Labeling Land:

Land Use Classification Using Convolutional Neural Networks

Claire Miles

July 2019

Introduction

Satellite imagery can help us find solutions to the growing number of environmental problems that humans face today. It allows us to not only get a bird's eye view of what's around us, but also uncovers parts of the world that are rarely seen. Tapping into the potential of categorizing land cover and land use around the world means that humans can more efficiently make use of natural resources, hopefully lowering cases of waste and deprivation. But despite its potential to be incredibly useful, satellite data is massive and confusing, and making sense of it requires complex analysis.

Inquiry: Can I create a model that can classify the landscape/land use of a given image?

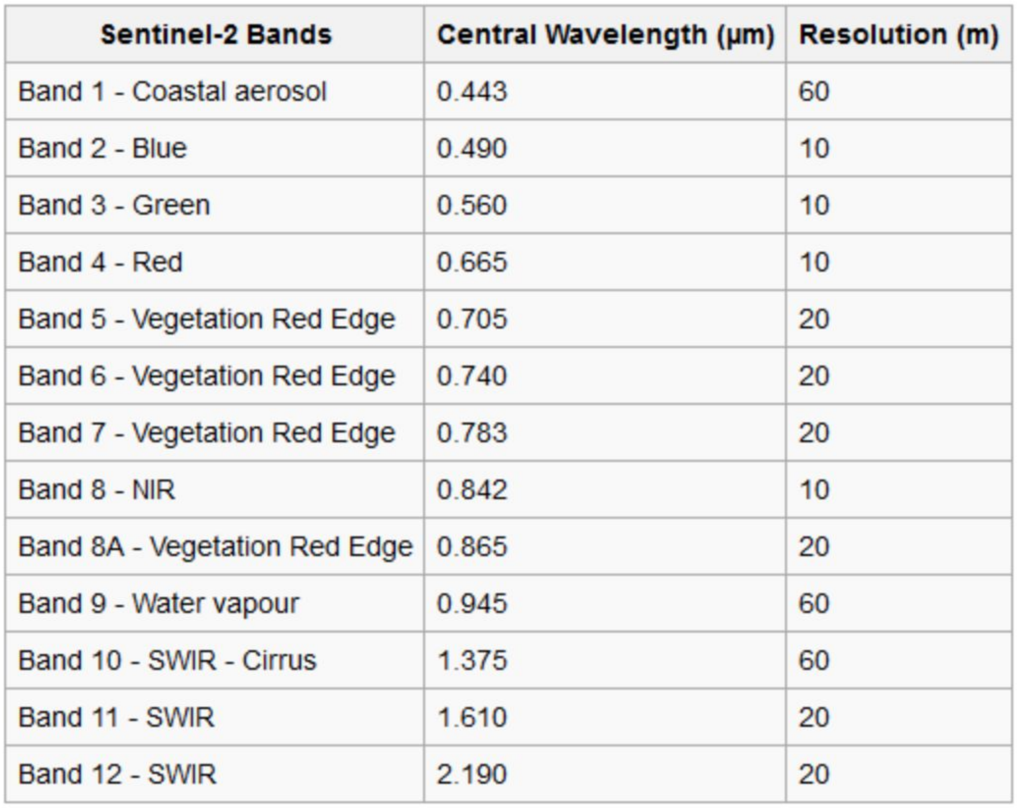
This project could help a variety of stakeholders, including conservationists, urban planners, and environmental scientists, survey and identify patterns in land use to see which natural areas are under threat or which areas are best for urban development. In this case, I will tailor this project to an environmental imagery tech company. Imagery companies can use land use classification models to categorize what's in each image and optimize their efforts towards the parts of land that are important to them. For example, an agriculture company will only want to be concerned with land that's labelled as pasture, annual crop, or permanent crop. If a satellite is constantly taking images, this project would help save hours of time manually sorting through imagery.

In addition to sorting imagery, land use classification is important for identifying the parts of an image to which certain analyses are applied. For example, if the company has different crop stress algorithms for fields that are next to urban areas versus fields that are next to rivers, this project would help them to automate the process of applying these specialized algorithms and more effectively solve environmental problems.

For this project, I'm using open source EuroSAT Sentinel-2 satellite images from the German Research Center for Artificial Intelligence, which can be downloaded locally [link]. The dataset consists of 27,000 labeled images of 10 different land use classes:

1. Annual Crop
2. Forest
3. Herbaceous Vegetation
4. Highway
5. Industrial
6. Pasture
7. Permanent Crop
8. Residential
9. River
10. Sea / Lake

Each multispectral image consists of 13 different color bands that represent different wavelengths of light/color and different resolutions (Figure 1). These different light bands help distinguish parts of the landscape that reflect certain types of light in particular ways. Since most images don't include special bands like Vegetation Red Edge, Coastal aerosol, or SWIR, for this project I chose to only use the red, green, and blue bands in an effort to make my model generalizable to most images.



Sentinel-2 Bands	Central Wavelength (µm)	Resolution (m)
Band 1 - Coastal aerosol	0.443	60
Band 2 - Blue	0.490	10
Band 3 - Green	0.560	10
Band 4 - Red	0.665	10
Band 5 - Vegetation Red Edge	0.705	20
Band 6 - Vegetation Red Edge	0.740	20
Band 7 - Vegetation Red Edge	0.783	20
Band 8 - NIR	0.842	10
Band 8A - Vegetation Red Edge	0.865	20
Band 9 - Water vapour	0.945	60
Band 10 - SWIR - Cirrus	1.375	60
Band 11 - SWIR	1.610	20
Band 12 - SWIR	2.190	20

Sentinel-2 Bands, Wavelength, and Resolution

Figure 1. The bands available in the images of the dataset used for this project.

Data Preparation

Creating Folders: In order to be compatible with the keras function *flow_from_directory*, I created train, validation, and test set folders for the data. Within each folder, the data was separated into more folders by category.

Translating File Type: Compatibility with keras/tensorflow also required me to translate the files from tif to jpeg format. I found that this was easiest using the *gdal_translate* function in the command line, which I accessed within the notebook using the *!* notation.

When translating, I selected bands 2, 3, and 4, which are the red, green, and blue bands. Choosing the RGB bands makes for a traditional image, rather than a specialized satellite image which may have extra near-infrared, red-edge, or short-wave infrared bands. This makes the model more accessible to the average, everyday image of the outdoors.

After translating a file, I moved it to the training data folder. When all files of a certain category were trained and moved, I separated the jpeg files into the validation and test data folders with a train:validation:test separation of 80:10:10.

Exploratory Data Analysis

Before building the deep learning model, it's worth it to get more familiar with the dataset. Especially with an image dataset, visualizations help us see differences in the different image categories as well as the red, green, and blue bands of a single image.

