Final Report

Aerial Perspective Object Detection

Prepared by:

Jukai Hu	400485702
Ray Albert Pangilinan	400065058
Luke Vanden Broek	400486889

December 12, 2022

Table of Contents

Table of Tables

Table of Figures

1. Introduction

1.1 Problem Statement

In this project, the goal was to train a model to identify the different objects in a birds' eye view. This is an object classification question, where input images (in this case, taken from drones) are fed to the algorithms which will then classify the objects in those images with different labels (eg, "paved-area", "dirt", "grass", etc.) based on different colors assigned to each pixel.

1.2 Approach

We first tried to solve the segmentation problem with three conventional machine learning approaches (ML): Random Forest, KNN (k-nearest neighbors), and Gaussian NB (naive Bayes). Firstly, to create a training dataset for image classification, we should have annotated images with labels that will be used by the algorithm. Here in our project, we were provided with the label images, semantic images and the original RGB color images. Then, we fed those images to our supervised algorithms for training. After that, we derived a classification of objects in input images based on different colors assigned to those objects. This is because we would have already trained our models with the training dataset, in which all objects are separated from each other using different pixel values.

Next, we pre-processed the data to prepare it for loading into a deep convolutional neural network (CNN) model. We then investigated three different CNNs to consider applying to the dataset: VGG-16, ResNet and Inception. Finally, we chose VGG-16 as our predictive model and applied it to work with the dataset to see if it proved better than the traditional ML techniques on behalf of final accuracy and computation complexity. All code for this report can be found in the linked GitHub page (Hu *et al.,* 2022).

1.3 Task Description

Based on the classification of image pixels to different classes, as well as our project outcome to identify objects such as cars, people, and roads in drone imagery, it was determined that the main goal of our project was a classification task. Since the classes were provided by the dataset, we were able to infer that our problem falls under the category of supervised learning. Furthermore, this task can be extended to a multi-class classification given the quantity of twenty-four labels outlined by the dataset.

2. Dataset Information

2.1 Dataset Description

The dataset used for this project includes 400 different images at resolutions of 4,000 pixels x 6,000 pixels. Included in the files are images of the data in their original format, semantic format and with an RGB color mask to identify different objects, as well as a .csv

file that indicates the RGB color values representing each classification type (*Semantic Drone Dataset*, n.d).

2.2 Data Visualization

Data in Table form

	name	r	g	b
0	unlabeled	0	0	0
1	paved-area	128	64	128
2	dirt	130	76	0
3	grass	0	102	0
4	gravel	112	103	87
5	water	28	42	168
6	rocks	48	41	30
7	pool	0	50	89
8	vegetation	107	142	35
9	roof	70	70	70
10	wall	102	102	156
11	window	254	228	12
12	door	254	148	12
13	fence	190	153	153
14	fence-pole	153	153	153
15	person	255	22	96
16	dog	102	51	0
17	car	9	143	150
18	bicycle	119	11	32
19	tree	51	51	0
20	bald-tree	190	250	190
21	ar-marker	112	150	146
22	obstacle	2	135	115
23	conflicting	255	0	0

