Capstone Project 2 | Milestone Report 2
Claire Miles

Project Subject: Land Use/Land Cover Classification

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 a 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 1). No wonder the accuracy was around 10%, which is what one would expect if they were randomly guessing for a dataset with 10 categories.

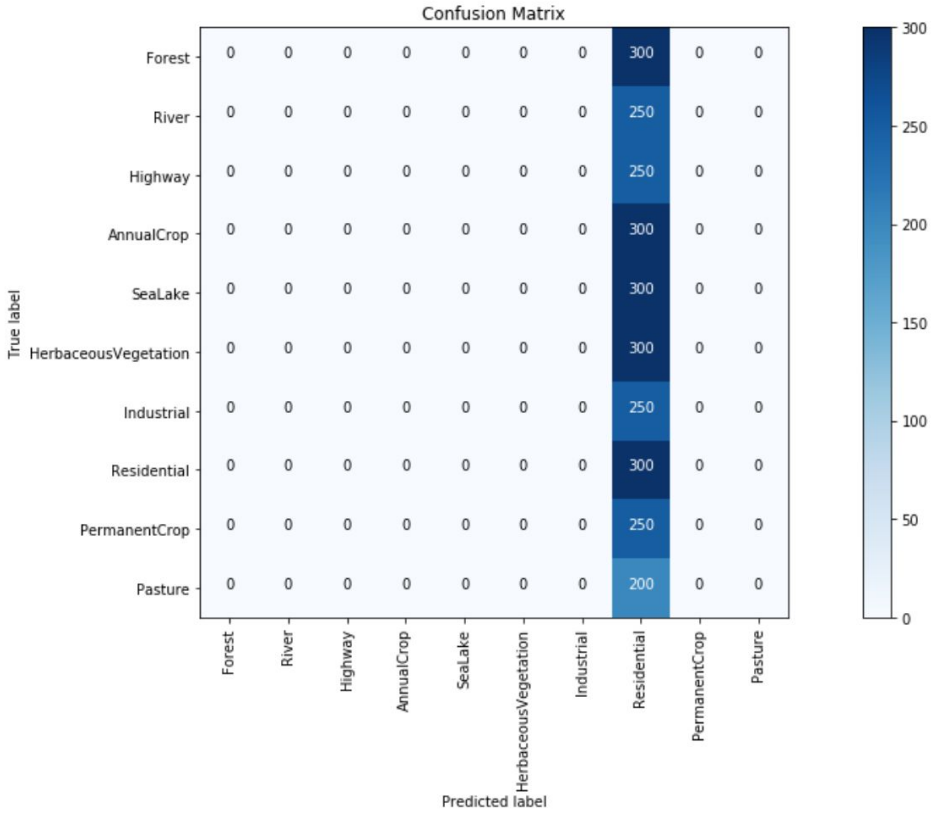


Figure 1. The confusion matrix shows what the true labels were for images in the test dataset, and what the model predicted them to be. In the case of the simple model, every image was predicted in the "Residential" category, performing no better than a random guess.

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 2). 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 14 layers so that only the final 10 would be trained for this project (Figure 3).

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 2. 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 3. The layers added to the end of the transferred model to fit the project dataset.

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 4). 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 5).

