have rotated the images, shifted the images horizontally and vertically by 10% of total width and height respectively, also we have shared the images randomly by 10 degrees, zoomed in and out by 10% each, and even flipped them horizontally. We also stated the Strategy to fill newly created pixels, which can appear after a rotation or a width/height shift. We used the same settings to change the masks just like we did with the images. To make sure the images and masks still matched up after these changes, we used a specific method and a consistent setting across both. This made sure our altered images and masks lined up perfectly, which is important for teaching our system effectively.

Segmentation

In our code for working with satellite images and their matching masks, we start by loading these images and masks. Then, we resize them to make sure they all have the same size. After resizing, we do a step called thresholding, which helps to bring out the important parts of the images. Next, we use a special technique that picks out the specific areas we're interested in from the satellite images, guided by the masks. This results in a bunch of images that are neatly cut out and ready to use. These processed images are great for training computer programs to recognize and understand different parts of an image, like in image segmentation tasks. By using these well-prepared images, we make sure that our models learn accurately and perform well.

Image Enhancement:

For the Image Enhancement part of our project, First, we made use of boosting contrast clearer of the images. This was done using a tool from PIL, where we increased the contrast by 1.5 times. This made everything in the image stand out more. The Next step included, we turned the image into black and white, or grayscale, which helps focus on the light and dark parts rather than the colors.Then, we used a technique called histogram equalization with a tool from OpenCV, another Python library. This technique spreads out the light and dark parts of the image, making it easier to see details in areas that were either too bright or too dark. After that, we changed the image back into its original format using PIL and then made it sharper. Making the image sharper, twice as much as normal, helps to make the edges and small details in the image more noticeable.

Model Fusion:

In this project, we used deep learning to create a model for a particular task. ResNet50 and VGG16 are the two models whose features we combined. These models can be thought of as vast libraries of patterns that they have acquired through their use of the enormous ImageNet image collection. We only needed these models' capacity to learn patterns, so we didn't use the final product they produced. We concatenated the features that both ResNet50 and VGG16 had learned. Compared to using a single model, this combination improves our model's ability to comprehend images. We used layers called Conv2DTranspose and Conv2D to transform these combined patterns into something that would be helpful for our task. Additionally, we incorporated ReLU into our layers. This is a trick to enable the model to extract more intricate information from the pictures. In the end, we employed a layer configuration that is typical when attempting to extract features from the pictures. In this configuration, a Conv2DTranspose layer is used first, and then a Conv2D layer for gradually upsampling the images to the desired output. In the output layer we have applied sigmoid activation function , which provides probabilities in the range 0 to 1 for the binary mask of the images.

U_net:

The U_net model begins by performing a two-step, three-by-three window scan of the image to identify any patterns. As an activation function, we apply a technique called ReLU after every scan to ensure that we only retain relevant data. Next, we use a method known as max pooling to reduce the size of the image, which aids in the model's ability to concentrate on the crucial areas. As we go through these steps, we increase the detail the model sees in the image. We also use something called dropout layers. These layers randomly ignore some parts of the image during training, which sounds odd, but it helps the model not to rely too much on any one part of the image. This prevents what we call overfitting, where the model learns the training images too well but can't handle new images. Next, the model starts building back up to the original image size. It does this by reversing the shrinking process and adding back the details. We also bring back some of the patterns we saved from earlier to help the model understand where things are in the image. The final touch is a very small window (1x1) that looks at every single point in the image and decides if it belongs to the object we're interested in or not. Since we're only looking for one thing (binary segmentation), this step uses a special setting called sigmoid activation, which is great for binary masks.

Edge Detection:

In our project where we pull out road maps from satellite pictures, edge detection is a really helpful step after we predict the images. Here’s what we do: First, we check if the picture is in color. If it is, we change it to black and white using a tool called cv2.cvtColor. Black and white images are easier to work with for finding edges because they just show how light or dark things are, not the colors. The main part of this step is using something called the Canny edge detection algorithm. It's a smart way to find all sorts

of edges in the picture by looking for spots where the lightness or darkness changes quickly. We set it up with two numbers, 100 and 200, which help it decide what counts as an edge. Why do we care about edges? Well, in road maps, the edges help us see exactly where the roads are. This is super helpful for things like making maps, planning cities, or even for self-driving cars that need to understand the roads. When we're done, the pictures show the roads with clear lines, making it easier to see the road layouts and how they connect. This method is a neat way to make the roads pop out in the image, showing us just the important part of the roads.

Morphological operations:

We used morphological processing to make the satellite images clearer, especially the roads. Creating a Kernel like a small window or a brush that we use to refine the image. We make this kernel in an elliptical shape using a tool from OpenCV (a library for image processing). The size of this kernel is set by us, and the elliptical shape is chosen because it works smoothly on the images. First, we do an opening operation on the image. It's like first erasing and then drawing back details on the image. This is good for getting rid of small white spots that aren't part of the roads, kind of like cleaning up little specks of dust. Next, we do the closing operation. It's the opposite of opening – We illustrate details first, then we remove them. To make the roads in the photos appear more seamless and connected, fill in any small holes or gaps with this step. These steps are what we do to ensure that the roads in our photos are uninterrupted, unobstructed, and free of anomalous particles. To make just the roads stand out beautifully, it's like cleaning and polishing an image. Using this method helps ensure that our road maps are accurate and helpful for planning and navigation.