Figure 1: RGB colour values in tabular form

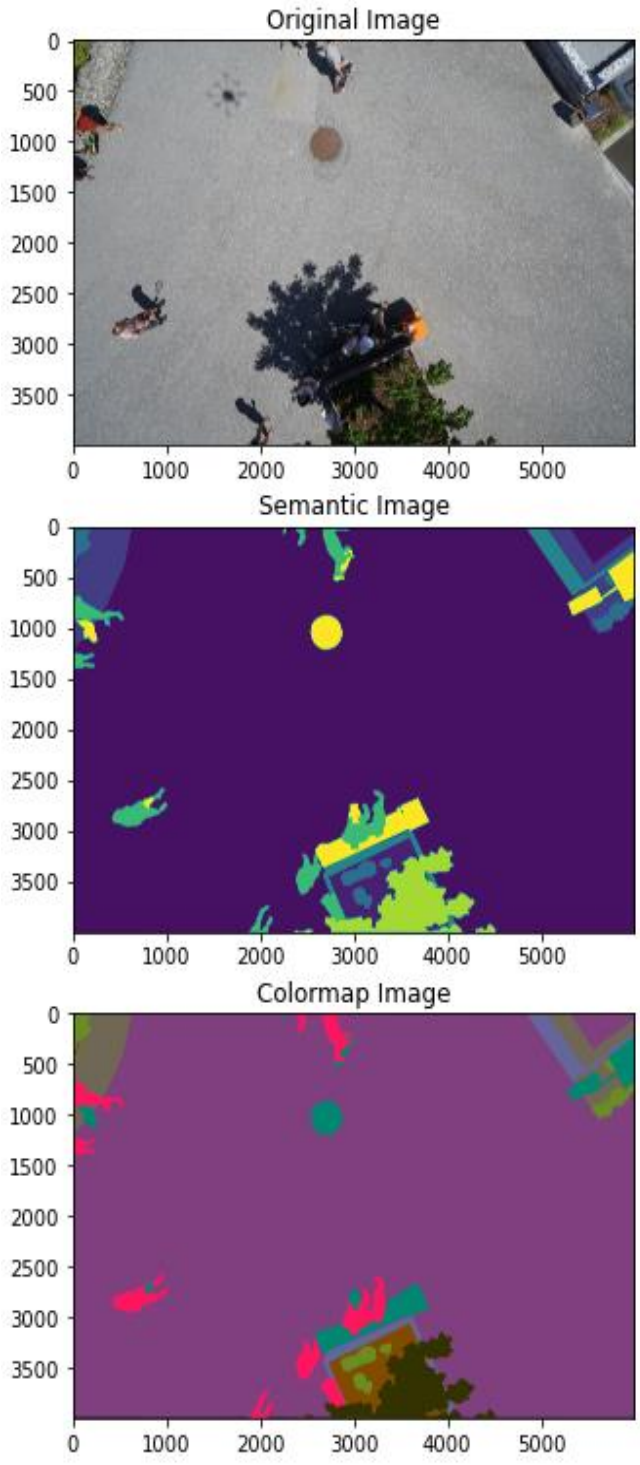

Figure 2: Plot of image 000 in original, semantic and colormap formats

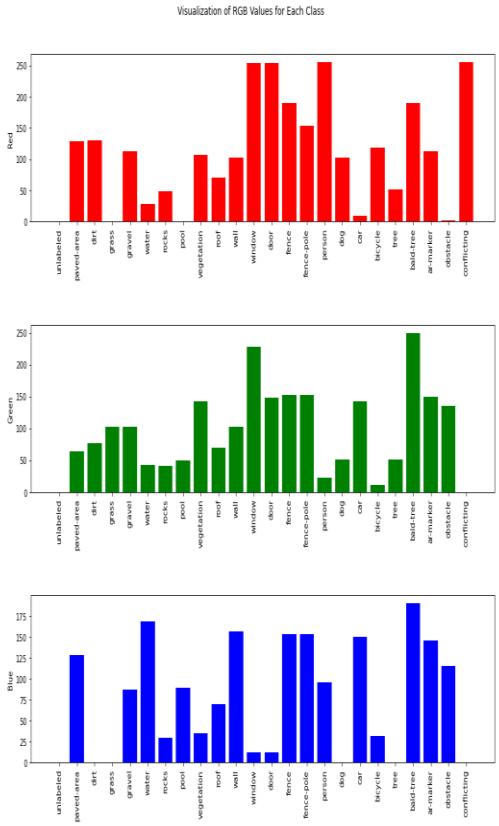

Figure 3: Visualization of RGB color values in histogram form

2.3 Data Distribution

Figure 4: Legend for colormap values

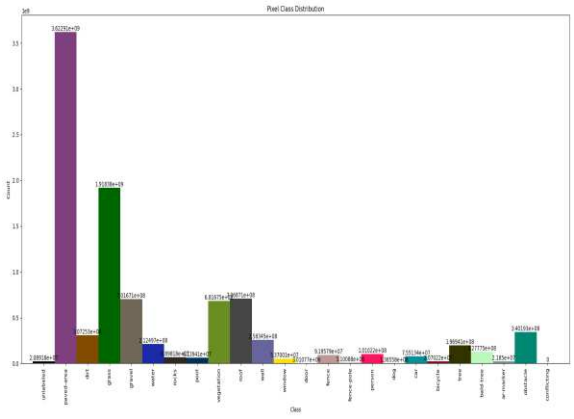

Figure 5: Pixel distribution for full dataset

3. Machine Learning Solution

3.1 Process

We began work on our project task by first determining how we might structure the data for input into machine learning models. Given that our raw data is in the form of images, we imported them into a Python script as three-dimensional numpy arrays of dimensions 4000 x 6000 x 3, corresponding to the pixel width and height of the images as well as the RGB values of each pixel. We then flattened the initial array, resulting in a two-dimensional array with dimensions 24,000,000 x 3, with each pixel representing a data point and the RGB values representing features. Performing the same data manipulation steps on the semantic images gave us our pixel labels in a matching format. With each image in a format sufficient for training conventional machine learning algorithms, we fitted nine images in the dataset to Random Forest, K-Nearest Neighbours, and Naive Bayes models, and predicted the class labels for a tenth image.

While we would have liked to expand our training and test sets to include more of the dataset, as well as introduce cross-validation, the extremely high image resolution resulted in training times of about an hour. We also attempted to train a support vector machine model alongside the three aforementioned algorithms but did not see the algorithm proceeding past training the first image. The predictions of each machine learning model are outlined below in Figure 6.