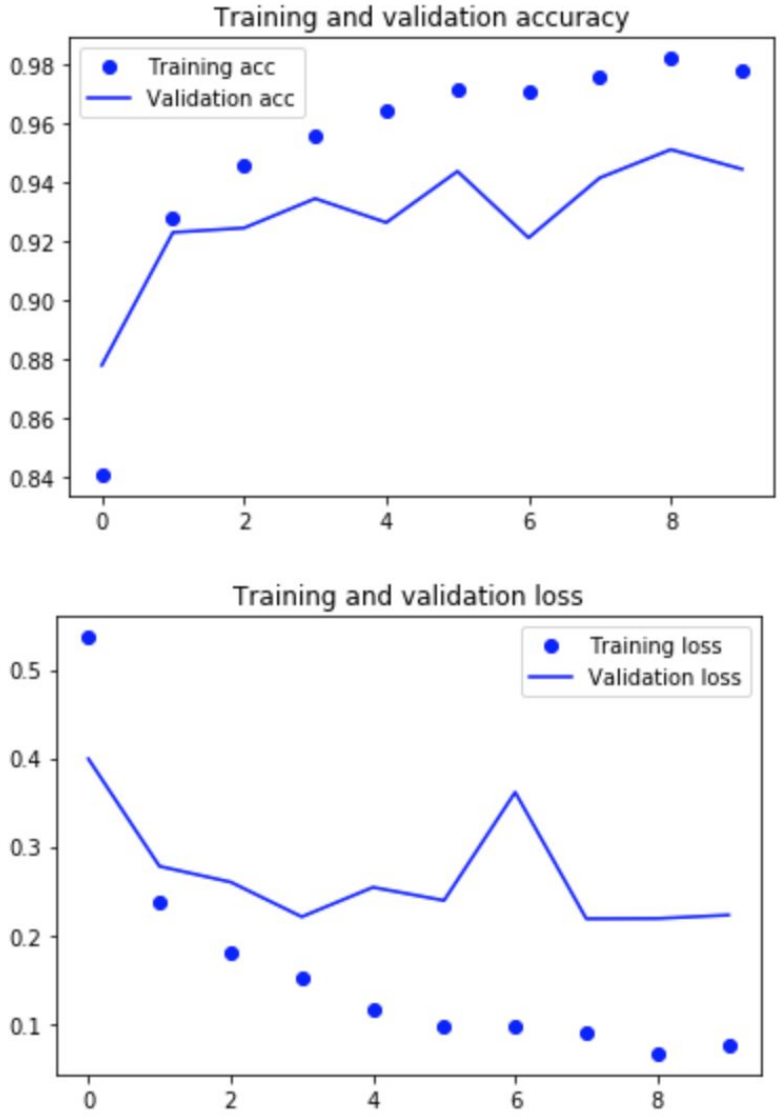


Figure 4. The training and validation accuracy (top) and loss (bottom) of the transfer learning model.

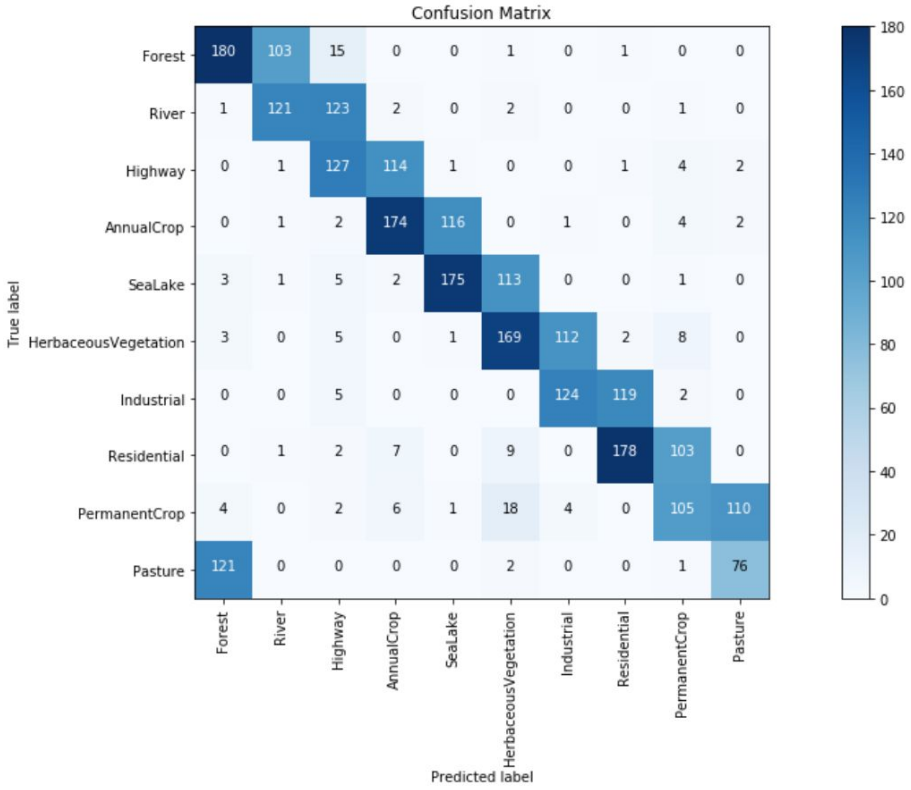
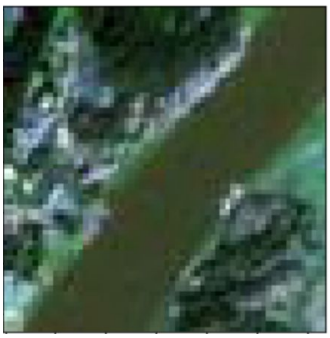


