Multi-Class Image Classification of Satellite Imagery using CNN.

Nahom Tamene

Introduction

According to ARM, “computer vision is a field of artificial intelligence that enables computers to interpret and analyze the visual world. It applies machine learning models to identify and classify objects in digital images and videos.”

In the last decade, computer vision has proven to be one of the most useful technological advancements and its impact is prevalent throughout the industry due to how applicable it is. All around the world, it has been used for a wide range of applications; from a medical standpoint of detecting cancerous cells, to learning and identifying patterns of QR codes for purchasing items. Its application is endless, and it can only improve from here within its timeline.

Since there are many types of computer vision such as: Object Detection, Facial Recognition, and Image Segmentation, I decided to focus on one topic for this project which was Image Classification. This is the task of assigning an image to one or more predefined categories. As I mentioned earlier, its application is used for almost everything, but one that particularly interested me was Satellite Imagery. It is in our instinct as humans to learn about our surroundings; and Earth being the planet we live in, we are discovering and learning new things about it every day that helps up grow and revolutionize. Satellite image classification offers a powerful tool to analyze the Earth from a global perspective and it enables us to learn and apply knowledge by monitoring the earth, creating, and updating maps, observing cloud formations for weather forecasting, urban planning and many more. As technology continues to develop, we can expect to see even more innovative uses emerge in the future.

Classifying those images plays a big role in getting started of using the features. By developing a robust satellite image classification system, we can automate the process of extracting meaningful information from these images which in turn takes out the manual labor that would otherwise take so much time and resources to complete. The main objective of this project is to avoid this problem and to implement image classifier for seamless data analysis. By developing an accurate and efficient image classifier, it can help researchers, government agencies, and businesses to leverage the power of satellite imagery for a better future.

Approach

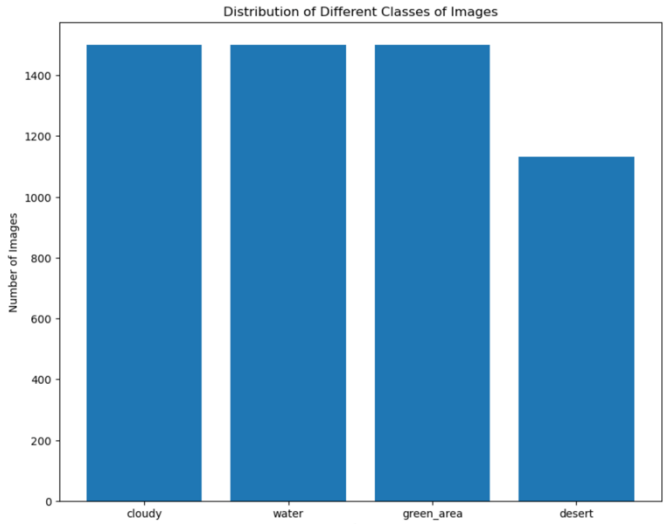
The idea was to develop a reliable model that accurately classifies satellite imagery to their respective categories. For this, I had to select a dataset with enough samples to choose from for my training and testing phases. A viable option seemed to be a dataset I found from Kaggle, which was a folder split up into 4 classes with images in them: cloudy, water, green area, and desert. These were the most common

landscapes, so the images were perfect for this project. My project aimed to classify four categories of images. While I initially envisioned using a standard image classification method, the chosen dataset's characteristics led me to pursue a multi-class image classification approach. To effectively address this multi-class problem, I needed to explore image classification techniques beyond what we discussed in class, presenting a valuable learning opportunity which I was excited about.

After I had selected my dataset, my initial project phase involved having some issues setting up the environment I would be working on. I couldn't access the data from my local computers IDE which halted my progress. I figured that Kaggle had their own IDE environment that could also access the dataset, so I started working on there. However, after some time I was getting GPU errors, and I just couldn't work on that environment for the reminder of the project. So, I did some research and found some videos and articles explaining my problem. I utilized those resources and at the end, I was able to access the data and Jupyter Notebook from my local terminal. I wanted to highlight this problem here because it was giving me a lot of trouble at the start of this project, and I believe it is important to report every phase.

My first plan of action was to parse through the data and see if I could find any that were corrupted or obscured so that it wouldn't cause me any problems later. I searched for any files that didn't end in ".jpg" or ".jpeg" which fortunately ended up on having any. This meant I didn't have any bad data so I could progress to my next planning phase.