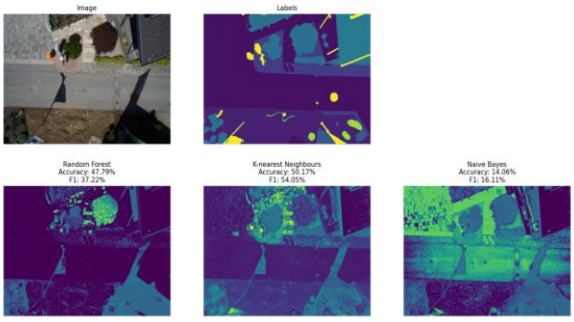

Figure 6: Results of machine learning techniques

3.2 Results

Analyzing the prediction results of the three chosen machine algorithms, we noted their performance to be underwhelming, with both Random Forest and K-Nearest Neighbours posting accuracy ratings and F1 scores around 40-50%. Additionally, naive Bayes did not perform well at all, with about 14% accuracy and 16% F1 score. Switching to a visual perspective of the results, we observed that the algorithms tended to incorrectly classify areas covered by shadows, as well as show lackluster results in terms of segmenting regions of the image. We theorize that this was due to our algorithms performing classification on a pixel-by-pixel basis, as well as using the RGB values of the pixels as the only features. Furthermore, we believe that more complex approaches such as feature selection, normalization, hyperparameters, and the use of convolutional neural networks will help alleviate these issues as well as the long processing times noted above.

4. Data Preprocessing

4.1 Normalization

To normalize our image dataset, we divide each image by 255 after importing them as numpy arrays, as per the line of code below. This results in the RGB values of each pixel being normalized to a range of 0 to 1 as opposed to the traditional range of 0 to 255.

```
image = np.array(Image.open(images_dir + image_name)) / 255
```

However, normalization ended up being unnecessary for our final code as we loaded the dataset as a TensorFlow DataFrame. When loading the data in this way, it automatically normalizes the data, making it unnecessary to utilize the code above.

4.2 Image Resizing

The Semantic Drone Dataset consists of 400 images at a resolution of 4,000 x 6,000 pixels out of the box. However, we have experienced issues with importing and processing images of such high resolution due to the high amount of memory required for a single image. Furthermore, we believe

that the initial pooling of each image to achieve a more manageable processing size will result in large amounts of data loss due to the large amount of downsampling. To alleviate this, we decided to first downsample each image down to a resolution of 80 x 120 pixels. Furthermore, each image was divided into four equal quarters, resulting in a longer dataset containing 1,600 images at resolutions of 40 x 60 pixels. An example of the image resizing results is shown in Figures 7 and 8.

Figure 7: Original vs tiled RGB image (000.jpg)

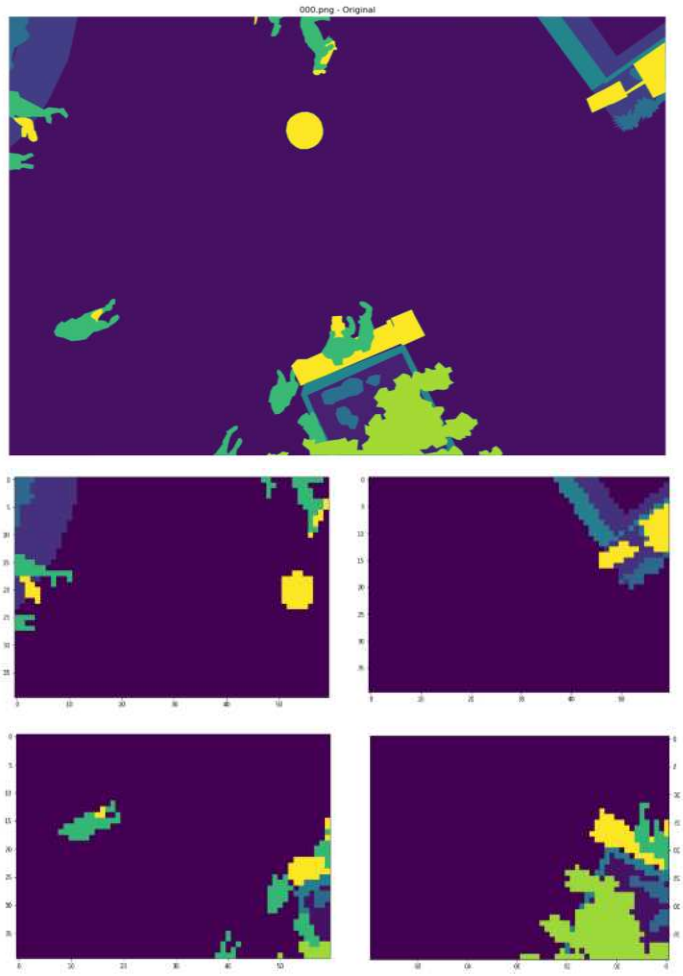

Figure 8: Original vs tiled semantic label image (000.png)

4.3 Missing Values and Outliers

As seen previously in Figure 5, the pixel distribution of our dataset reveals that about 20 million pixels are unlabeled, corresponding to about 0.2% of the total 96 billion pixels. In terms of outliers, due to the dataset being sourced in a similar and our task's designation as image segmentation, we believe that outliers have been taken care of prior to the release of the dataset. Furthermore, since we are working with an image segmentation problem, which requires

classification of regions of an image as opposed to the individual pixels, we believe that missing values and outliers will have a negligible impact, if any, on our model's output.