Figure 5. Confusion matrix for the transfer learning model, which achieved a much higher accuracy than the simple CNN.

Using one image as an example, we 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, we 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 6).



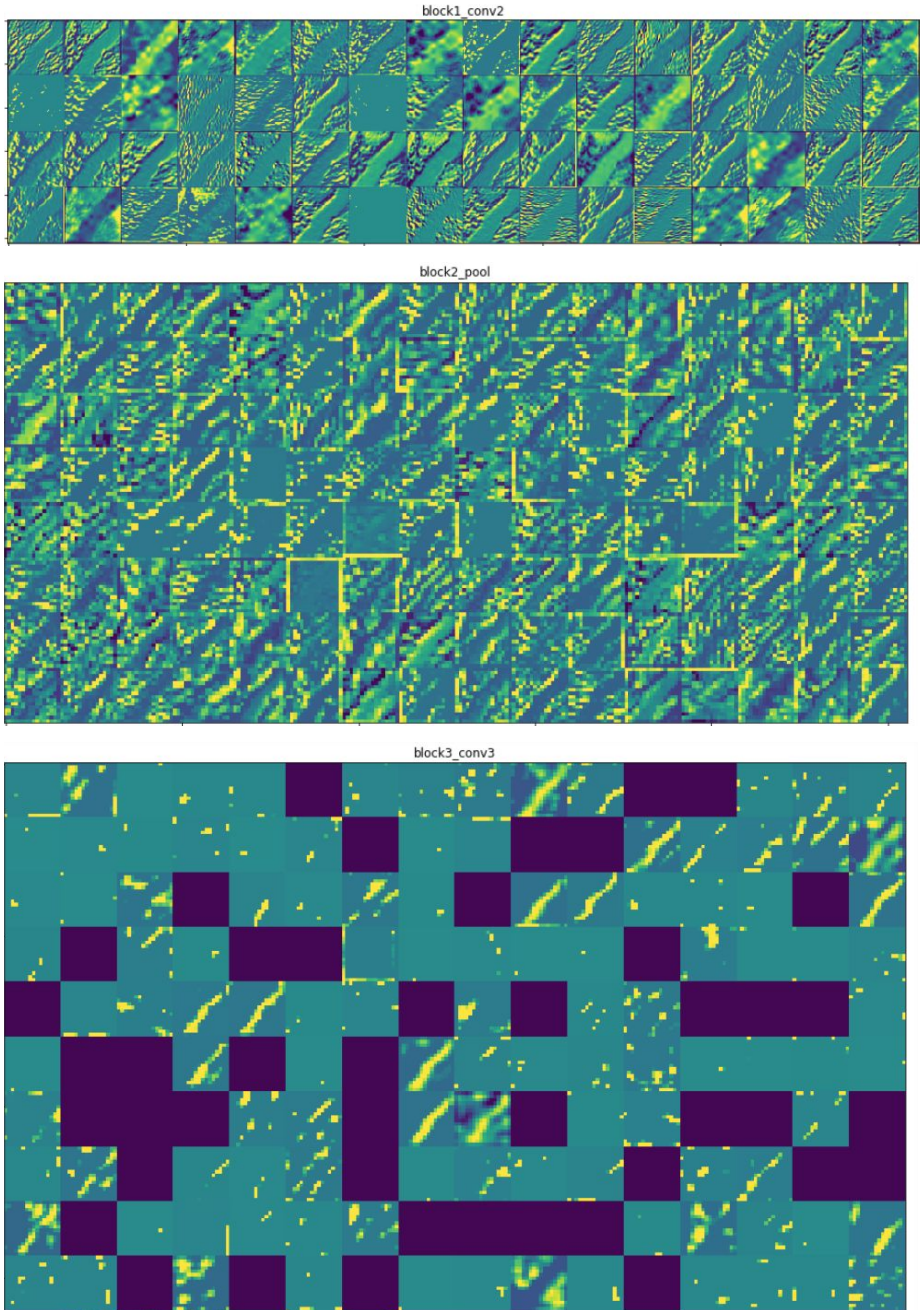


Figure 6. The input image (top) and the interpretations of the image at three different layers in the model.

Model Tuning

Even though the transfer learning model performed much better than the simple three-layer model, there is still much room for improvement. Among the several ways to improve a model's performance, this project will employ 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 7 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.