Conditional Random Field:

We are using CRF as post-processing for the predicted images, we start by adjusting the road predictions we already have so they work better with the CRF technique.At first, we flip the predictions, changing the roads to non-roads and vice versa, to get them ready for the next steps. We create ‘'unary energy' from these flipped predictions. Think of it as a way to tell the CRF how confident we are about each part of the image being a road or not. If our image isn't in the right format, we change it so it is. This is important for the CRF to work properly. We set up the CRF model with the image's dimensions and tell it there are two things to choose from: road or not road. We added some rules to the CRF about how to make its decisions. These rules help the CRF make smoother and more natural-looking road predictions, considering both how the image looks and how the colors change. The CRF then takes all this information and makes a final decision on where it thinks the roads are in the image. It does this a few times to be more accurate. We reshape the CRF's decisions to match our image's layout. We do all of the above steps for each image we have. For each image, we take our initial road prediction, refine it with CRF, and then save this refined prediction. In summary, this code helps us refine our initial road predictions to make them more accurate and realistic. It's like giving the CRF a rough sketch of where we think the roads are and then letting it fine-tune that sketch into a clearer picture. We preferred to take reference from online code.

Experimental Protocol

We have used the Satellite Road images dataset from Kaggle. It consists of 1 metadata CSV file, 1 class_dict CSV file, and 3 folders namely train, test, and val. The train contains 12452 images, including masking and satellite images, while the test contains 1101 and valid contains 1243 satellite images.
First, we trained our models on the training dataset where we split the 6226 satellite images and 6226 masking images in an 80:20 ratio where 80% of the data is used for training and 20% for validation, and then to compare the results we needed to predict the masking of the test dataset and see how the model is predicting. We have implemented the setup on various parameters such as experimenting with different preprocessing and post-processing techniques. In testing the dataset ground truth of the images was not present.
Since this dataset contains the maximum number of images, we preferred this over another. In addition to that, it also contains the masking images in the training dataset which are helpful in the model training and preprocessing part. From which model gets an accurate idea for training purposes.
For evaluating the success we are using the testing dataset, Initially, we also thought of using F1, Precision, and Recall, but we were not able to use these evaluations as we do not have masking images for the testing dataset. So, To check how our models have performed well, we have plotted the predicted masking images from the models and compared those with actual images. We have used the absolute path for accessing the pickle file for model training purposes and for that, we have defined the absolute path. Hence to run a code on test images you first need to download the file and change the path accordingly to get an output.
This project is very computationally heavy since we have to train a large dataset of 3.86 GB. Additionally, we also trained the U-Net model in the provided code comprising a total of 18 Conv2D layers, 4 MaxPooling2D layers, 9 Dropout layers, 4 Conv2DTranspose layers, 4 Concatenation operations, and 1 Output Conv2D layer, which sums up to a total of 40 individual layers and model fusion consisting of Resnet and VGG with Layers from ResNet50 (excluding the top) are 48 layers, Layers from VGG16 (excluding the top) are 16 layers and Custom layers are added 1 Input, 1 Concatenation, 6 Conv2DTranspose, 5 Conv2D, 1 Output layers, in total they are 14 layers, which are very computationally heavy models. We have made sure the image size is constant throughout the preprocessing, model training, and post-processing process which is (224,224). For the training purpose, we have made sure the optimizer is Adam, the number of epochs is 20, the loss is binary cross entropy and the metric is accuracy. Hence runtime required for each part that is preprocessing, postprocessing, and model training is over 60 manhours per person. We have used the Google Colab version for running such huge models.