4.4 Balanced vs. Imbalanced Dataset

Returning to the dataset's pixel distribution shown in Figure 5, some classes such as paved areas, grass, and gravel make up the vast majority of pixels, while other classes like people, cars, and bicycles are present in a much smaller proportion. This gives off the impression of an unbalanced dataset from a pixel-based perspective. We believe that this is expected in an image segmentation problem, as the classes with smaller pixel counts are what we would like to detect against an environment or background class. However, some classes that would fall under the "object to be detected" designation, particularly doors, fence poles, and dogs, are not as present compared to others. Similarly, some "environment"-type classes like rocks, pools, dirt, and water appear less commonly. To balance out the dataset, many of these classes could be aggregated together with others boasting a higher pixel count. For example, rocks, gravel, and dirt could be combined into one "dirt" class. One important detail to note is that this aggregation would depend on whether this distinction between different classes is important in the neural network's final output.

4.5 Feature Selection

As our problem focuses on image segmentation, dataset features were extracted through our deep learning model, as opposed to being selected in preprocessing.

5. Research on Deep Learning Models

5.1 Very Deep Convolutional Networks for Large-Scale Image Recognition (VGG)

The VGG (Visual Geometric Group) model is one of the top DL (deep learning) models for image classification in use today (Huilgol, 2022). Introduced in 2014, it achieved great success at the ILSVRC conference, coming in second place overall for the competition that year (Simonyan & Zisserman, 2014). The goal of this model was to observe how adding depth through more layers would affect accuracy (Ayyar, 2020). It is a sequential model that is very deep and uses many filters and layers to increase its depth (Huilgol, 2022). It introduced the idea of using multiple convolution layers with smaller kernel sizes rather than one layer with a large kernel size (Ayyar, 2020). The model uses a series of 3x3 filters in each step of the process, and has a total of 16 or 19 layers, depending on if one uses the VGG-16 or VGG-19 variant (Simonyan & Zisserman, 2014). The VGG-16 model specifically uses a combination of 13 convolution layers and 3 dense layers, as seen in Figure 9 , while the VGG-19 has 16 convolution layers and 3 dense layers (Huilgol, 2022).

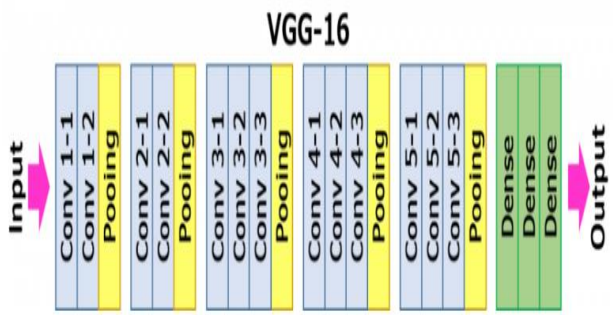

Figure 9: VGG-16 model architecture (Huilgol, 2022)

Pros:

- High accuracy and speeds when it was introduced (Ayyar, 2020)
- Increase in number of layers with smaller kernels increased non-linearities (Ayyar, 2020)
- Provided a base architecture for other similar concepts to develop from (Ayyar, 2020)
- Simpler model to understand in comparison to Inception and ResNet

Cons:

- Large model size with a large number of parameters can result in slower speeds than VGG and Inception (Huilgol, 2022)
- Vanishing and exploding gradients can be a notable issue in this architecture (Ayyar, 2020)

5.2 Residual Network (ResNet)

ResNet was introduced in 2015, and proved to be a great success winning many awards in the ILSVRC (ImageNet Large-Scale Visual Recognition Challenge) and COCO (Common Objects in Context) competitions that year (He *et al.,* 2016). The motivation behind residual networks was that as a neural network becomes deeper, it also becomes more difficult to train, resulting in accuracies that saturate and then degrade quickly (He *et al.,* 2016). The residual network architecture was actually introduced to resolve many of the issues from VGG, such as vanishing and exploding gradients (Ayyar, 2020). Residual learning works in a manner inspired by lateral inhibition in the human brain, where neurons are able to control the firing of neighbor neurons (Ayyar, 2020). This translates to the residual network as the model not needing to relearn a feature once it has already learned it the first time (Ayyar, 2020). To do this, this model introduces a DL residual framework, which allows the previous layers to fit a residual shallow mapping rather than directly fitting to the original mapping of the deep network (He *et al.,* 2016). The approach with this model is to add a shortcut connection that allows data to flow from one layer to another, but also skipping certain blocks if it has already learned them (Nandepu, 2019). An example of a ResNet module can be seen in Figure 10. This results in the addition of layers not hurting the model's performance, as if they are not useful they can be skipped, and if they are useful, the weights of the layers will be non-zero, and performance may increase (Nandepu, 2019).