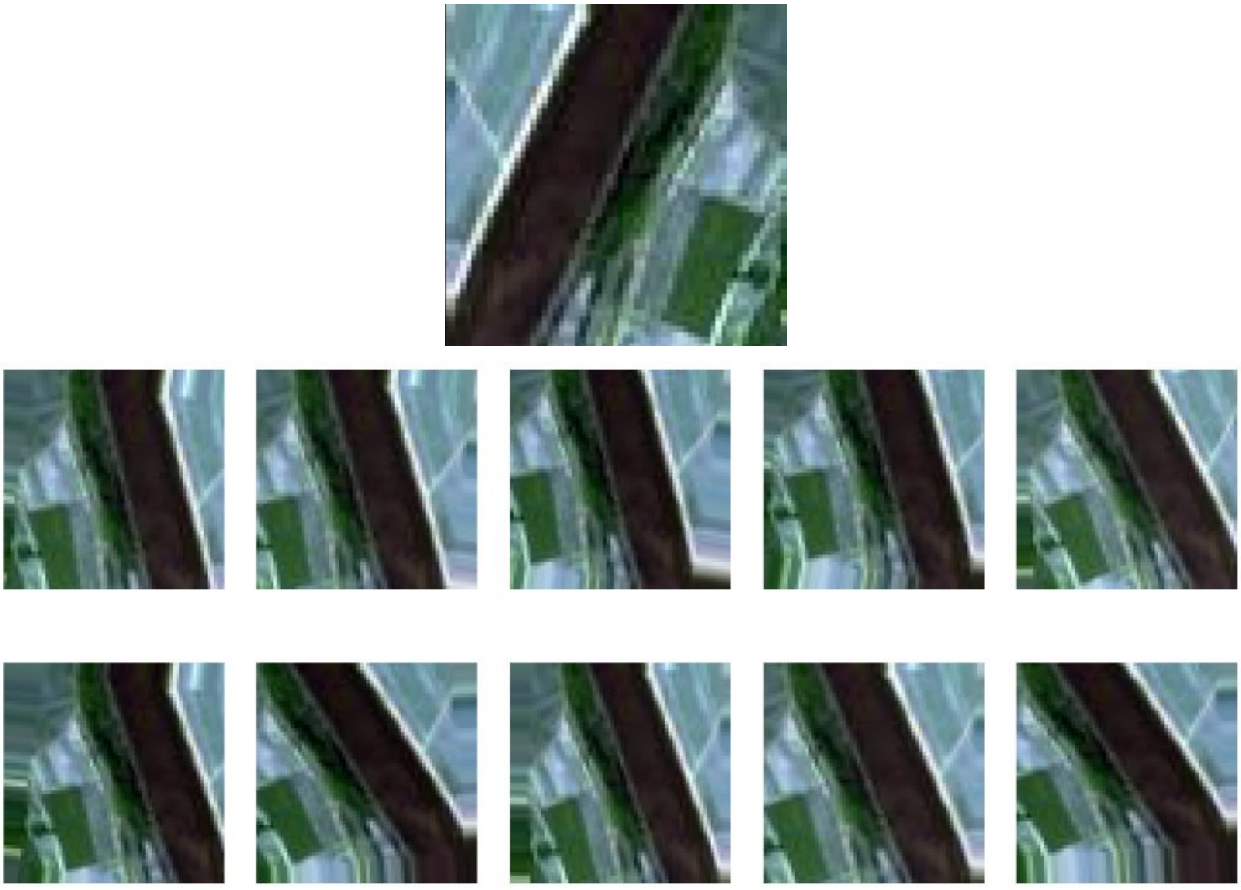


Figure 7. An image from the original dataset (top) and ten different versions of the same image below. These slightly augmented images enrich the diversity of the dataset and improve the neural network.

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 on the model, 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 8).