Photos by Category: The bar chart shows the number of photos in each category in the dataset, which range from about 2500 photos in the Industrial and River categories to about 6000 in the AnnualCrop, SeaLake, HerbaceousVegetation, and Residential Categories (Figure 2). Imbalances in representation could affect the efficacy of the model, but I will leave the dataset as is for now.

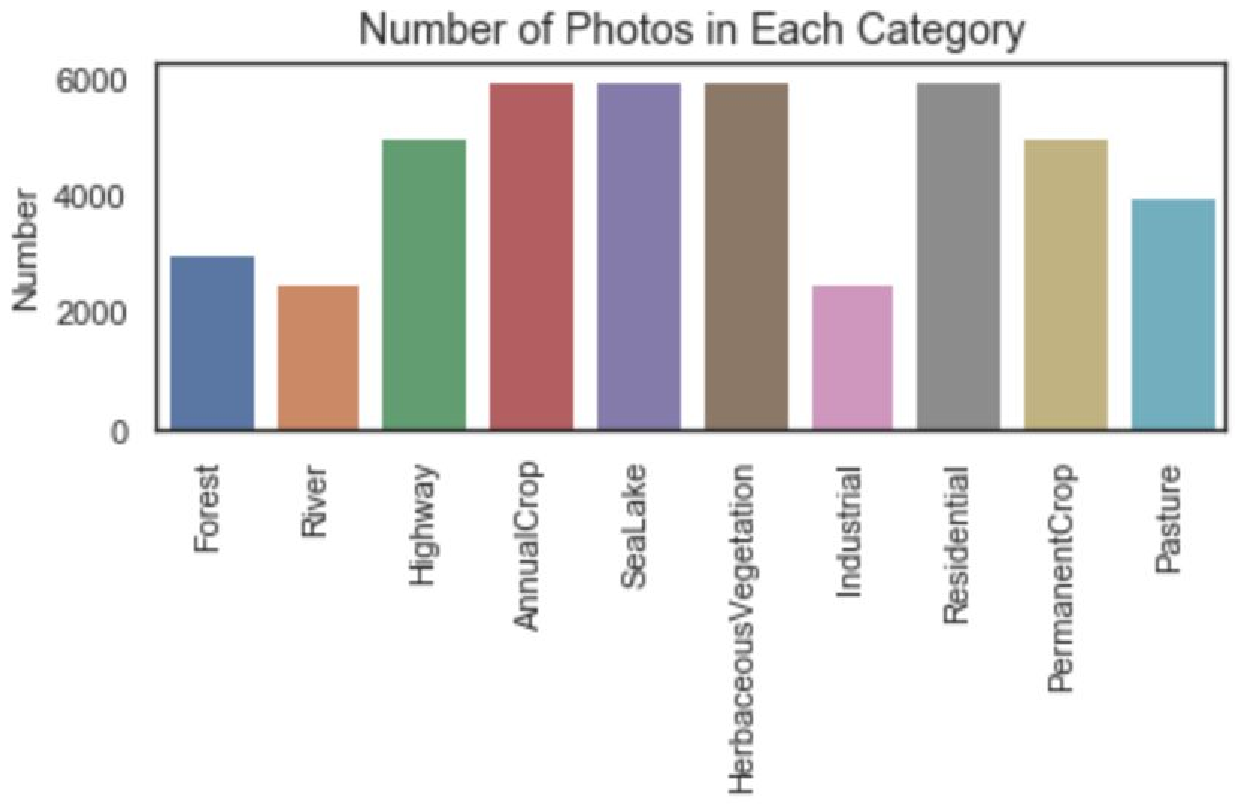


Figure 2. Bar plot of the number of images in each category of the dataset.

The RGB representation of a photo from each category provides a better sense of what the images actually look like (Figure 3). Though they are relatively low quality, it's still pretty easy to see the difference between what is denoted as cropland, residential, highway, etc.

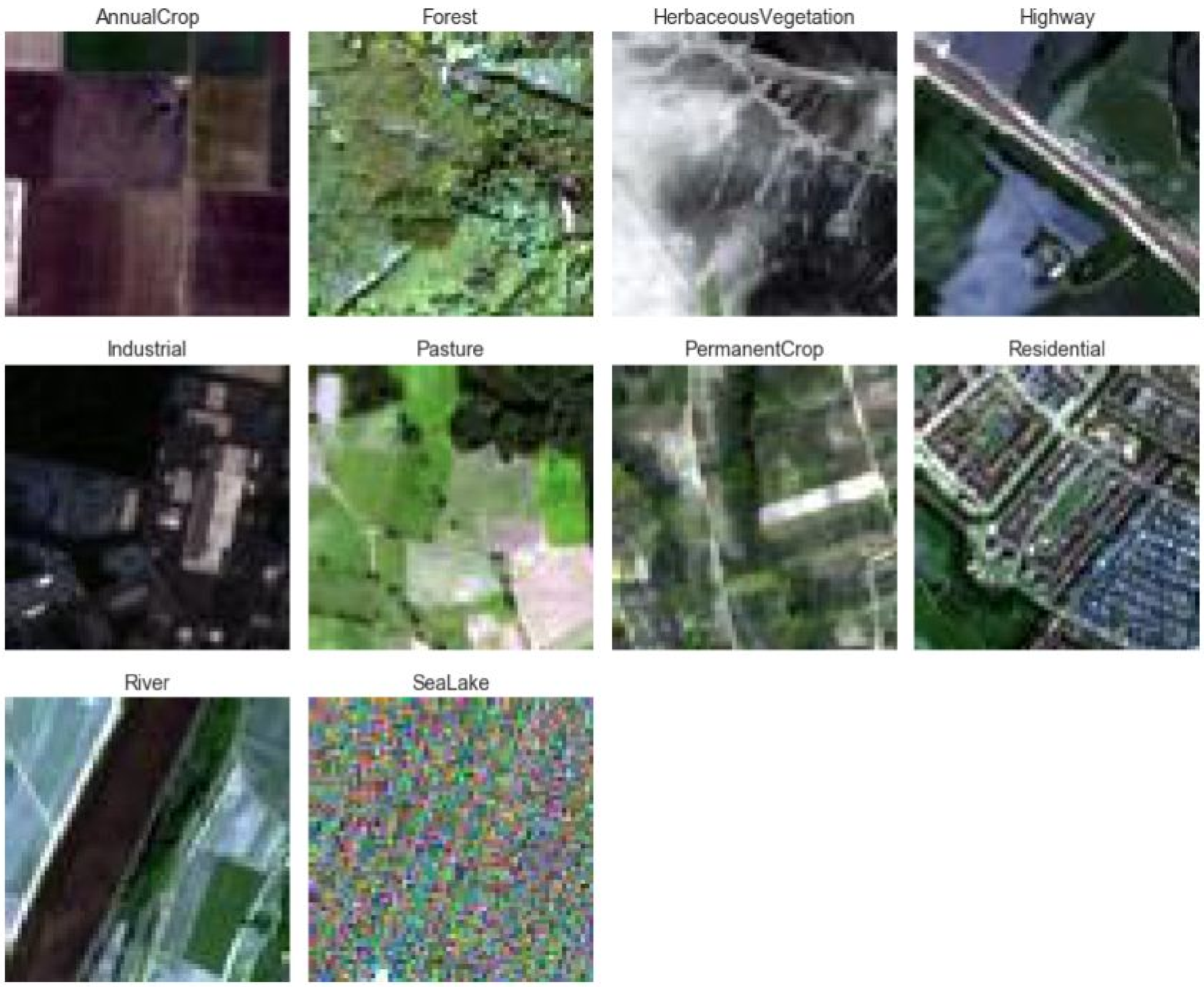


Figure 3. One image from each category in the dataset.