Results

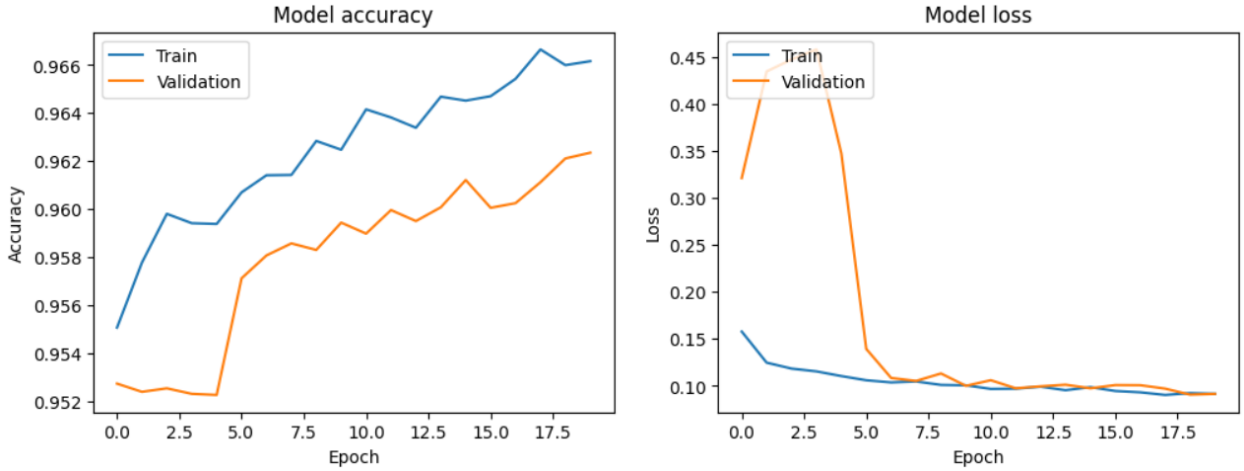
From the preprocessing model implementation and post-processing, we have achieved these results. From the image itself, it is better to understand. We have divided it into 3 parts that are Augmentation, segmentation, and Image enhancement. In that Modelfusion then training graphs for the model post-processing on the output and then a similar sequence for Unet.

1] Data Augmentation

Training Data

Training Analysis

Model Fusion



This image indicates that Model Fusion provides an accuracy of 96.6% on the training dataset and 96.2% on validation for data augmentation as preprocessing.

Prediction by model fusion

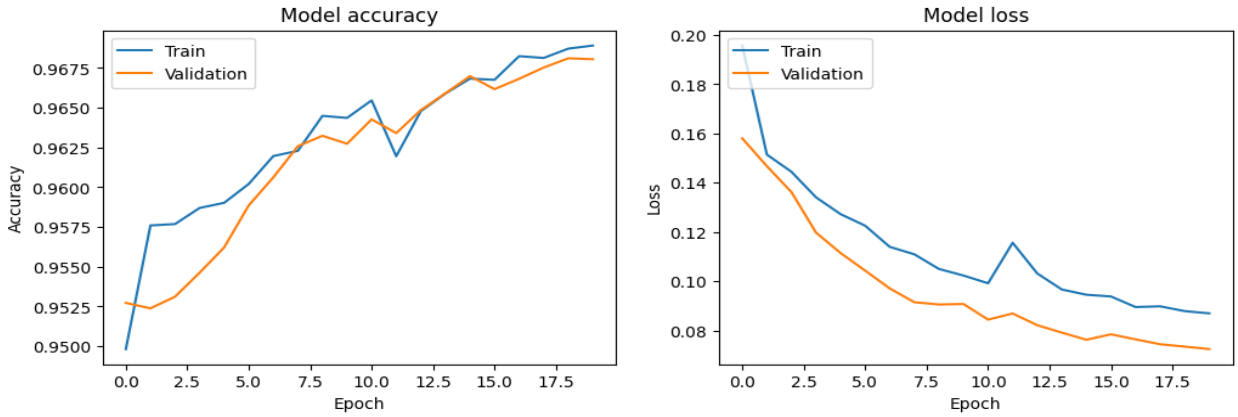
Using Post-Processing Methods:

- **Edge Detection**

- **Morphological Operations:**


- **CRF:**


U_Net:



Both validation and training accuracy are increasing in data augmentation preprocessing with unet model from 95% to 96.75%

Predicted Image:

Post Processing:

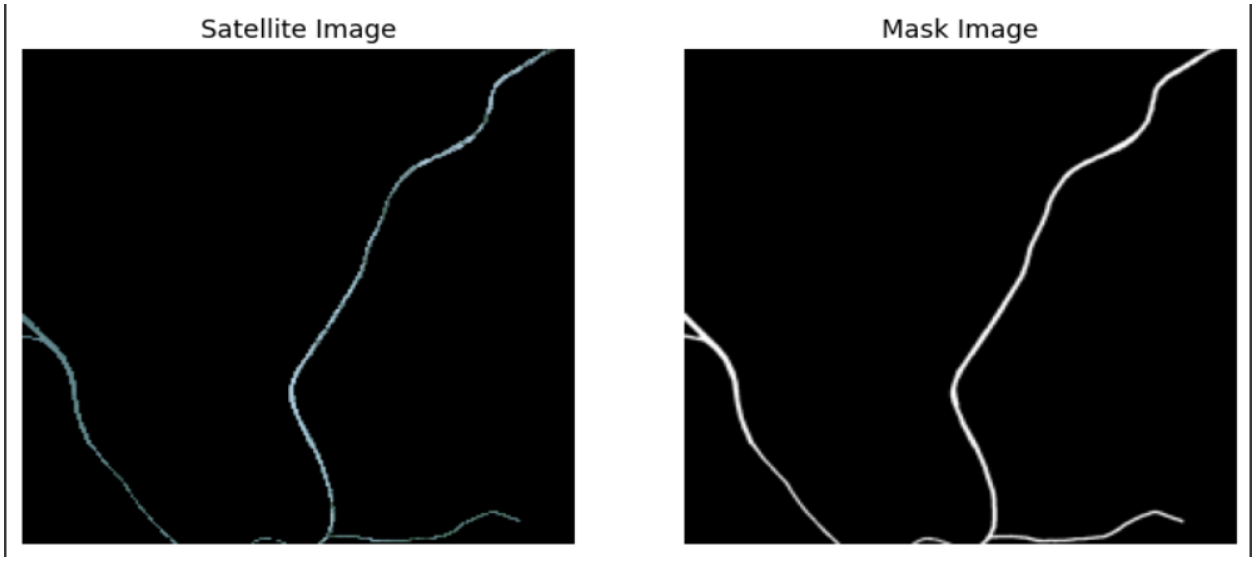
- **Edge Detection**

- **Morphological Operations:**

- **CRF**

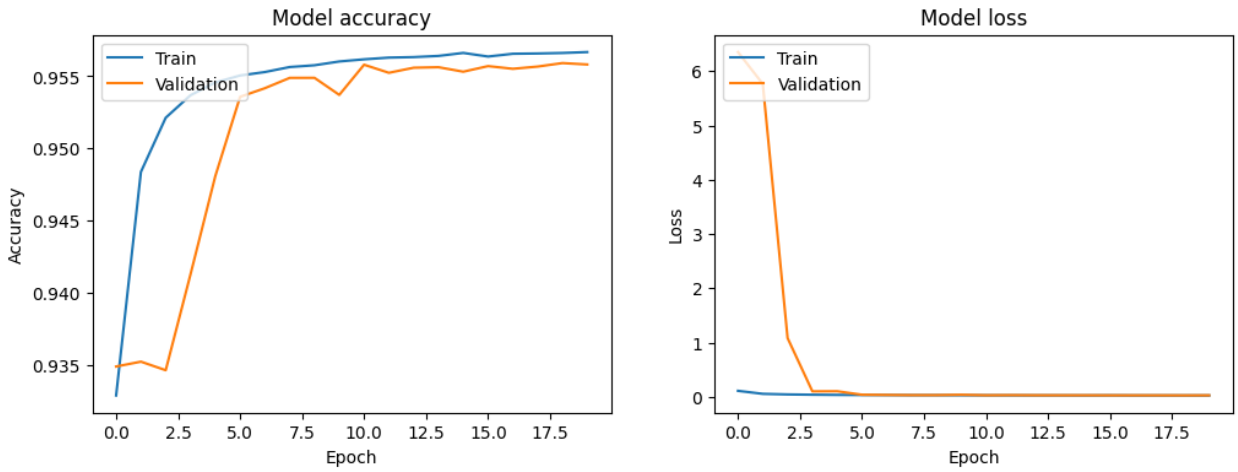
2]Segmentation:

Training Data:



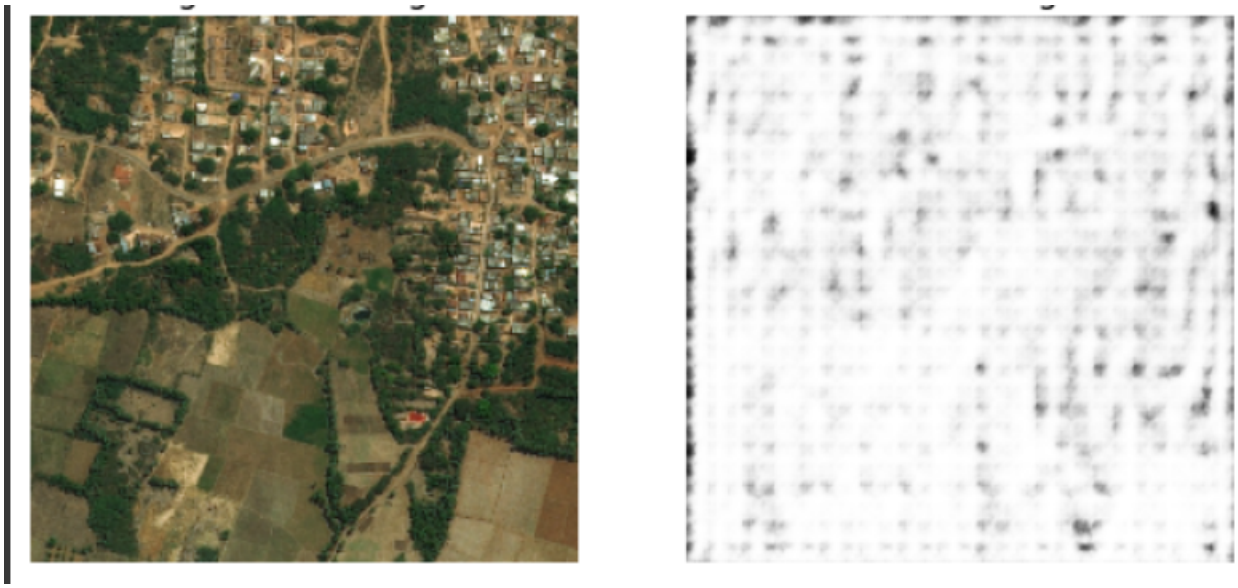
Model Fusion:

Training graphs

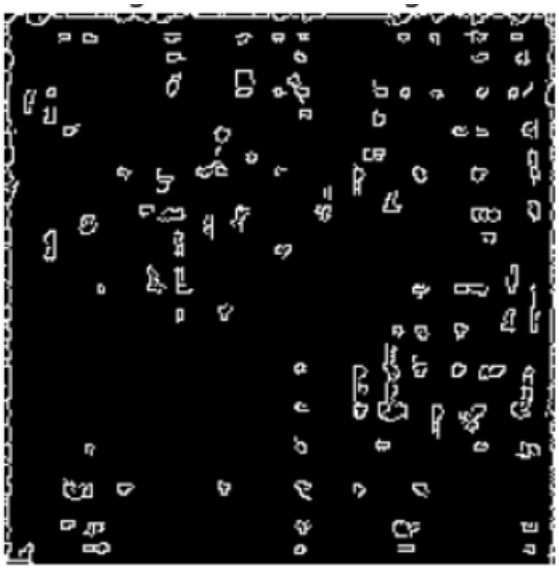


In the above graph it is shown that training model fusion gives 95.5% accuracy and almost the same on validation data for segmentation as preprocessing.

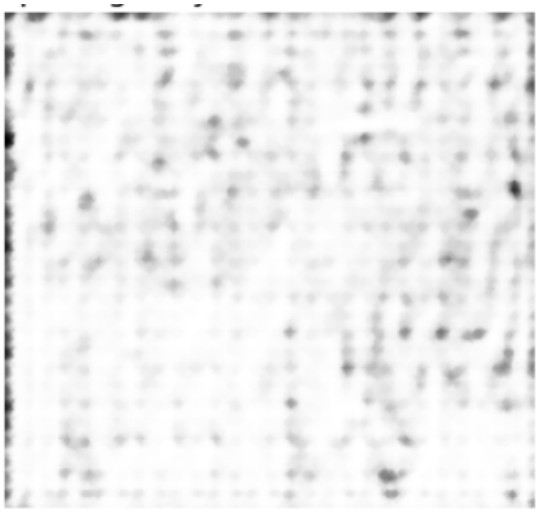
Predicted Image:



Edge Detection:



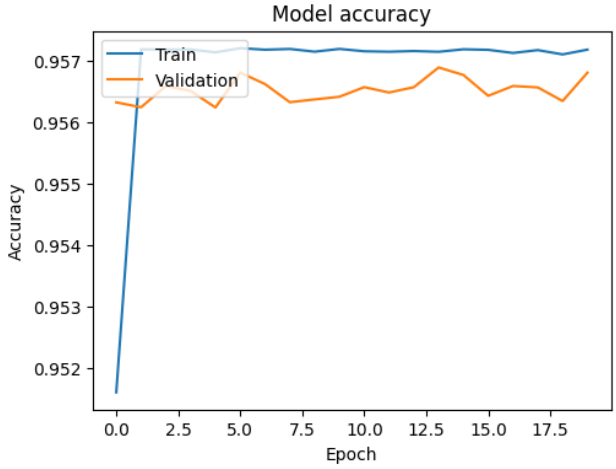
Morphological:

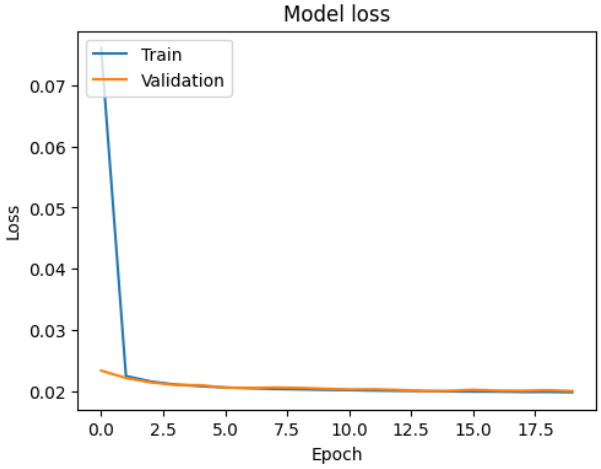


CRF:

U_net:

Training Classifier

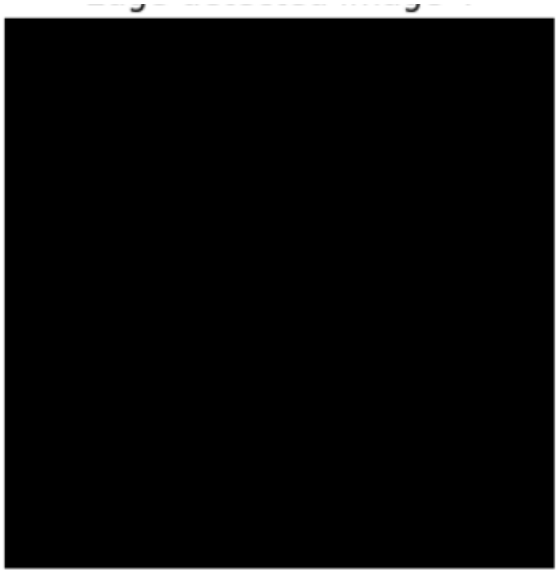




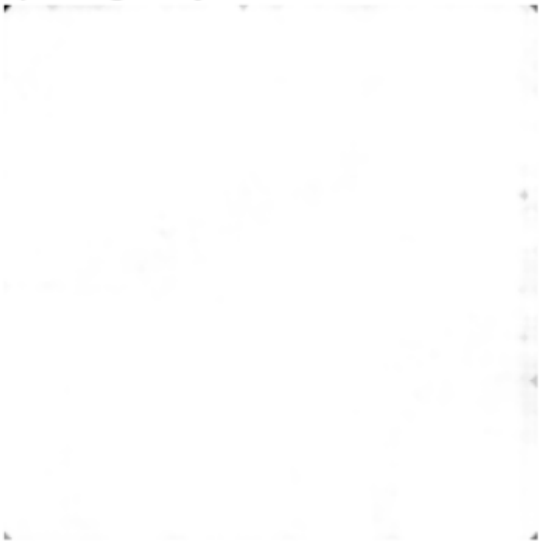
U net model for image segmentation gives 95.7% accuracy for training and almost similar for validation dataset.

Predicted Images:

Edge Detection:



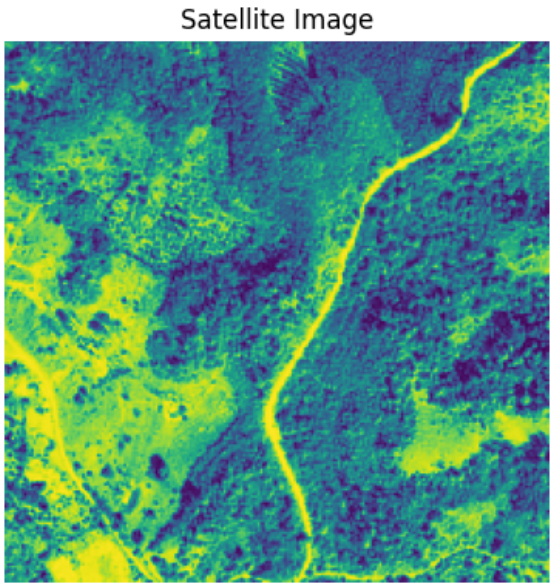
Morphological operations:

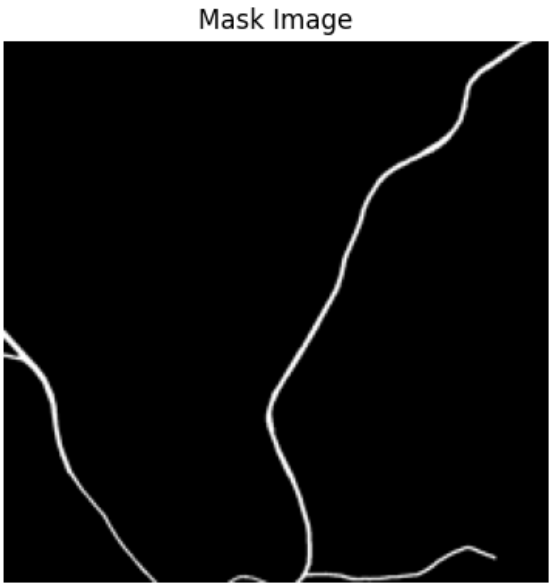


CRF:

Image Enhancement:

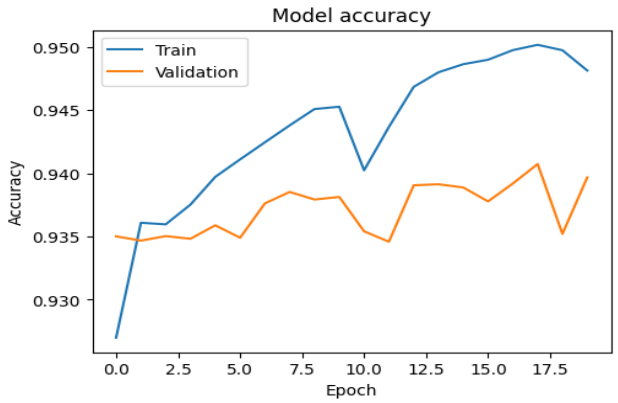
Training Data:

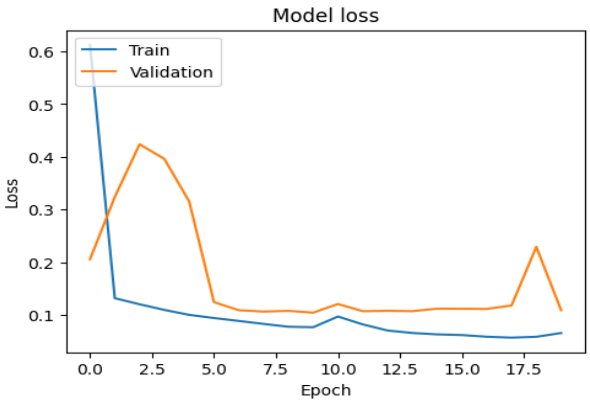




Model Fusion:

Training graphs



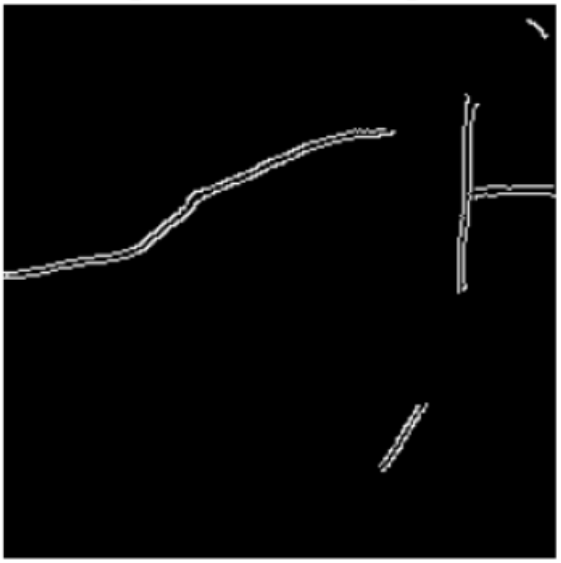


For Image Enhancement as a preprocessing technique model fusion gives 94.5% accuracy on training and 94% on validation dataset.

Predicted Image on training data:

Post-processing:

Edge Detection:

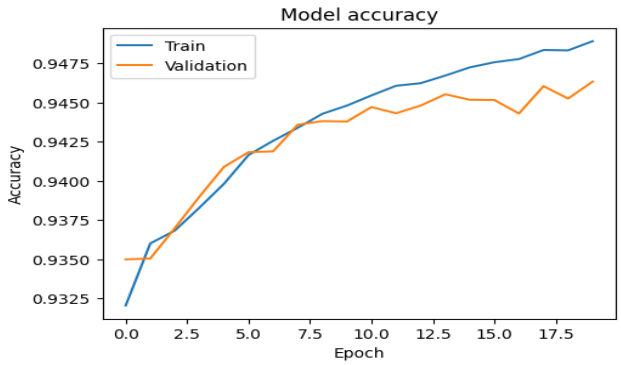


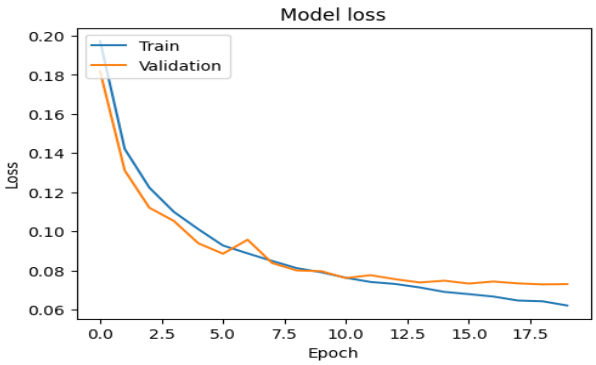
Morphological Operations:

CRF:

U_net:

Training graphs:





The U-net model provides over 94.75% accuracy on training and 95.50% for validation dataset.

Predictions:

- **Edge Detection:**

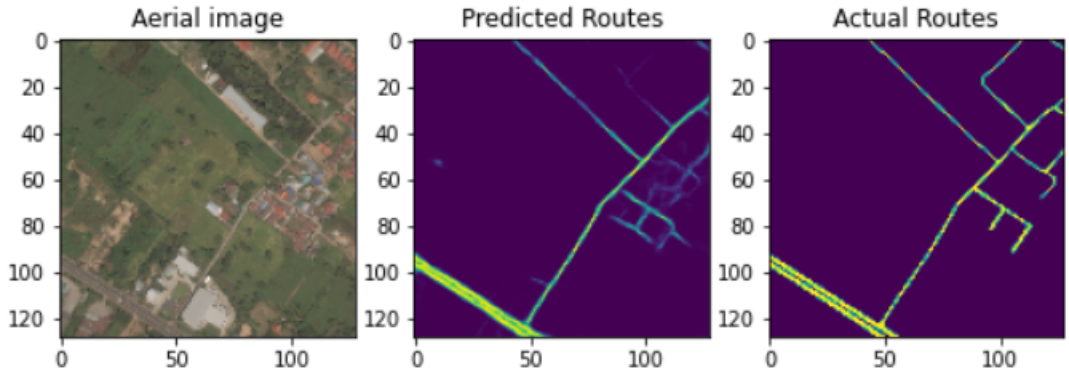
Morphological Operations:


CRF:


Overall observing from all images, Data Augmentation pre-processing with Unet as a model and edge detection as a post-processing technique shows better output, after edge detection the morphological operations and CRF provide better output whereas, for the segmentation part, we are not getting the expected output as in the case of augmentation and image enhancement. For the Image enhancement part, while comparing it with the augmentation outputs , it is not extracting the maximum features for roads. Also in the U net model with edge detection as post-processing and has almost similar output after morphological operations. While Performing edge detection as a post-processing technique, we have used directly used the sigmoid probabilities predicted by the model since applying thresholding was providing blank images. For other parts of the post processing we have set a threshold for 0.1, the reason for using the low threshold is we did not want to lose information hence we minimized it as we could.

Initially we tried using 0.5 as threshold value and then reducing it gradually to check how we are getting binary images, but many of the information was getting lost.

Kaggle from where we have taken the dataset, some work on Unet has been done and the following is the output for that:



Output given after Edge detection and Morphological operations is similar to the output they are getting in state of the art.

Accuracy Table:

Models	Validation Accuracy
Unet (Augmentation)	96.75%
Unet (Segmentation)	95.70%
Unet (Image Enhancement)	94.50%
Model Fusion (Augmentation)	96.20%
Model Fusion (Segmentation)	95.50%
Model Fusion (Image Enhancement)	94.00%
Kaggle U_net(40 Epochs)	97.83%

Analysis

In our project, we conducted a detailed analysis of the algorithms used for road extraction from satellite images. This analysis revealed that the U-Net model, specifically designed for image segmentation tasks, outperformed our model fusion of ResNet50 and VGG19. Despite the latter's known strengths in robust feature extraction - with ResNet50 excelling in deep layer learning and VGG19 in capturing intricate details - they did not match U-Net's performance in our specific application. This finding highlights the specialized efficiency of U-Net in handling segmentation tasks, particularly in extracting precise road features from complex satellite imagery.

Our analysis also identified limitations in the segmentation phase for both the Model Fusion and U-Net models. We encountered issues such as blank and blurred image outputs, indicating a shortfall in the preprocessing steps of both models. This insight suggests that the preprocessing methods currently in use may not be entirely suitable for the task of road extraction from images. This part of our analysis not only underscores the importance of algorithm selection for specific tasks but also emphasizes the need for ongoing evaluation and refinement of our methodologies. The proficiency in analyzing these outcomes demonstrates our commitment to optimizing our application and ensuring its reliability and accuracy.

Discussion and Lessons Learned

We have tried a new approach to solving a problem by implementing a model fusion along with VGG to compare the results. It is predicting almost similar to Unet which is pre-established, but in comparison, U net is providing more accurate results.
We learned that trying out different approaches is necessary as sometimes it may give wrong answers but that will guide you to the right path. As in machine learning, and computer vision we need to try different algorithms on the same problem or even the same algorithm with different parameters in it.
In the project we learnt that segmentation is not providing the desired results. Few assumptions for such behavior might be , as we are providing segmented image to the model which is black and highlighted the road according to the masks and when testing on the images we are providing original image , so the model might not able to provide the desired results.
Computational resources required for running and implementing the codes are very high hence it is taking a lot of time to run. There are a lot of implementations like every model requiring correct data as input since it trains on that.

Bibliography

Paper 1:
Demir, I., Koperski, K., Lindenbaum, D., Pang, G., Huang, J., Basu, S., ... & Raskar, R. (2018). Deepglobe 2018: A challenge to parse the Earth through satellite images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (pp. 172-181).
Paper 2:
Shen, W., Chen, Y., Cheng, Y., Yang, K., Guo, X., Sun, Y., & Chen, Y. (2021, July). An improved deep-learning model for road extraction from very-high-resolution remote sensing images. In *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS* (pp. 4660-4663). IEEE.
Paper3:
Wang, H., Yu, F., Xie, J., & Zheng, H. (2022). Road extraction based on improved DeepLabv3 plus in remote sensing image. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *48*, 67-72

References:

https://medium.com/
https://www.analyticsvidhya.com/
https://www.geeksforgeeks.org/
Model Fusion:
https://datascience.stackexchange.com/questions/86769/error-after-merging-two-deep-learning-models-vgg16-and-resnet50
https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/
Unet:
https://www.kaggle.com/code/devesudev/mini-project-road-map-from-satellite-images
https://www.kaggle.com/code/ekremzturk/satellite-image-road-segmentation
https://www.geeksforgeeks.org/u-net-architecture-explained/
https://discuss.pytorch.org/t/u-net-output-has-256-channels/161949
Segmentation:
https://realpython.com/python-opencv-color-spaces/
CRF
https://pseudo-lab.github.io/SegCrew-Book/docs/Appendix/DenseCRF.html
Image Engancement
https://www.geeksforgeeks.org/image-enhancement-techniques-using-opencv-python/