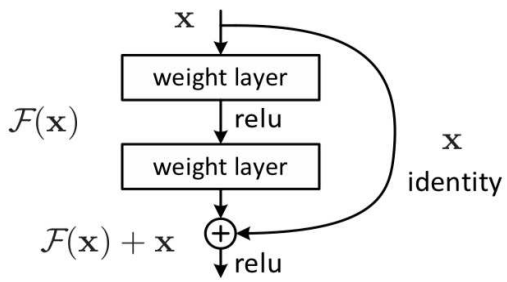

Figure 10: ResNet module architecture (He et al., 2016)

Pros:

- Resolves the vanishing gradient problem in other models such as VGG (Ayyar, 2020)
- Higher accuracies than VGG and Inception (Anwar, 2020)
- Higher speed and better computational power than VGG (Anwar, 2020)
- Adding more layers doesn't hurt performance, might even improve it (Nandepu, 2019(
- Resolves the degradation problem in VGG (Ayyar, 2020)
- Does not need to fire in every epoch as once it has learned a feature, it does not attempt to learn it again, reducing computational power (Ayyar, 2020)
- Low validation loss (Ayyar, 2020)

Cons:

- Skip connections can be difficult to work with, especially with varying dimensionalities between layers resulting in different output sizes
- Requires implementation of batch normalization throughout architecture (He *et al.,* 2020)
- Model size is still larger than Inception, slower speeds in comparison (Anwar, 2020)

5.3 Inception/ GoogLeNet

The Inception Model was introduced in 2014, similarly to VGG-16, and actually came first place overall in the ILSVRC competition that year, beating VGG-16 (Huilgol, 2022). Similarly to the previous models, the idea behind this model was to find a solution to the overfitting and computational issues that arise when creating a deeper model (Shaikj, 2020). The solution they came up with was to replace fully connected layers with a sparse structure that is approximately constructed using dense components (Szegedy *et al.,* 2014). The architecture used for each of the "Inception modules" consists of a 1x1 filter, a 3x3 filter, a 5x5 filter and a max pooling filter in series, and each of the outputs of these filters are concatenated into one output vector from the module (Szegedy *et al.,* 2014). However, the use of multiple 3x3 and 5x5 filters also adds to over-computation, so more 1x1 filters are used to reduce their dimensionality, resulting in the module architecture seen in Figure 11, and the overall model architecture seen in Figure 12.

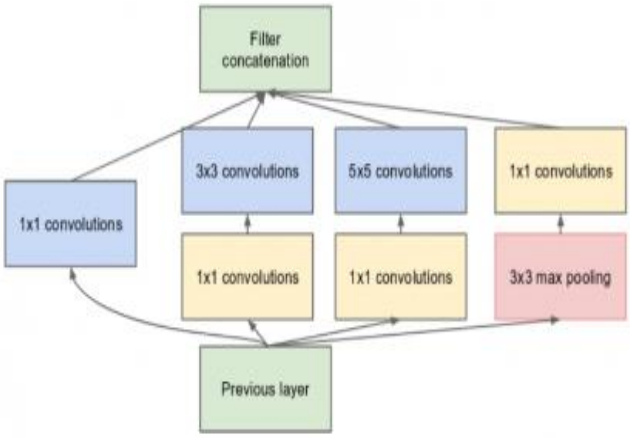

Figure 11: Inception module architecture (Szegedy et al., 2014)