Photos by Color Band: The three color bands within each photo can also provide more information about the satellite data. The package rasterio, has helpful tools to extract information about the images that will be useful when constructing the convolutional neural network.

For example, the height and width of the images is 64 pixels, and there are 3 bands in each image. This makes sense, since I extracted three bands in our translation of the images from tif to jpeg. The RGB bands can be corroborated by the realistic look of the sample images from each category.

Plotting each band doesn't tell too much about the makeup of the data, but it is proof that there are three different bands in each image (Figure 4). However, stacking the three bands results in a RGB composite photo with realistic looking colors (Figure 5).

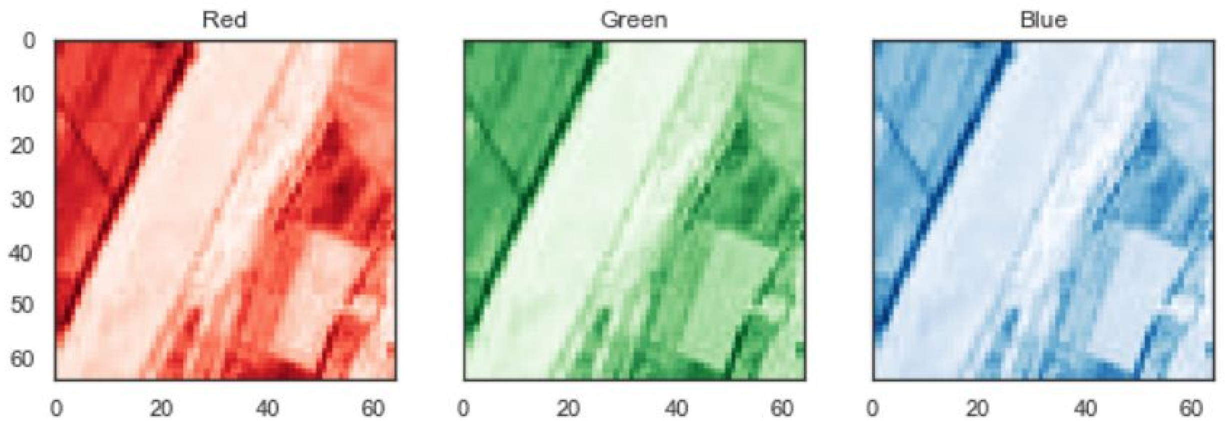


Figure 4. The red, green, and blue bands of a single image in the dataset.

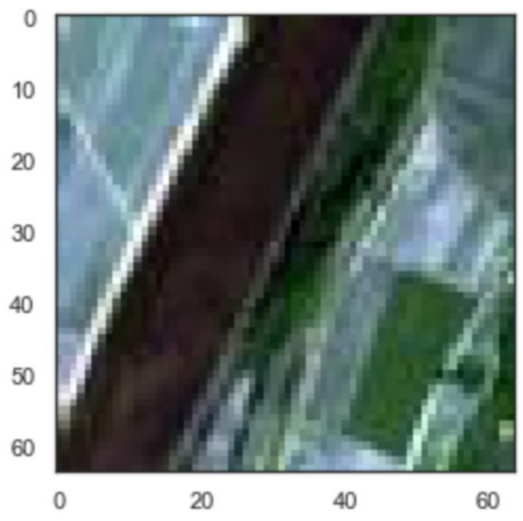


Figure 5. The same image when the red, green, and blue bands are stacked, creating a realistic look.

Plotting the wave frequencies of each band shows that there are differences in the pixel representation for each one (Figure 6). These differences could come in handy as the neural network learns the differences between image features in each category.

