Lisa Taylor
Capstone 2 Final Report
Springboard Data Science Bootcamp

Satellite Imagery Classification with Deep Learning

Overview

Satellite imagery is crucial for many applications, including agriculture, city planning, natural resource management, environmental monitoring, and disaster response. Since satellite imagery can cover large areas, the labor costs to manually categorize land uses within the imagery can be prohibitive. Deep learning has emerged as an important approach for automating land use classification over extensive areas.

For this project, I have successfully deep learning using convolutional neural networks (CNNs) to classify satellite image tiles into six land use classes. Two approaches were implemented: building a custom CNN from scratch and using a pre-trained CNN. Both approaches achieved over 96% accuracy on held-out data.

The business case for developing computer vision models for satellite imagery is that automating these tasks can both reduce costs and increase accuracy of making land use classifications. This

enables automated approaches for applications like tracking land use changes over time, and mapping land uses across extensive areas that are impractical to delineate by hand.

DeepSat-6 Dataset

The DeepSat-6 dataset available on Kaggle contains pre-labeled image tiles extracted from the National Agriculture Imagery Program (NAIP) dataset. The full NAIP dataset covers the entire US at 1-meter resolution over 4 bands: red, green, blue and near-IR. The DeepSat-6 dataset consists of 28x28 pixel non-overlapping tiles extracted from NAIP imagery over different regions of California. The labels are one-hot encoded vectors for the following 6 land cover classes: barren land, trees, grassland, roads, water and buildings. Both training and test datasets are available, and the datasets have been randomized.

Data Acquisition

I decided to use the Google Colab environment for this project. The first challenge was to bring the data from Kaggle into Colab using the Kaggle API. This process involved: obtaining an API key, using shell commands within Colab to move the key to the appropriate location in the Colab file system, installing the Kaggle public API python package, and downloading the DeepSat-6 dataset via the package's command line tools.

Since the VMs in Colab file system don't persist after each session, I need to copy the dataset to my personal Google drive to make it accessible for future work. The copy was done by mounting google drive to the VM, and running shell commands from within Colab. The DeepSat-6 dataset contained the data in both csv and matlab (.mat) formats. The matlab format, since it is more compact, easy to load with scipy, and in a format close to that expected by Keras.

Data Inspection and Reworking

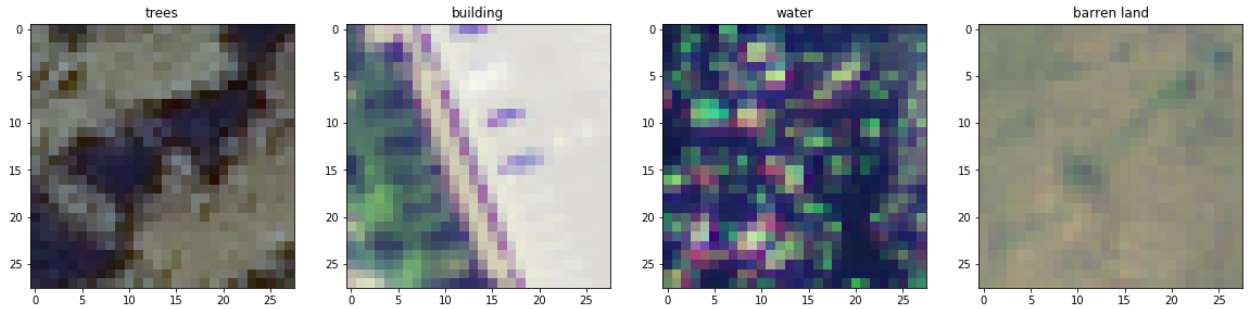
After obtaining the data, I used scipy.io.readmat() to load the dataset into a dictionary. The dictionary contained key-value pairs for the test and training data, test and training labels, and a set of annotations for translating the labels into human-readable categories. The data values were all stored as ndarrays.

A series of tests were applied to verify the quality of the raw data. These included testing that each image was assigned to a single category, that there were no null values in the image arrays, and that all pixel values were between 0 and 255. No data quality issues were identified.

The imagery data required some reworking to convert it to the format expected by Tensorflow. These steps included: 1) re-dimensioning the multidimensional image arrays from the original shape (rows, columns, channels, samples) to the channels-last convention expected by TensorFlow

(samples, rows, columns, channels); 2) transposing the label arrays; and 3) creating a pandas Series of the training labels (as text) indexed by image number.