Given the large size of the dataset, there's a potential for class imbalance, where certain classes might be significantly underrepresented. Visualizing the class distribution through a bar chart revealed some imbalance. The "desert" class contained 400 fewer images compared to the other classes. This could lead to some issues with my model down the line, so I researched some ways to fix data imbalance within image classification but after doing some research, I learned that the ratio wasn't great enough to impact the actual performance of the model, so I went on without changing anything.



This is a bar chart showing the visualization of classes (features) within the dataset I am using.

In my EDA and data preprocessing phase, I normalized pixel values because it helps in achieving faster convergence during training, stabilizing the training process, and enhancing generalization.

Model

My search for the choice of model for this project was done through extensive research and learning from online resources. There were many ways to approach this task, all with their own advantages as well as disadvantages I opted for Convolutional Neural Networks (CNNs) for this project. CNNs are particularly well-suited for image classification tasks because they can automatically extract hierarchical features directly from the images. This eliminates the need for manual feature engineering, a significant advantage compared to other methods. CNN models also excel at handling large datasets, which is a common requirement in multi-class image classification. Their architecture allows for efficient processing of vast amounts of image data, making them suitable for projects with a significant number of images and classes.

I tested different hyper parameters to optimize the model's settings. My first step was tuning and testing different filters in the convolutional layer that determine the number of feature maps extracted. I tried out using 2 and 3 convolutional layers to see which would perform better. Another parameter I found to have an impact on the model's performance was the learning rate. The learning rate is a crucial hyper parameter as it acts like a step size that determines how much the model adjusts its internal weights during the training process. The lesser the learning rate, the more time it takes to train the model and for it to take the "steps" in this process.

In this project, I investigated the impact of the learning rate on a 2-layered model. The left image depicts the model with a learning rate of 0.5. As the results indicate, this configuration doesn't achieve optimal performance. To address this, I experimented with different learning rates (0.01 and 0.0010) to identify the one that yields the most accurate model.

In contrast, the model on the right utilizes a reduced learning rate of 0.0001 and incorporates three convolutional layers instead of two. This configuration shows clearly leading to a significant improvement in performance compared to the previous model.

	precision	recall	f1-score	support
cloudy	0.67	0.98	0.80	1500
water	1.00	0.38	0.55	1131
green_area	0.94	0.60	0.73	1500
desert	0.70	0.96	0.81	1500
accuracy			0.75	5631
macro avg	0.83	0.73	0.72	5631
weighted avg	0.82	0.75	0.73	5631

	precision	recall	f1-score	support
cloudy	0.94	0.88	0.91	1500
water	0.94	0.79	0.86	1500
green_area	0.84	0.96	0.90	1500
desert	0.88	0.97	0.92	1131
accuracy			0.89	5631
macro avg	0.90	0.90	0.90	5631
weighted avg	0.90	0.89	0.89	5631

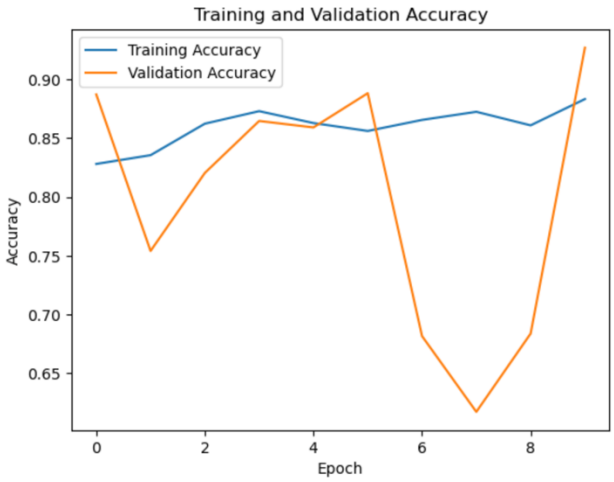
Results

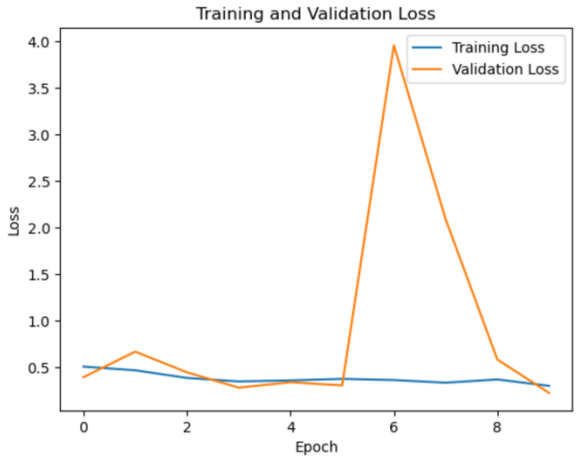
The final result for this project took some time for me to complete because the model's performance required careful attention to detail. I iteratively explored feature techniques to prevent overfitting and experimented with multiple learning rates and convolutional layer configurations. After iteratively tuning the features for a while and improving the overall performance of the model, I was glad to see the accuracy of the model being high. However, that isn't the real measure of performance since I still needed to see what the precision, recall and f-1 score were as they are true indicators. It wasn't optimal initially, but by refining various aspects of the model, I was able to achieve significant improvement in these crucial metrics.