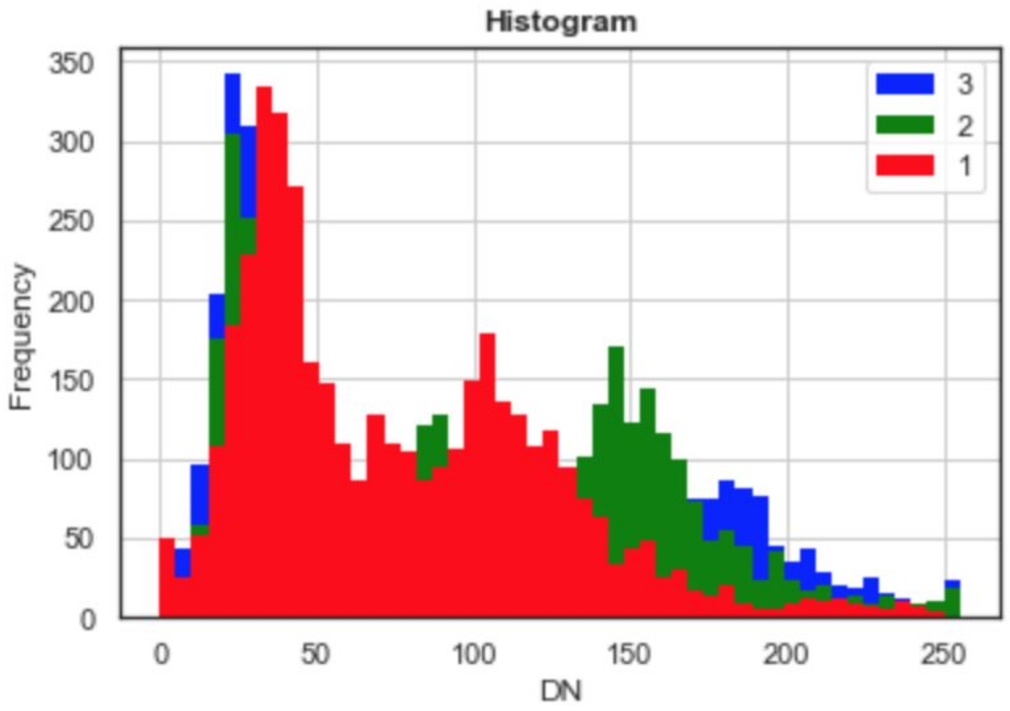


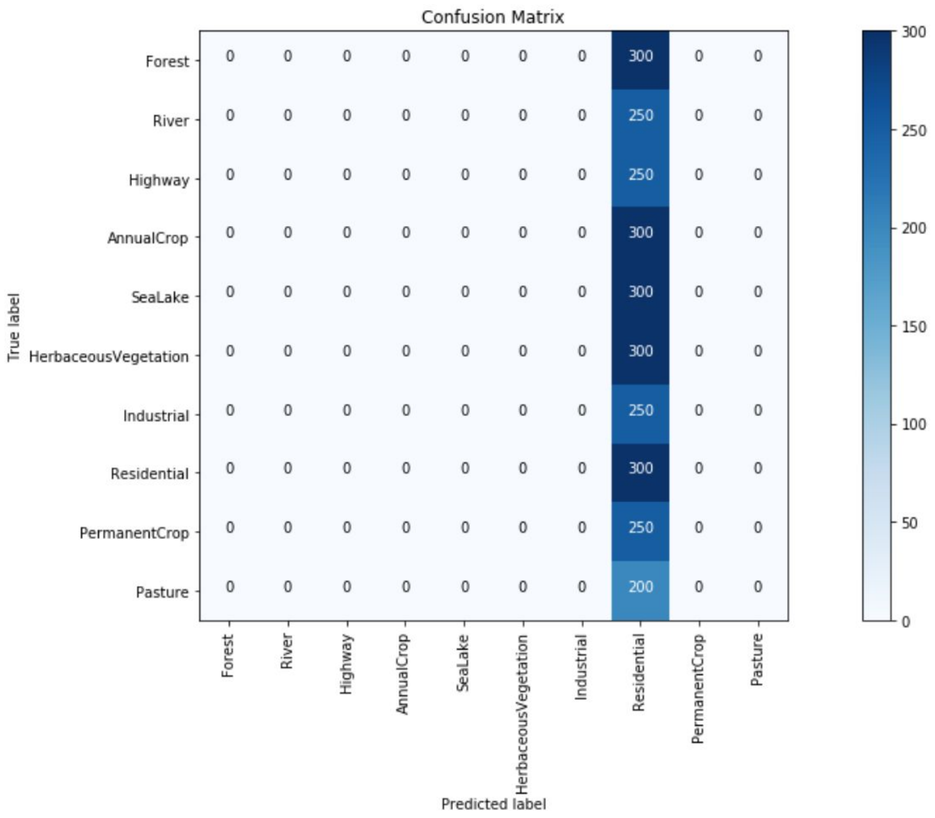
Figure 6. Histograms of the wave frequencies of the red, green and blue color bands of one image in the dataset.

Basic Convolutional Neural Network (CNN)

The first convolutional neural network explored in this project was a simple CNN with a three layer architecture: one convolutional input layer, one flatten layer in the middle, and one dense output layer. The model was compiled using the Adam optimizer, which is a stochastic optimization function that's computationally efficient.

The simplicity of the first model is a relatively fast test that the data preparation was sufficient and that the neural network was set up with the right parameters. However, it's not very good for model performance - since satellite data is pretty complex the model will require more layers in order for it to learn the distinct features of each category of the dataset. This was evident once the model was trained - the accuracy stayed constant at 11% for all five training epochs.

The confusion matrix of the model's performance on the test data is more evidence that this model is way too simple. The CNN must not have learned much, because it guessed every test image as the same category (Figure 7). No wonder the accuracy was around 10%, which is what one would expect if they were randomly guessing for a dataset with 10 categories.



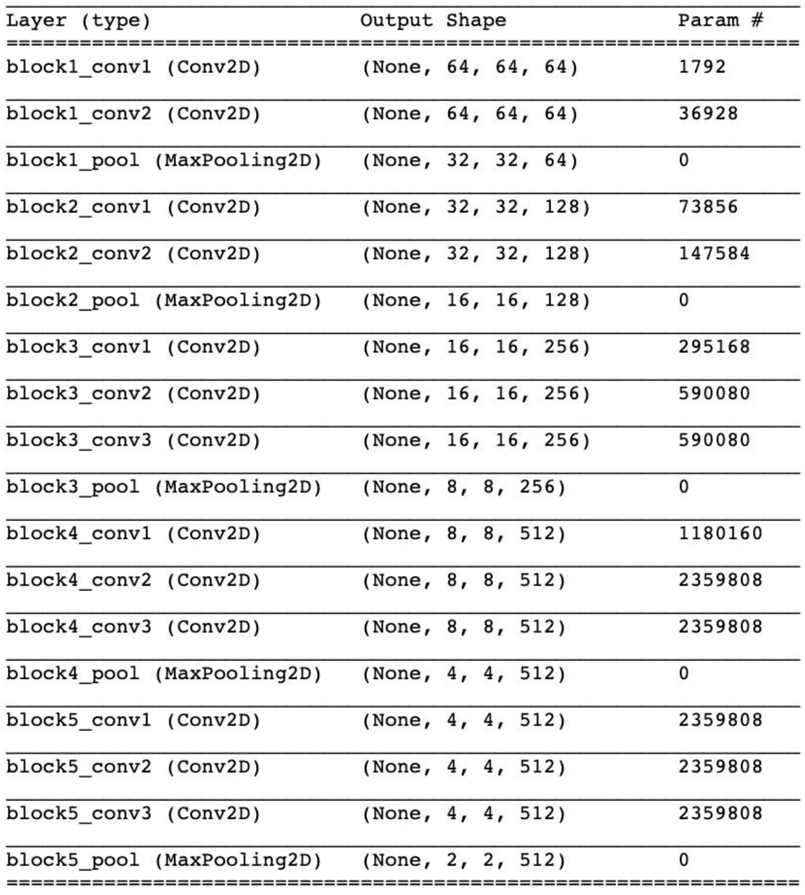
True label \ Predicted label	Forest	River	Highway	AnnualCrop	SeaLake	HerbaceousVegetation	Industrial	Residential	PermanentCrop	Pasture
Forest	0	0	0	0	0	0	0	300	0	0
River	0	0	0	0	0	0	0	250	0	0
Highway	0	0	0	0	0	0	0	250	0	0
AnnualCrop	0	0	0	0	0	0	0	300	0	0
SeaLake	0	0	0	0	0	0	0	300	0	0
HerbaceousVegetation	0	0	0	0	0	0	0	300	0	0
Industrial	0	0	0	0	0	0	0	250	0	0
Residential	0	0	0	0	0	0	0	300	0	0
PermanentCrop	0	0	0	0	0	0	0	250	0	0
Pasture	0	0	0	0	0	0	0	200	0	0

Figure 7. The confusion matrix shows what the true labels were for images in the test dataset and what the model predicted them to be. In the case of the simple model, every image was predicted in the "Residential" category, performing no better than a random guess.

