Report 2

Aerial Perspective Object Detection

Prepared by:

Jukai Hu	400485702
Ray Albert Pangilinan	400065058
Luke Vanden Broek	400486889

November 2, 2022

Jukai Hu 400485702
Ray Albert Pangilinan 400065058
Luke Vanden Broek 400486889

Table of Contents

1. Data Preprocessing **3**
    1.1 Normalization 3
    1.2 Image Resizing 3
    1.3 Missing Values and Outliers 7
    1.4 Balanced vs. Imbalanced Dataset 7
    1.5 Feature Selection 8
2. Deep Learning Models **9**
    2.1 Very Deep Convolutional Networks for Large-Scale Image Recognition (VGG) 9
    2.2 Residual Network (ResNet) 10
    2.3 Inception/ GoogLeNet 12
3. Conclusion **15**
4. References **16**

Jukai Hu 400485702
Ray Albert Pangilinan 400065058
Luke Vanden Broek 400486889

1. Data Preprocessing

1.1 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
```

1.2 Image Resizing

The Semantic Drone Dataset consists of 400 images at a resolution of 6,000 x 4,000 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 divide each image into four equal quarters, resulting in a dataset containing 1,600 images at a resolution of 1500 x 1000 pixels. An example of the image resizing results from the code block below is shown in Figures 1 and 2.

```
def divide_image(image, tile_height, tile_width):
 # RGB images
 if (len(image.shape) == 3):
   image_height, image_width, channels = image.shape

   tiles_arr = image.reshape(image_height // tile_height,
                             tile_height,
                             image_width // tile_width,
                             tile_width,
                             channels)

   tiles_arr = tiles_arr.swapaxes(1, 2)
   num_tiles = (image_height // tile_height) *
               (image_width // tile_width)
   tiles_arr = tiles_arr.reshape(num_tiles, tile_height,
                                 tile_width, channels)

 # Single channel images (labels)
 elif (len(image.shape) == 2):
   image_height, image_width = image.shape

   tiles_arr = image.reshape(image_height // tile_height,
                             tile_height,
                             image_width // tile_width,
                             tile_width)

   tiles_arr = tiles_arr.swapaxes(1, 2)
   num_tiles = (image_height // tile_height) *
               (image_width // tile_width)
   tiles_arr = tiles_arr.reshape(num_tiles, tile_height,
                                 tile_width)