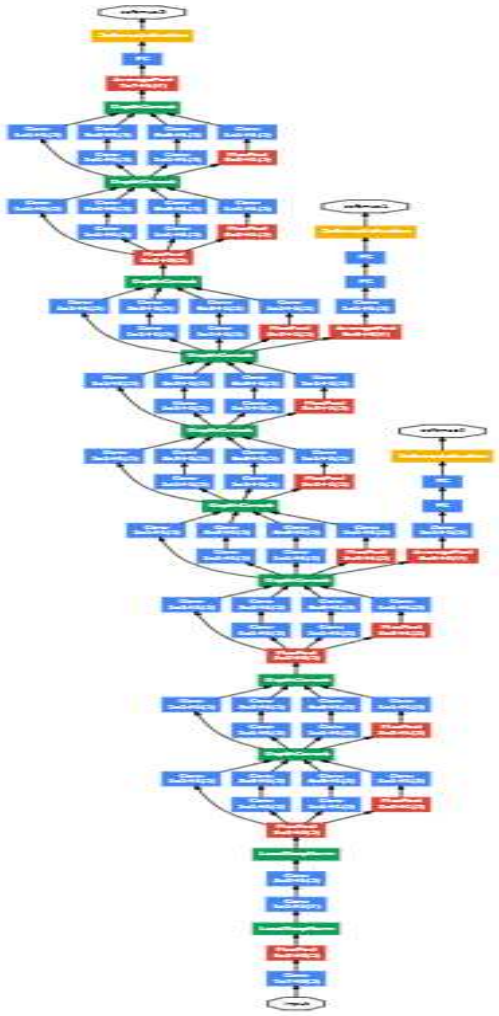

Figure 12: Inception full model architecture (Szegedy et al., 2014)

Pros:

- Lowest computational requirements (Anwar, 2019)
  - Higher speeds
  - Lower energy and memory requirements
- Can extract data at varying filter sizes
- Could be simpler to work with than ResNet

Cons:

- High accuracy, but not as high as ResNet (Anwar, 2019)
- More complex architecture than VGG

5.4 Deep Learning vs Machine Learning

VGG-16, ResNet and Inception are all DL CNN (convolutional neural network) models, which should provide higher accuracies than the traditional ML (machine learning) methods used in the first report. Traditional ML methods tend to only work with 1-dimensional vector data, rather than images which are 2- or 3-dimensional, depending on if they are grayscale or RGB (red-green-blue) (Lai, 2019). For a traditional ML model to be applied to an image, it either needs to be stretched to a 1-D vector or have its features extracted step by step (Lai, 2019). However, this causes issues as the stretched vectors stop previously adjacent pixel values from being near each other, and extracting features one step at a time can result in the relevance of features not being recognized by the model (Lai, 2019). In DL, the image can be loaded as a direct input to a CNN where its features are directly extracted via convolution, and the new data can then be upscaled or downscaled to analyze it at different resolutions (Lai, 2019). This gives DL algorithms a stronger learning and feature extraction ability than traditional ML methods (Lai, 2019).

6. Final Deep Learning Model

6.1 Model Description

The model selected for the final test results was a combined VGG-16 random forest model. This model's architecture is less

complex than the others, and should be able to effectively work on the semantic drone dataset. A summary of its architecture can be seen in Figure 13.

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 40, 60, 3)]       0

 block1_conv1 (Conv2D)       (None, 40, 60, 64)        1792

 block1_conv2 (Conv2D)       (None, 40, 60, 64)        36928

 block1_pool (MaxPooling2D)  (None, 20, 30, 64)        0

 block2_conv1 (Conv2D)       (None, 20, 30, 128)       73856

 block2_conv2 (Conv2D)       (None, 20, 30, 128)       147584

 block2_pool (MaxPooling2D)  (None, 10, 15, 128)       0

 block3_conv1 (Conv2D)       (None, 10, 15, 256)       295168

 block3_conv2 (Conv2D)       (None, 10, 15, 256)       590080

 block3_conv3 (Conv2D)       (None, 10, 15, 256)       590080

 block3_pool (MaxPooling2D)  (None, 5, 7, 256)         0

 block4_conv1 (Conv2D)       (None, 5, 7, 512)         1180160

 block4_conv2 (Conv2D)       (None, 5, 7, 512)         2359808

 block4_conv3 (Conv2D)       (None, 5, 7, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 2, 3, 512)         0

 block5_conv1 (Conv2D)       (None, 2, 3, 512)         2359808

 block5_conv2 (Conv2D)       (None, 2, 3, 512)         2359808

 block5_conv3 (Conv2D)       (None, 2, 3, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
```

Figure 13: Full VGG-16 model

The input variables fed into random forest in this project were derived after resizing (from 4000p x 6000 to 80 x 120), dividing (divided each 80px x 120px image into 4 40px x 60px tiles), and feature extraction by VGG-16, during which the dimension of the images would be noticeably modified. To match the dimension of the input variables of the random forest classifier with the dimension of input labels of random forest classifier, we applied the first two layers of the pre-trained VGG-16-random forest model (the input variables dimension would be 1600 x 40 x 60 x 3 and 1600 x 40 x 60 x 1). Meanwhile, if we used this model to predict an extremely large dataset, the hardware used for this code might not meet the heavy computational requirements for the full VGG-16 model. Based on these reasons, only the first two layers of the model were applied for this experiment, as shown in Figure 14.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 40, 60, 3)]       0

 block1_conv1 (Conv2D)       (None, 40, 60, 64)        1792

 block1_conv2 (Conv2D)       (None, 40, 60, 64)        36928

=================================================================
Total params: 38,720
Trainable params: 0
Non-trainable params: 38,720
_________________________________________________________________
```

Figure 14: VGG model used for this project (first two layers)

6.2 Model Results

First, the model extracted features from the training images, displayed below in Figure 15.