Transfer Learning

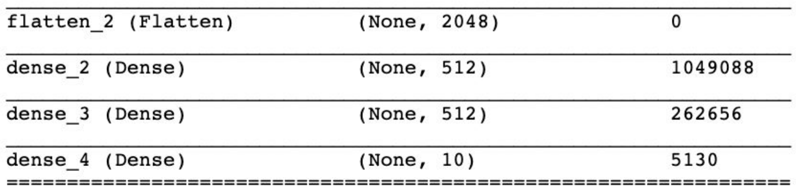
Since the simple model was not very good, and there was not a ton of computing power available for this project, the next step was to use transfer learning to improve the performance of this model on the dataset. Transfer learning let's one adapt a large pretrained model to their own project. In this case, the ImageNet VGG16 model was used. The VGG16 model is a CNN that was trained on 15 million images in over 22 thousand categories. While the first several layers of the model will be kept in their pretrained state, I will train the last set of layers to learn the specific features of this dataset, which only has a few thousand images and ten categories.

The VGG16 model architecture is an input layer followed by a pattern of two or three convolutional layers and a max pooling layer that repeats five times (Figure 8). I changed the model to match the input size of this project's data, as well as the output size of ten categories. I froze the first 12 layers so that only the final 10 would be trained (Figure 9).



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0
=================================================================
```

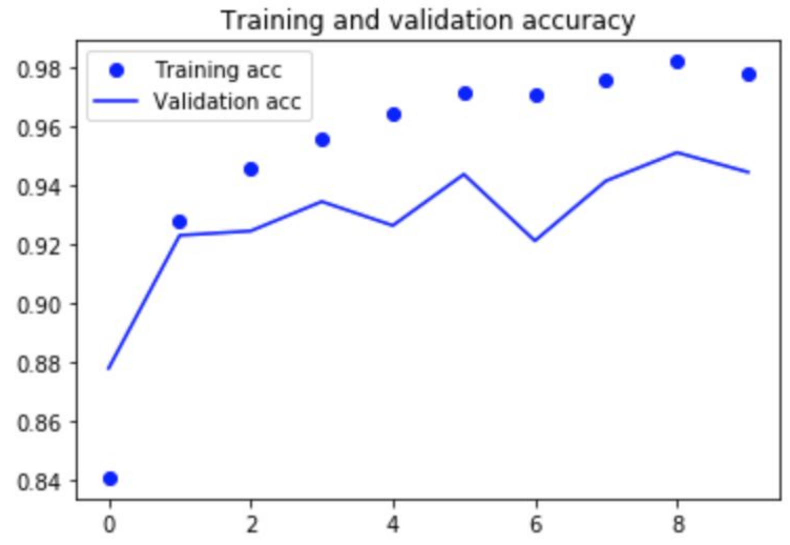
Figure 8. The architecture of the VGG16 model.