The most surprising lesson I have learned from this project was how long it took for the model to be trained in each epoch. It would take at least 3 minutes for interval which created some limitation when it came to CPU usage and time constraints.

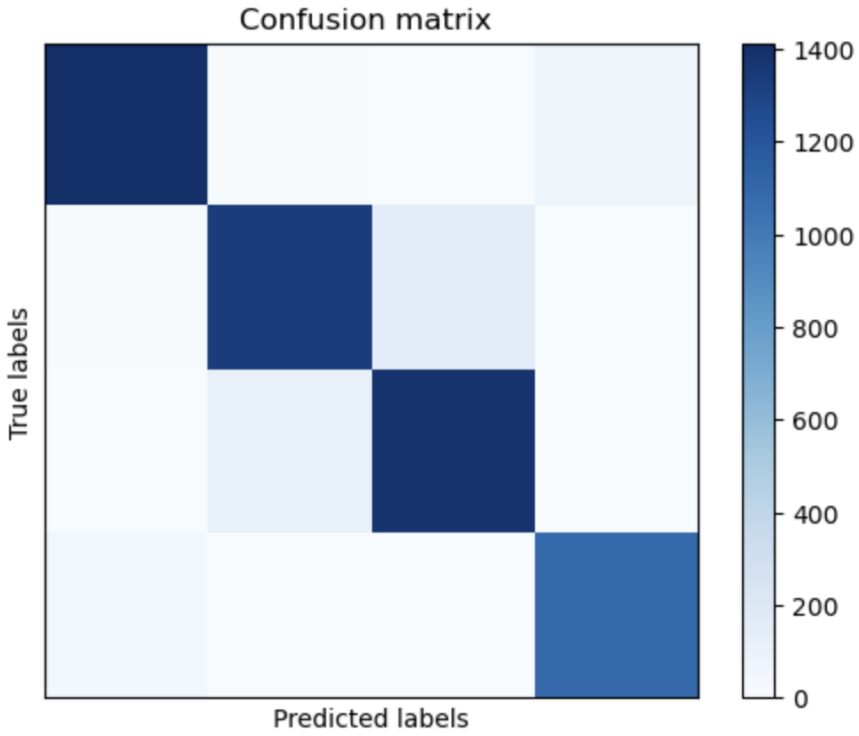
The next two images show the relationship between the training and validation values during this whole process and as they gradually become more optimal by decreasing loss and increasing accuracy. This graph shows how the model changed overtime and how it learns from the training data after every epoch (iteration). But there's a sudden surge of decrease in accuracy which I am not sure caused it. The accuracy value is still displayed as high but there's a decrease for a certain interval before it gets better again.

As I have already listed the precision, recall and f-1 score earlier, I won't display them again, but they also show how the model progresses overtime when the hyper parameters are changed.





The last performance evaluation I did was adding a confusion matrix to visualize the predictions of the model against the true class labels for a set of test data. Through this, we can see which classes the model performs well on and doesn't. The model underperformed than what I expected against the desert class, and it made some errors for the False Positive labels but overall, still a good result.



Conclusion

To summarize everything I have done, I think implementing a CNN model was successful for this project. I was able to collect data from Kaggle where I found some interesting insights to build an image classifier. Fortunately, I was able to get the desired result after iterative editing and tuning.

There is still room for improvement; however, for instance I don't know if there would be a big difference as the value got smaller, but I would have liked to test more different values of the learning rate to see if we could have achieved even better results. Additionally, I could work on the model to take in more classes in the dataset to have a broader model than can be used for better generalization. I am still happy with our result and this project served as a great learning experience.

Acknowledgments: www.arm.com/glossary/computer-vision

https://www.youtube.com/watch?v=jztwpsIzEGc&t=1994s

https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

https://washcollreview.com/wp-content/uploads/2020/10/multi-class-image-classification-using-machine-learning.pdf