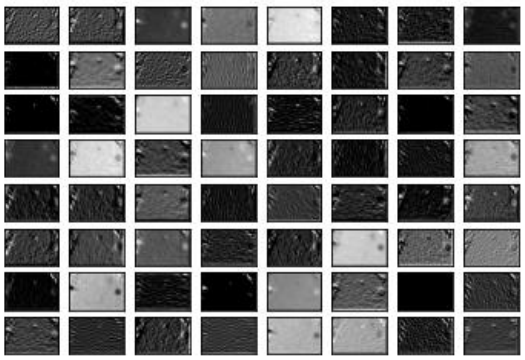

Figure 15: Features selected by the model

After fitting the model to the training data, and applying it to the test set, an accuracy of 66.35% was attained, as observed in Figure 16. Figure 17 shows the predicted class

output of the first image in the dataset relative to the original pre-processed image.

```
Accuracy =  0.6635034784647854
```

Figure 16: Model accuracy prior to grid search

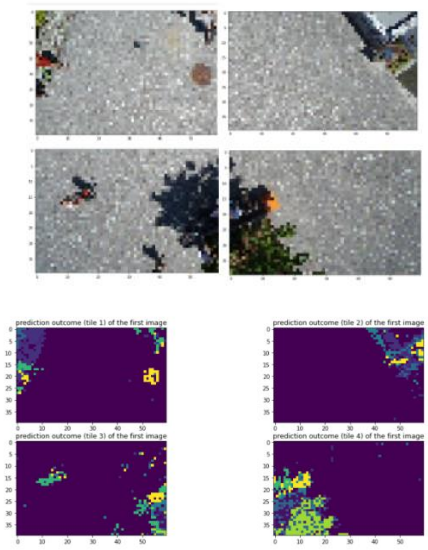

Figure 17: Original picture vs predicted classes

In an attempt to improve the model accuracy, grid search was used to find an optimal hyperparameter combination. The hyperparameters tested were criterion (being set to 'gini', 'entropy' or 'log_loss') and max_features (being set to 'sqrt' or 'log2').The results of the grid search can be seen in Figures 18 and 19, and it can be observed that the best results have criterion set to 'entropy' and max_features set to 'sqrt'.

Figure 18: Grid search results

```
Best score = 0.6610 using {'criterion': 'entropy', 'max_features': 'sqrt'}
mean test accuracy +/- std = 0.6596 +/- 0.0005 with: {'criterion': 'gini', 'max_features': 'sqrt'}
mean test accuracy +/- std = 0.6570 +/- 0.0006 with: {'criterion': 'gini', 'max_features': 'log2'}
mean test accuracy +/- std = 0.6610 +/- 0.0007 with: {'criterion': 'entropy', 'max_features': 'sqrt'}
mean test accuracy +/- std = 0.6589 +/- 0.0007 with: {'criterion': 'entropy', 'max_features': 'log2'}
mean test accuracy +/- std = nan +/- nan with: {'criterion': 'log_loss', 'max_features': 'sqrt'}
mean test accuracy +/- std = nan +/- nan with: {'criterion': 'log_loss', 'max_features': 'log2'}
```

Figure 19: Compact view of grid search results

After finding the best version of the model, it was then fitted to the training set and applied to the test set. This resulted in an accuracy of 66.54% as seen in Figure 20. The output prediction of the class labels can be seen in Figure 21.

```
Accuracy =  0.6653986120882079
```

Figure 20: Model accuracy after grid search

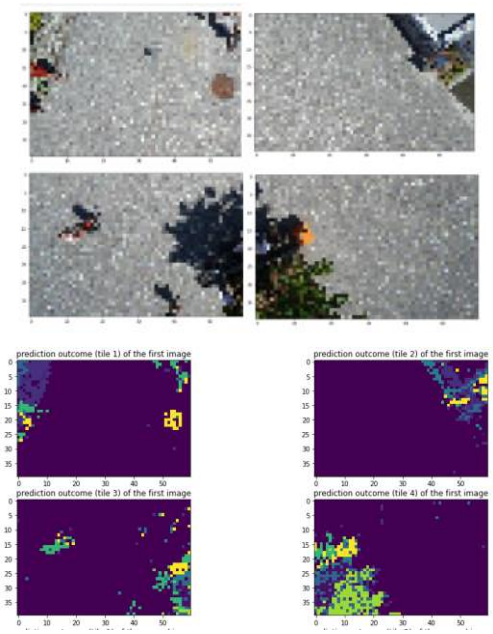

Figure 21: Original picture vs final predicted classes

In our report, we presented our final prediction accuracy generated with random forest after feature extraction (achieved with pre-trained VGG-16). The accuracy after and before fine tuning (Grid Search in our project) were derived and we conducted

comparison. Since the VGG-16 we applied was pre-trained, which suggests that the weight matrix we imported has already been heavily trained. Thus, we did not include the learning curves for training and validation for this model.