```
_________________________________________________________________
flatten_2 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               1049088
_________________________________________________________________
dense_3 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_4 (Dense)              (None, 10)                5130
=================================================================
```

Figure 9. The layers added to the end of the transfer model to fit the project context.

Even though this model took over a day to run for ten epochs, it out-performed the simple CNN by an impressive amount. In fact, this model achieved 94% validation set accuracy, guessing most categories in the test set correctly (Figure 10). The confusion matrix shows that while most images were categorized correctly by the model, it had the most trouble miscategorizing pastures as forests and rivers as highways (Figure 11). The model also tends to miscategorize adjacent categories, a phenomenon that may have to do with the CNN algorithm or the chosen parameters and something that would be explored given more time and resources for this project.



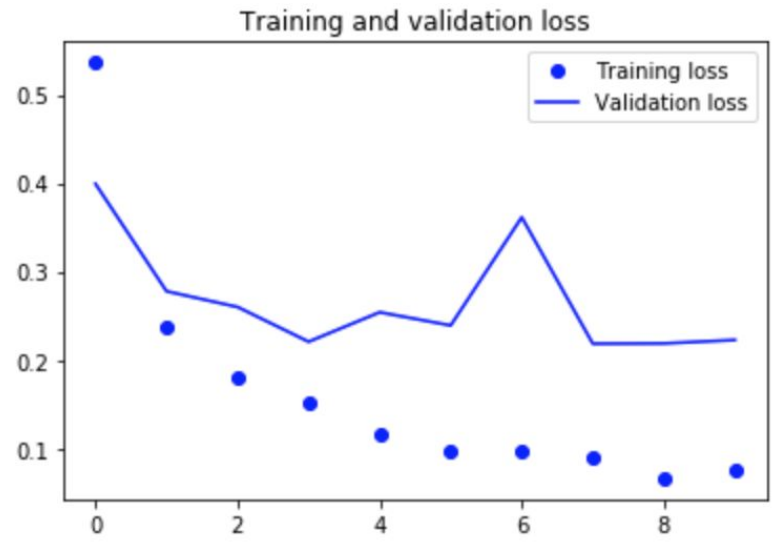


Figure 10. The training and validation accuracy (top) and loss (bottom) of the transfer learning model.

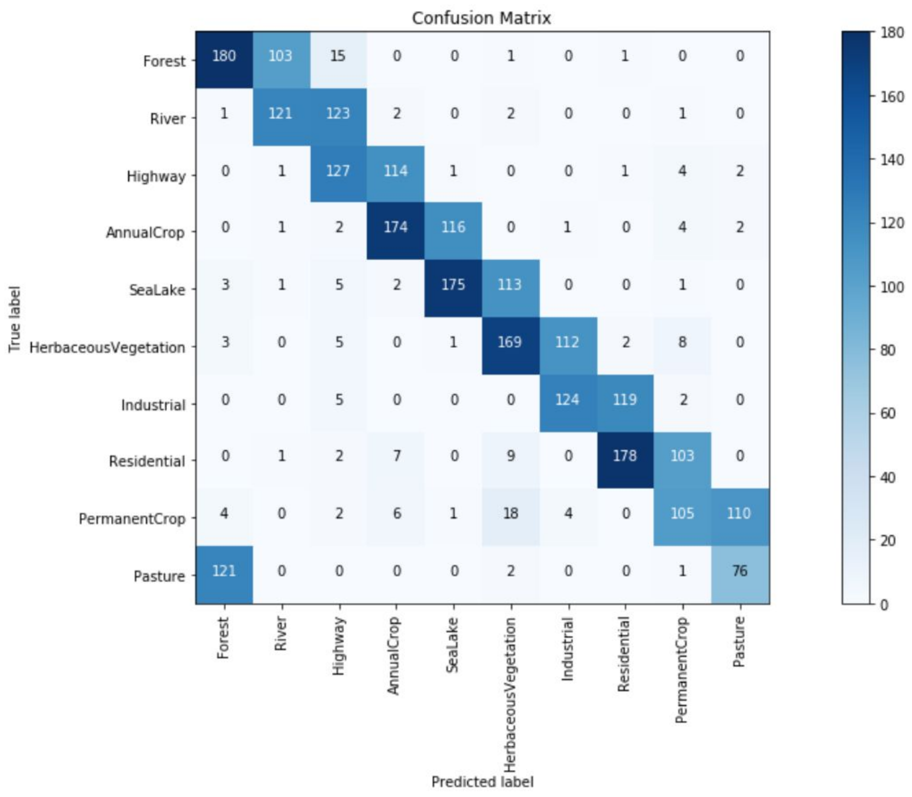


Figure 11. Confusion matrix for the transfer learning model, which achieved a much higher accuracy than the simple CNN.

Using one image as an example, I can visualize the ways in which the model is making sense of the features in the image to come to its conclusions. With a river image as the input, I see how specific parts of the image become highlighted in each convolutional layer, and how they are simplified into lower resolution images in the pooling layers. As the layers progress, certain neurons stop firing and turn black as the network becomes increasingly complex (Figure 12).