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 8. 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 other 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 results of this model result in a confusion matrix somewhere between that of the first two models in the project (Figure 9). 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%.

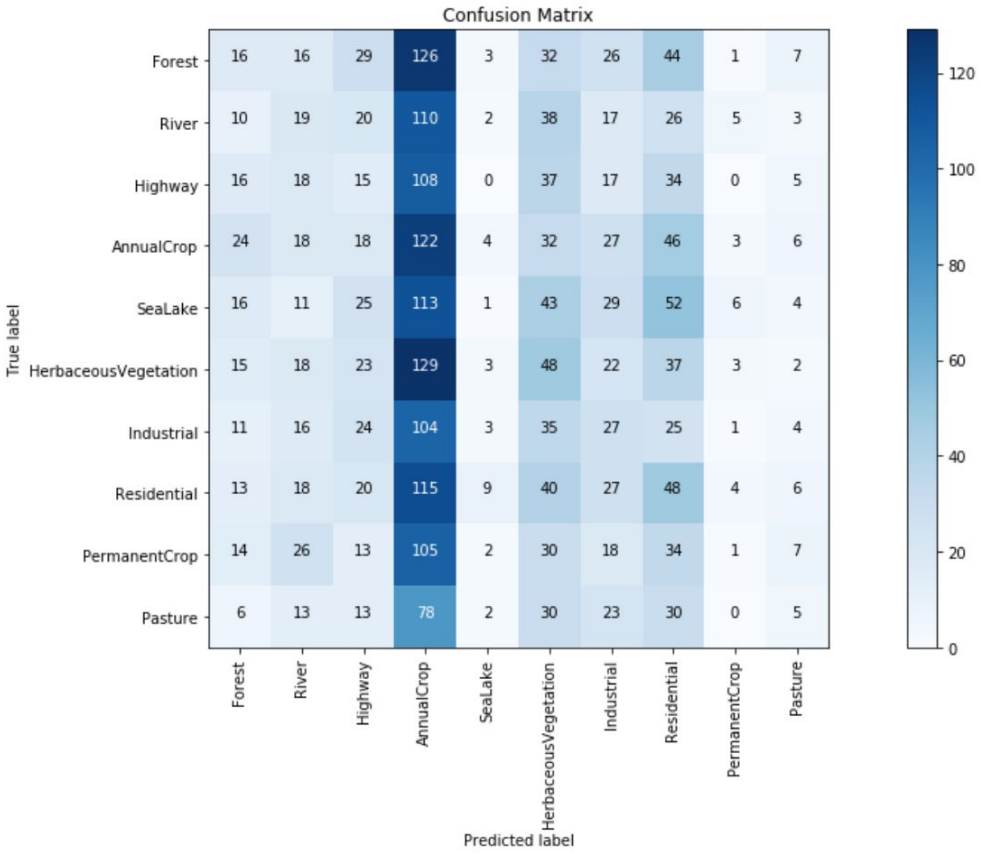


Figure 9. The confusion matrix for the tuned model's performance on the test dataset.