7. Conclusion

7.1 Comparison to Machine Learning Solution

The results of the final VGG-16 model proved to have an accuracy of 66.5%. This is much better than the accuracies of the traditional machine learning models, where the highest accuracy attained was 50.2% when using K-Nearest Neighbours. When comparing the accuracy to others who have worked on this problem, the accuracy performed well. Table 1 below shows comparisons between our model and other models posted on Kaggle.

Table 1: Accuracy comparison to Kaggle deep learning models

	Our Model	Kaggle Model 1 (Eldin, 2022)	Kaggle Model 2 (Hizal, 2022)
Model Basis	VGG-16	VGG U-Net	PyTorch
Final Test Accuracy (%)	66.5	75	79.3

7.2 Limitations

As observed above, there is certainly room for improvement for our model to reach better accuracies. In this project, the main limitations encountered were time constraints and hardware limitations. Our hardware limitations were both physical-based and cloud-based, as we were limited to a NVIDIA GTX 1060 laptop GPU and a NVIDIA T4 GPU through Google Colab. These also had an impact on our time constraints due to their slower compute times compared to top-of-the-line workstation and gaming GPUs. In order to improve the model in the future, more time would be required to build it, test hyperparameters and give it more training time. This is especially important for an image dataset where training times can be quite long due to the large file sizes. This relates to the hardware issues as well, as a more powerful computer, especially the GPU, would offset the longer training times required to process the large files properly through raw processing power, especially for a slower model such as VGG.

7.3 Future Steps

In terms of future work, it would be prudent to try to design the model in a way so that as many layers as possible could be used to achieve better results. In our project, we only applied the first two layers of VGG-16 to conduct feature extraction while still reaching a decent accuracy. Meanwhile, we would need access to a more powerful GPU in order to test stronger and deeper models to achieve a better performance. In addition, we would also like to attempt using the Inception and ResNet models as well, and compare their results with our observations from the VGG-16 model. We believe that the Inception model would likely run better on the hardware we have access to at the time of writing, as well as deliver improved results in

a more reasonable timeframe. If we were able to improve our hardware, we would want to use the ResNet model to optimize our accuracies.

References

Anwar, A. (2019, June 7). *Difference between AlexNet, VGGNet, ResNet, and Inception.* Medium. Retrieved November 1, 2022, from https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

Ayyar, T. M. (2020, November 6). *A practical experiment for comparing Lenet, Alexnet, VGG and Resnet models with their advantages and disadvantages.* Medium. Retrieved November 1, 2022, from https://tejasmohanayyar.medium.com/a-practical-experiment-for-comparing-lenet-alexnet-vgg-and-resnet-models-with-their-advantages-d932fb7c7d17

Eldin, O. E. (2022, November 25). *Aerial images semantic segmentation - U-net*. Kaggle. Retrieved December 6, 2022, from https://www.kaggle.com/code/omarreess/aerial-images-semantic-segmentation-u-net

He, K., Zhang, X., Ren, S., & Sun, J.. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* https://doi.org/10.1109/cvpr.2016.90

Hizal, C. (2022, November 6). *Semantic segmentation is easy with Pytorch □*. Kaggle. Retrieved December 6, 2022, from https://www.kaggle.com/code/ceyhzl3/semantic-segmentation-is-easy-with-pytorch#Evaluation

Hu, J., Pangilinan, R. A., & Vanden Broek, L. (2022). *AlbertPangilinan/ SEP740*. GitHub. Retrieved October 11, 2022, from https://github.com/AlbertPangilinan/SEP740

Huilgol, P. (2022, June 15). *Top 4 pre-trained models for image classification with python code.* Analytics Vidhya. Retrieved October 28, 2022, from https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/

Lai, Y. (2019). A comparison of traditional machine learning and deep learning

in image recognition. *Journal of Physics: Conference Series*, 1314(1), 012148. https://doi.org/10.1088/1742-6596/1314/1/012148

Nandepu, R. (2019, November 16). *Understanding and implementation of residual networks(resnets)*. Medium. Retrieved November 1, 2022, from https://medium.com/analytics-vidhya/understanding-and-implementation-of-residual-networks-resnets-b80f9a507b9c

Semantic Drone Dataset. Institute of Computer Graphics and Vision. (n.d.). Retrieved October 11, 2022, from https://www.tugraz.at/index.php?id=22387

Shaikh, J. F. (2020, May 14). *Deep Learning in the Trenches: Understanding Inception Network from Scratch*. Analytics Vidhya. Retrieved November 1, 2022, from https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/?utm_source=blog&utm_medium=top4_pre-trained_image_classification_models

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. https://doi.org/https://doi.org/10.48550/arxiv.1409.1556

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/https://doi.org/10.48550/arXiv.1409.4842