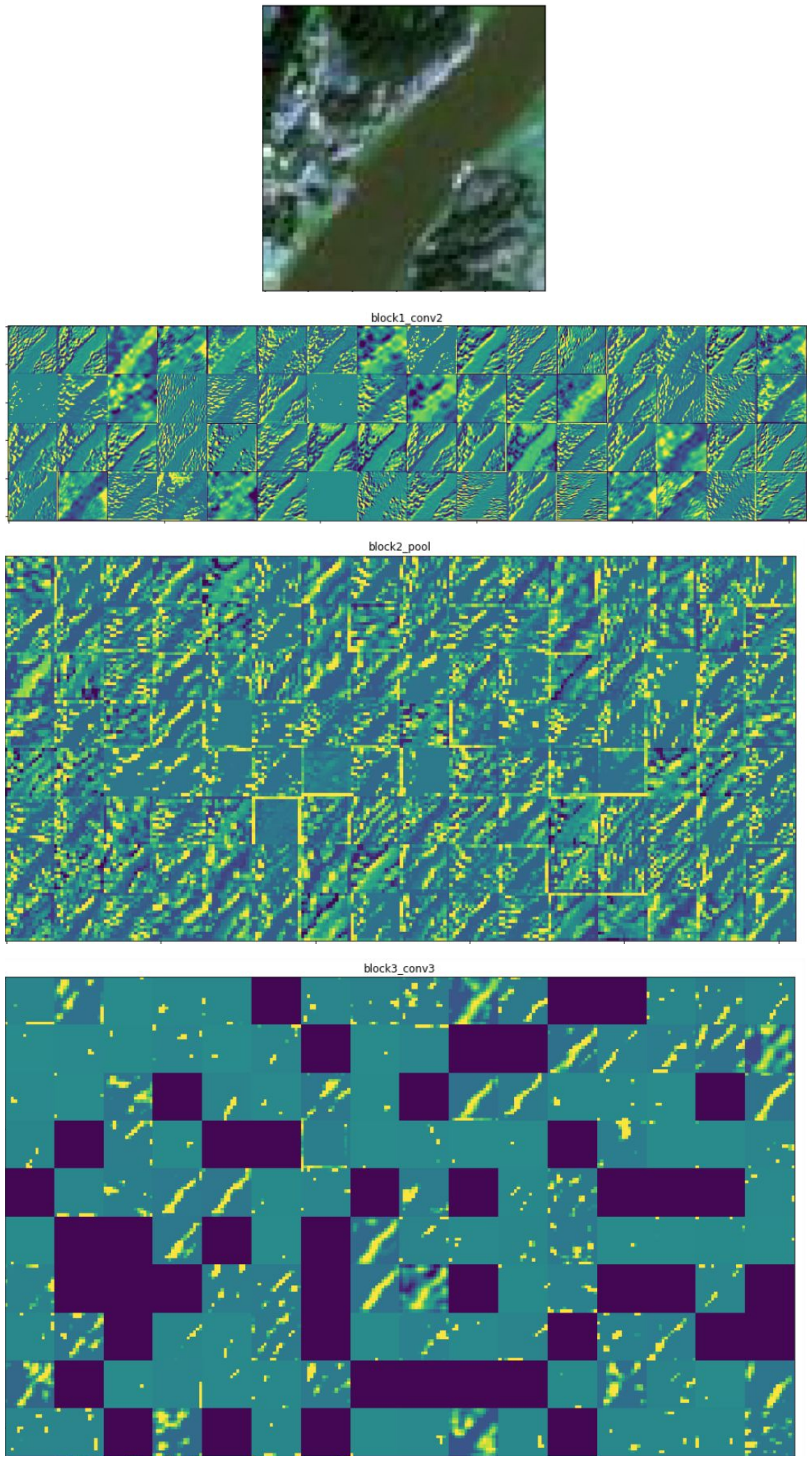


Figure 12. The input image (top) and the interpretations of the image at three different layers in the model.

Model Tuning

Even though the transfer learning model performed much better than the simple three-layer model, there is still room for improvement. Among the several ways to improve a model's performance, this project will explore data augmentation and dropout regularization.

Data augmentation increases the size of the dataset by creating multiple versions of the same image that are slightly transformed. For example, the original image in Figure 13 has been turned into ten different images that have been flipped, shifted, and zoomed in or out. Despite this seemingly small change, the slight transformation poses a large computational challenge to the neural network, and it becomes more accurate by learning to categorize many different versions of the same thing.

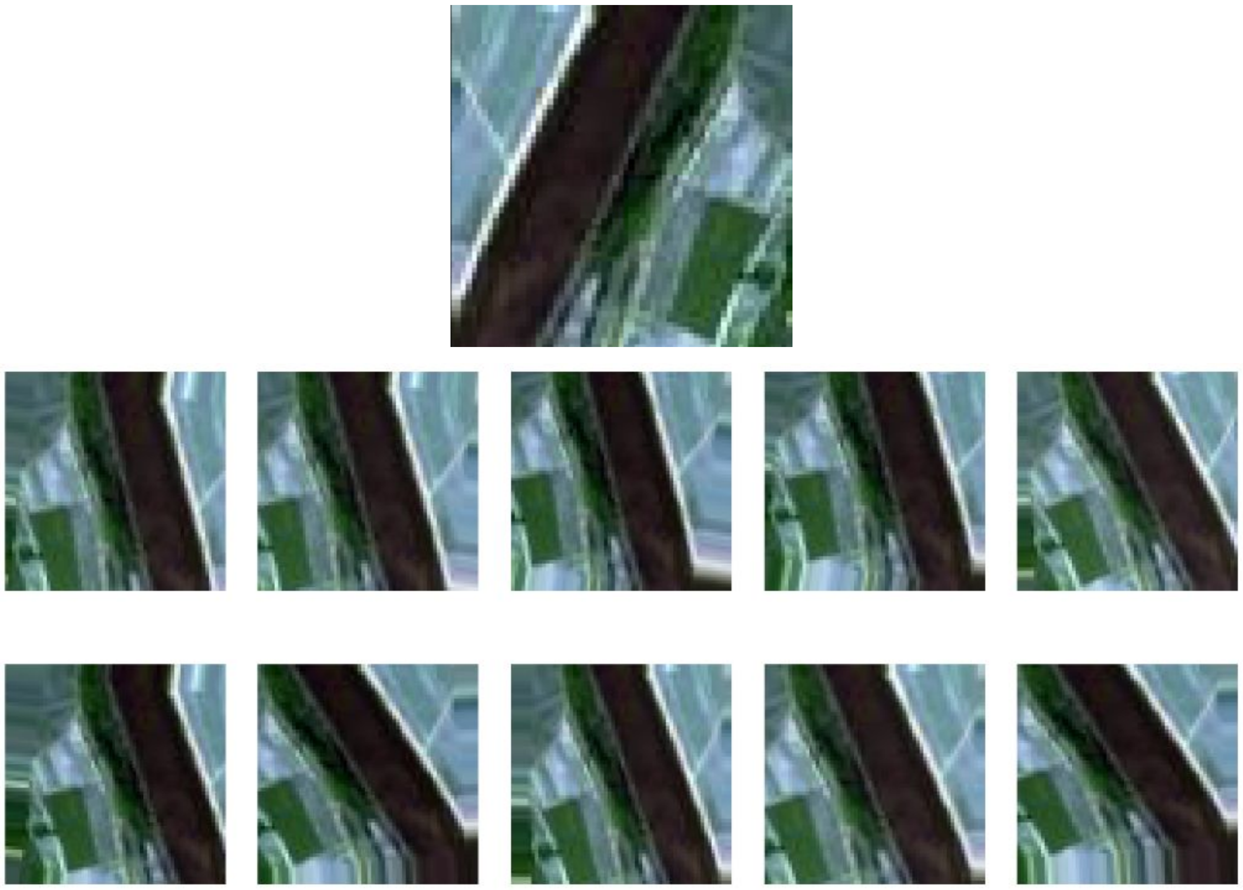
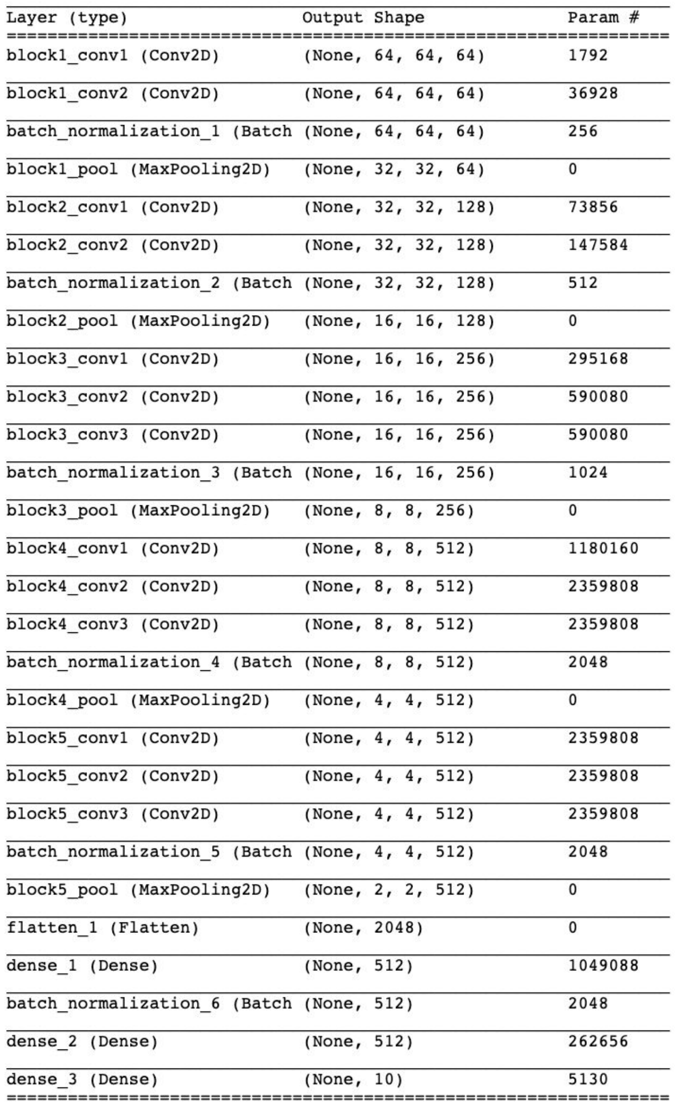


Figure 13. An image from the original dataset (top) and ten different versions of the same image below. These slightly augmented images enrich the diversity of the dataset and improve the neural network.

Batch normalization improves the speed and performance of a neural network by normalizing the data to a standard scale in the model's learning process. Normalization reduces the influence of unusually large values in the data which would otherwise unequally offset other parts of the network, like the weights in each layer. To incorporate batch normalization into the model, I included normalization layers before every pooling layer, and also before the last two dense layers (Figure 14).

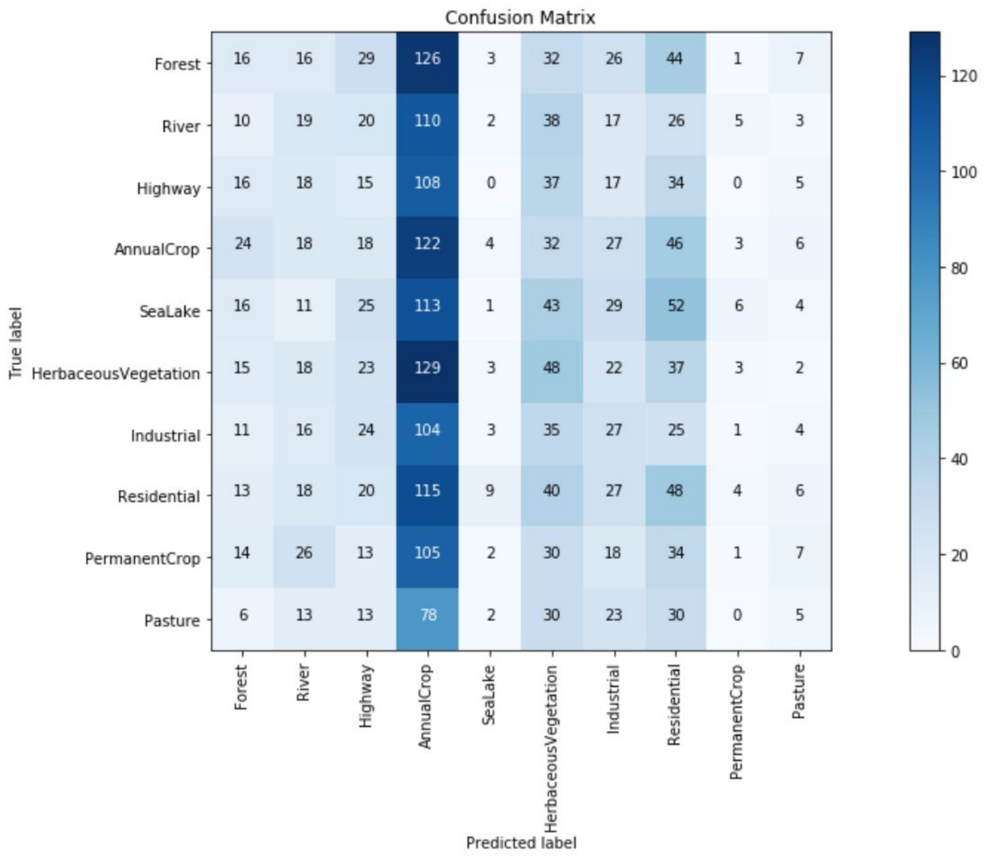