 return tiles_arr
```

Figure 1: Original vs tiled RGB image (000.jpg)

Jukai Hu 400485702
Ray Albert Pangilinan 400065058
Luke Vanden Broek 400486889

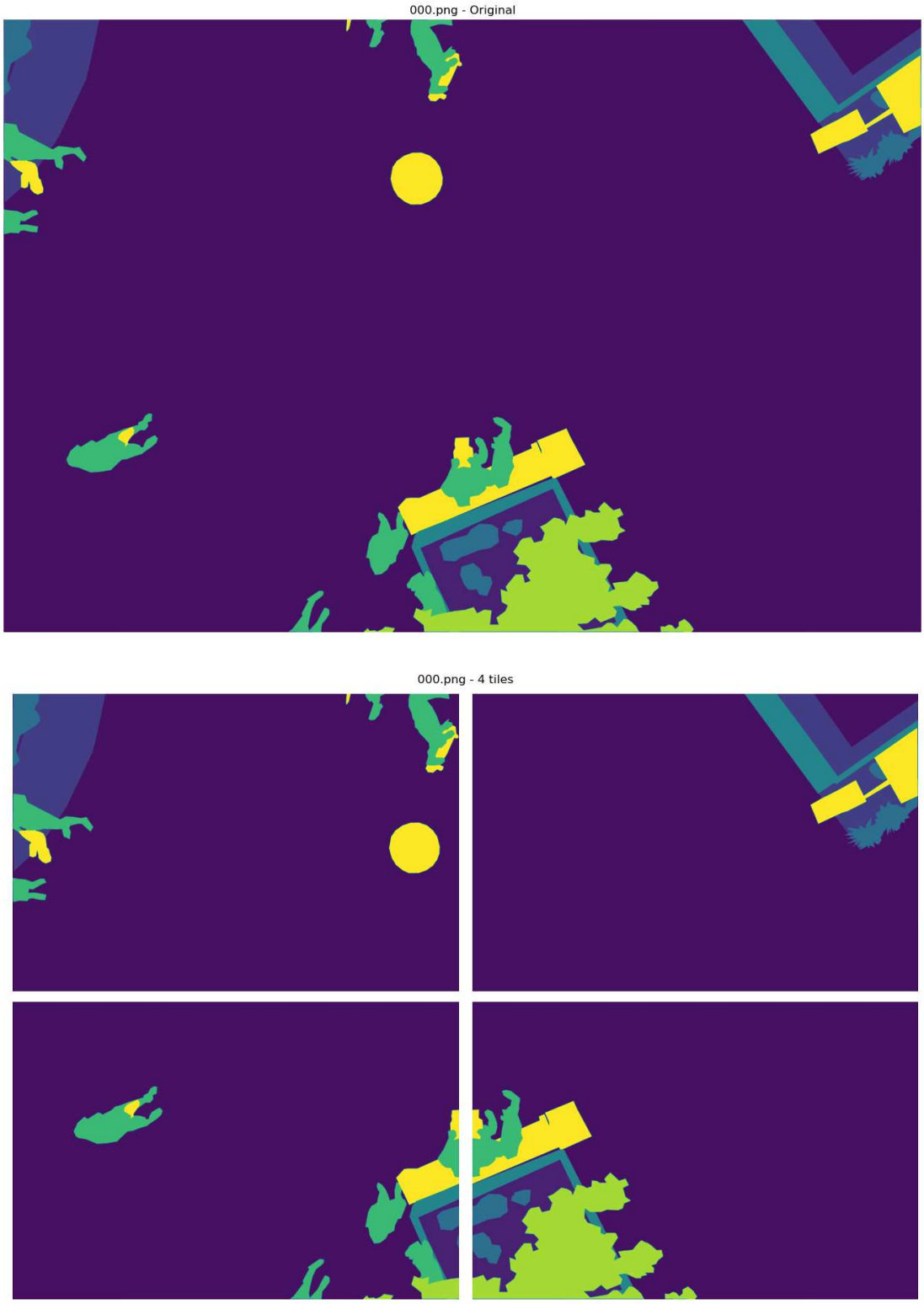

Figure 2: Original vs tiled semantic label image (000.png)

1.3 Missing Values and Outliers

The pixel distribution of our dataset reveals 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.

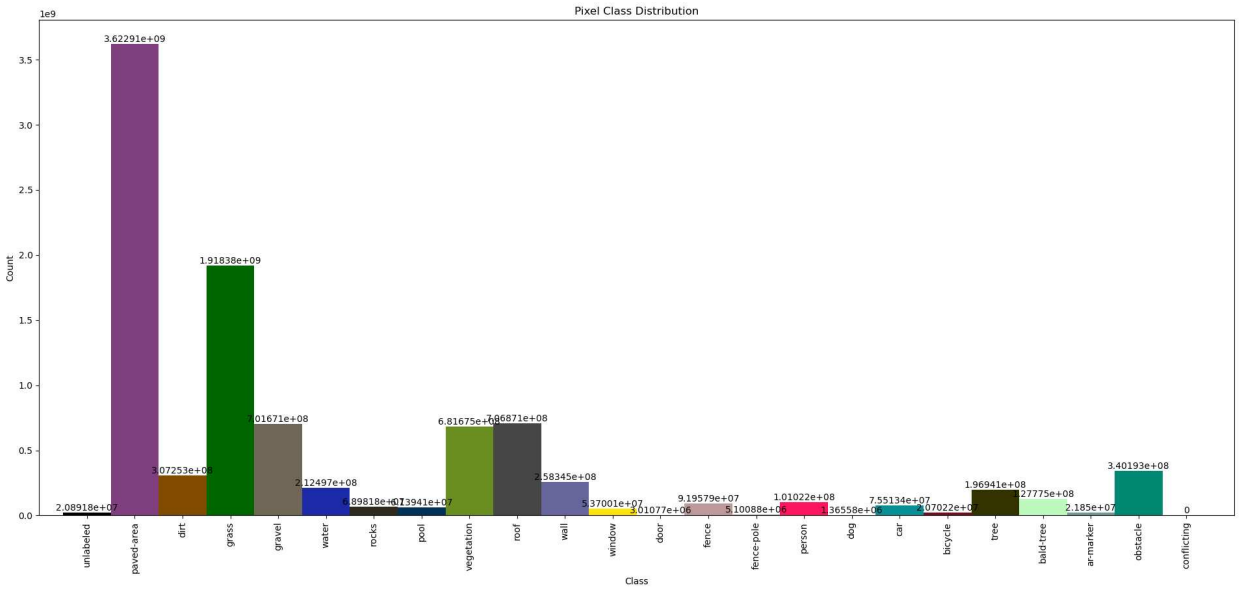

Figure 3: Pixel distribution for full dataset

1.4 Balanced vs. Imbalanced Dataset

Returning to the dataset's pixel distribution, 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.

1.5 Feature Selection

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

2. Deep Learning Models