```
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928
_________________________________________________________________
batch_normalization_1 (Batch (None, 64, 64, 64)        256
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 128)       512
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 256)       1024
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 8, 512)         2048
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
batch_normalization_5 (Batch (None, 4, 4, 512)         2048
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1049088
_________________________________________________________________
batch_normalization_6 (Batch (None, 512)               2048
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
```

Figure 14. Architecture of the CNN with batch normalization layers added.

With ten epochs of training, the tuned model achieved a 56% accuracy. Batch normalization is known to speed up model training, allowing for larger learning rates and fewer epochs. In terms of execution time, this is true for this project - the tuned model went through ten epochs in 13 hours while the previous model took 34 hours for the same number of epochs. Despite the difference in running time, the tuned model's accuracy is continuing to improve up until the last epoch, and would likely continue improving if given more epochs to train. For the purposes of this project, I kept all model parameters the same, but I might have seen the tuned model learn much faster, and reap the benefits of batch normalization, if I had increased the learning rate to something larger than 0.0001.

The performance of this model results in a confusion matrix somewhere between that of the first two models in the project (Figure 15). The tuned model clearly hasn't learned enough yet to counteract the "guessing" effect, where the majority of images are predicted to be a single category. However, there is more diversity in the predictions as there were for the simple model, and the accuracy is well above 10%.



	Forest	River	Highway	AnnualCrop	SeaLake	HerbaceousVegetation	Industrial	Residential	PermanentCrop	Pasture
Forest	16	16	29	126	3	32	26	44	1	7
River	10	19	20	110	2	38	17	26	5	3
Highway	16	18	15	108	0	37	17	34	0	5
AnnualCrop	24	18	18	122	4	32	27	46	3	6
SeaLake	16	11	25	113	1	43	29	52	6	4
HerbaceousVegetation	15	18	23	129	3	48	22	37	3	2
Industrial	11	16	24	104	3	35	27	25	1	4
Residential	13	18	20	115	9	40	27	48	4	6
PermanentCrop	14	26	13	105	2	30	18	34	1	7
Pasture	6	13	13	78	2	30	23	30	0	5

Figure 15. The confusion matrix for the tuned model's performance on the test dataset.

Conclusions & Next Steps

This land use classification model could be an important addition to the workflow at most businesses that use satellite data. For example, a company that uses satellite images to predict the number of customers parked outside of malls and other commercial buildings will only want to be concerned with the images of parking lots. Therefore, they will need to sort out the images of interest from the wide array of images that a satellite takes as it travels across the earth. Having to do this manually is tedious and expensive. Even if a company paid one employee fifteen dollars an hour to sort through images for forty hours per week, that totals to over thirty-thousand dollars spent per year on the task that this image classification model could do instead. For a larger company handling more data, this cost would be even higher.

Another valuable aspect to integrating machine learning models into workflows is the opportunity cost. While the models handle lower-level tasks such as image classification, employees can spend their time on more complex, intellectual pursuits such as research and development, company growth, and human-centered design. Letting employees focus on projects that will improve the company rather than simply maintain its functioning increases the company's value over time.

Realistically speaking, working to create a good model that can work within certain resource limits, such as computing power and time, can be even more valuable than creating an optimally-performing model that uses too many resources. In the case of this project, the first transfer learning model took less time to converge than the second one, meaning that it may be the better choice if the client has less time and money. Since this project addresses an environmental question and most environmental entities are in the public, nonprofit, or university sectors, resources and realism is a very important consideration. Even for small startups that have limited funding and need to prove their worth before the next funding round, taking advantage of limited resources is a big factor in deciding which models to use in production.

Future Explorations: There are many ways to improve the performance of this model that were not explored in this project. In addition to, or instead of, using batch normalization as a tuning technique, other types of regularization and tuning like weight initialization and dropout could be explored.

Dropout regularization works to prevent the neural network from overfitting on the training data, which increases its performance on the test set. When a dropout layer is included in the architecture of a neural network, it probabilistically removes a certain number of inputs to the next layer. This action simulates the creation of many different neural networks with varying architectures because different nodes are deactivated each time a dropout layer is reached during the learning process. This results in a neural network that is more robust to different kinds of inputs and performs better on the test dataset. Adding dropout layers, however, would not fit into the context of this project because it slows down learning, requiring many more learning epochs, more time, and more computer power.

Changing the weight initialization parameters can help a neural network learn faster by combating the vanishing or exploding gradient problem. When weights are randomly initialized, there is a greater chance that the variance of the weights is large and will slow down the training process. Training a model whose initialized weights purposefully have smaller variance, like with Xavier or He initialization, helps the model to converge faster and more efficiently. Keras actually uses a uniform Xavier weight initialization by default, but it would be worth it to explore others initialization methods with both uniform and normal distributions.

Another aspect of this project to explore would be the base model used in transfer learning. In this project, I used the popular ImageNet VGG16 model, but there are many others to choose from. Another popular model for transfer learning is MobileNet, which was built to work well on web browsers and on mobile devices. MobileNet models take up much less memory and have fewer parameters then ImageNet models, making them more flexible for use on computers, tablets, and smartphones. In exchange for this flexibility, MobileNet is not as accurate as its larger counterparts, but still performs rather well. Using a MobileNet model as the base for this experiment would be useful if the model were to be used on a smartphone for real time land use classification using the phone's camera, for example.

Finally, I can observe the strength of the model by testing its ability to generalize to a completely different dataset from another source. There are other similar satellite datasets out there that could be used as a test set for this model. If the model performed well on this outside data, it would signify that the model learned enough features of each category to generalize effectively. If not, I would know that the model could benefit from having additional data included in the training process.