2.1 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 4, while the VGG-19 has 16 convolution layers and 3 dense layers (Huilgol, 2022).

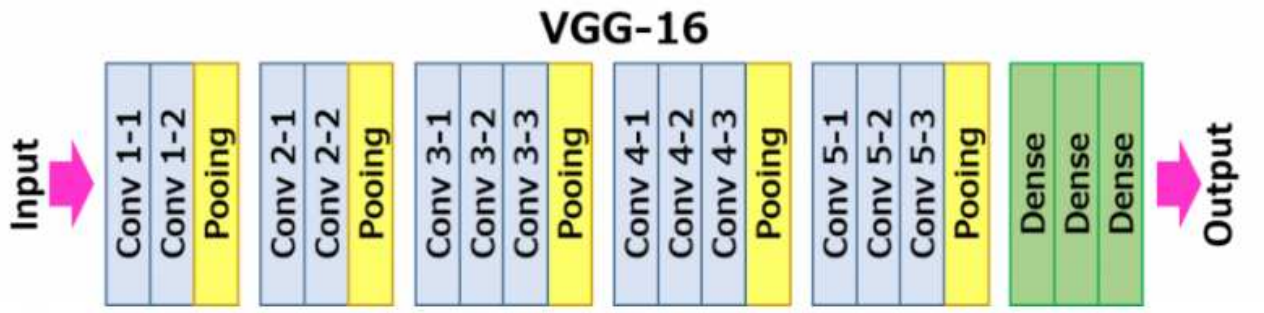

Figure 4: VGG-16 model architecture (Huilgol, 2022)

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)

2.2 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 5. 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).

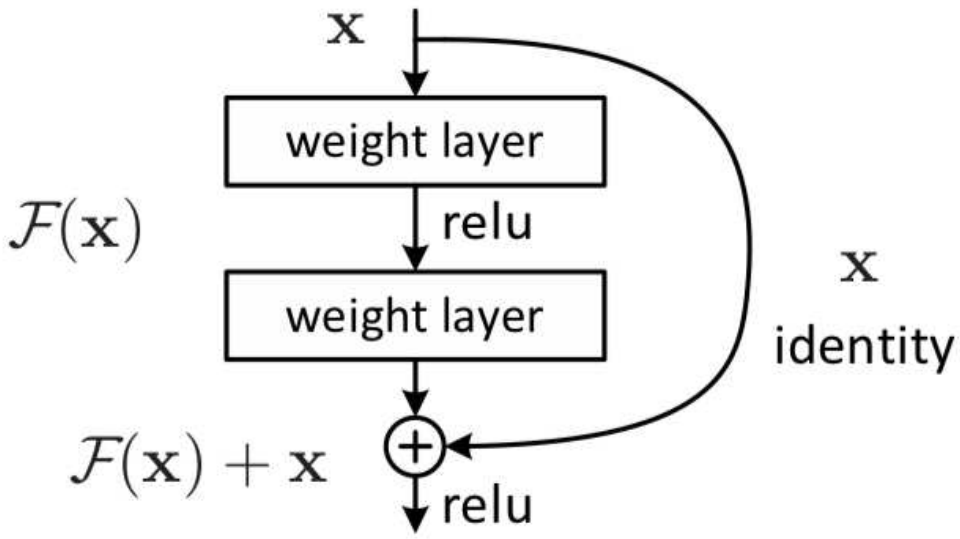

Figure 5: ResNet module architecture (He et al., 2016)

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)

2.3 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 6, and the overall model architecture seen in Figure 7.

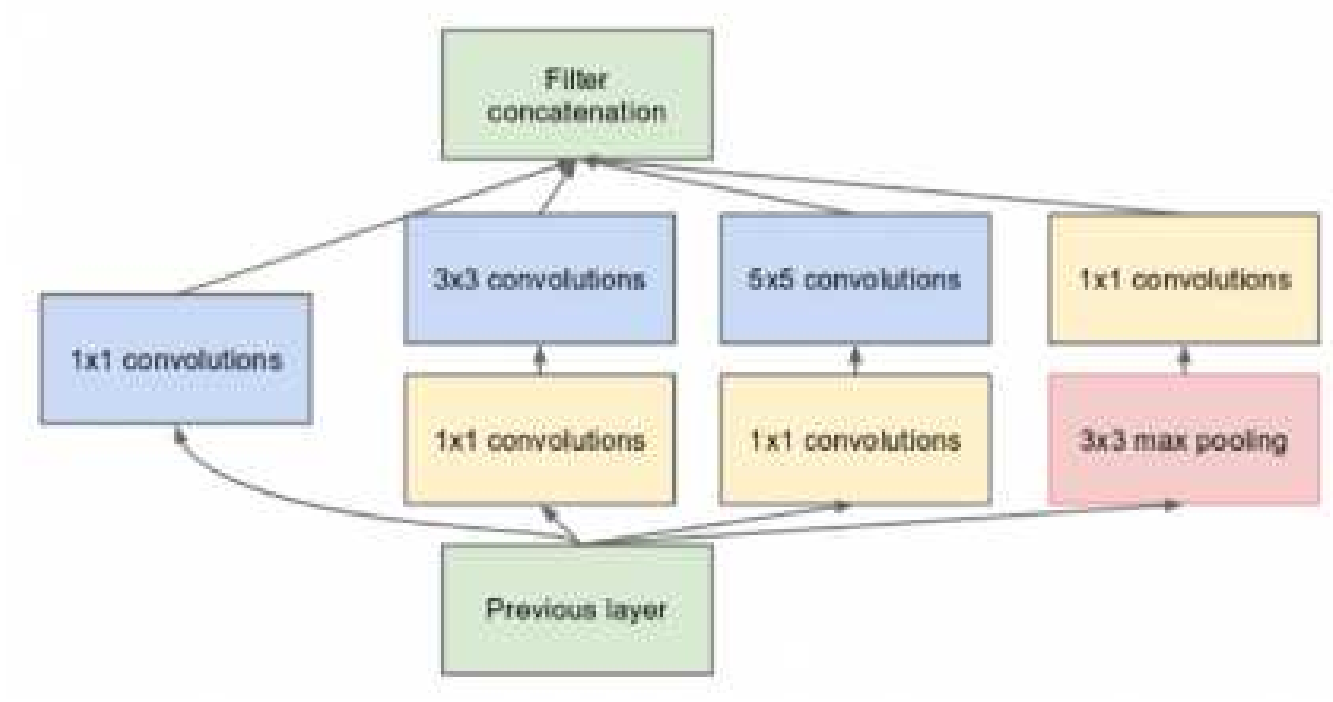

Figure 6: Inception module architecture (Szegedy et al., 2014)

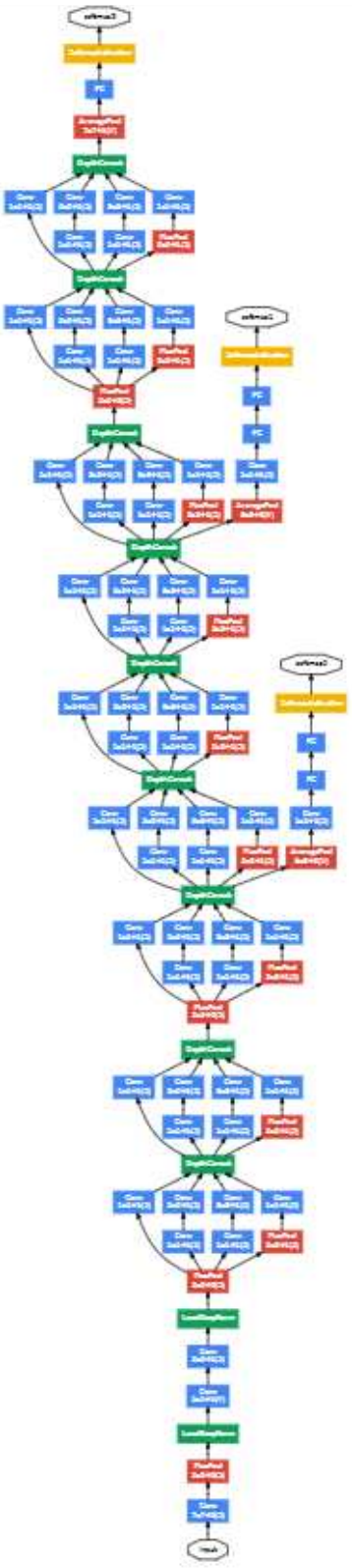

Figure 7: Inception full model architecture (Szegedy et al., 2014)

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

3. Conclusion

This report outlines the steps taken to enact pre-processing on the image dataset, as well as showcasing some of the DL models that may be used for the final implementation. For the final report, VGG, ResNet or Inception model will be used to train on the data and create a predictive program to detect objects in various images. These 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).

4. 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-7baaaeccc96

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

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

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