After these manipulations, individual image tiles were easy to plot and identify by assigned class:



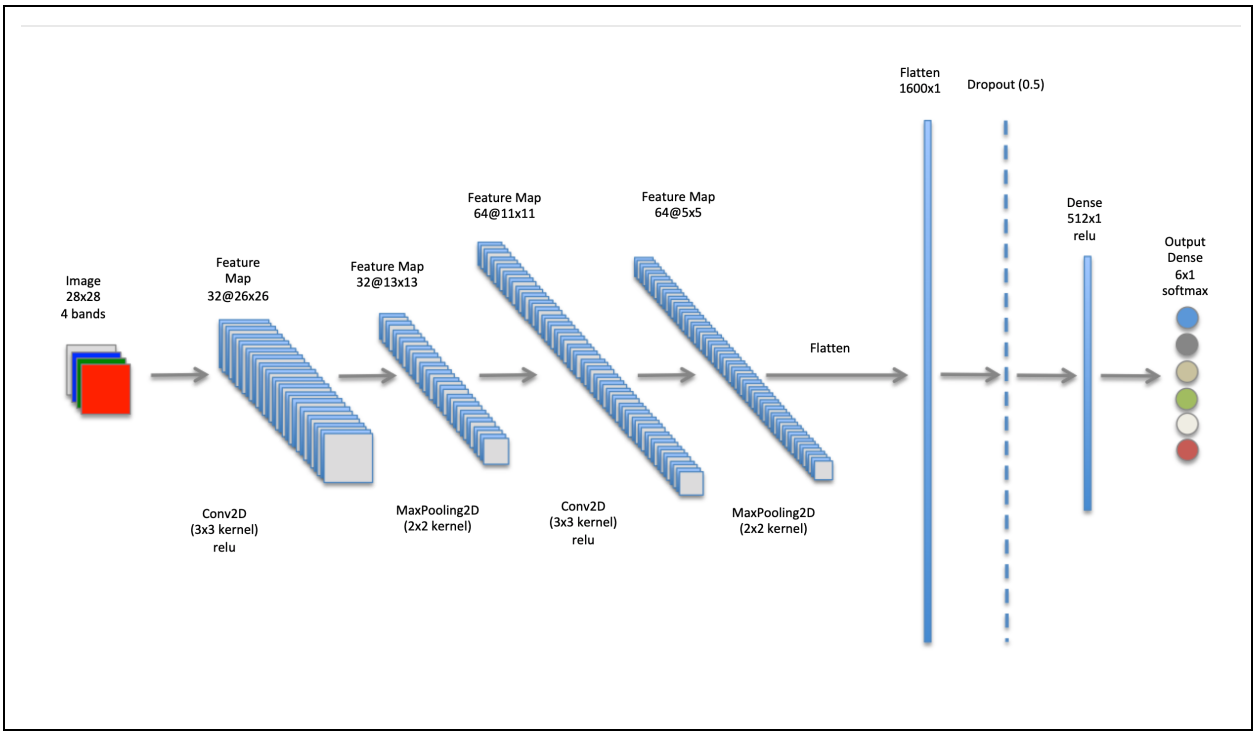
Application of Deep Learning

The convolutional neural network (CNN) is a deep learning algorithm commonly used in computer vision applications. Two approaches implementing CNNs were applied to develop classification models for the DeepSat-6 data. The first model, the “baseline” CNN, uses a simple CNN architecture. The second model implements a transfer learning approach using the VGG16 model pre-trained on ImageNet. Both models were implemented in Python on Google Colab using the Keras interface to Tensorflow and the general methods described in Deep Learning with Python (F. Chollet, 2018).

Both classification models implement a series of convolution and pooling steps (the convolutional base) followed by a densely-connected classifier to produce the final output. The convolutions learn a hierarchy of local patterns at different scales, while the dense layers learn global patterns.

Baseline Model

The baseline model was implemented using TensorFlow’s Keras API with the following architecture:



Baseline CNN design

The input to the baseline CNN is tensor representing a 28x28 4-band image tile. The first convolutional layer applies 32 convolutions (with a 3x3 kernel) to the input image followed by the rectified linear unit (relu) activation function to introduce non-linearity into the model. The result is a 3D tensor (feature map) with dimensions 26x26X32. Each of the two max pooling operations downsamples the feature maps by a factor of 2. This downsampling allows each convolution stage to learn patterns at a different scale. Due to the small extent of the input image tiles, only two stages of convolutions and max pooling were applied.

After the second max pooling operation, the input data has been transformed to a feature map with dimensions 5x5x64. This representation is flattened before being input a densely-connected classifier consisting of a fully-connected dense layer followed by a fully-connected dense output layer. The final dense layer estimates the class of the image, by applying the 'softmax' activation function to generate a probability of the input image being in each class. The class with the highest probability is assigned to the image.

As shown above, a dropout layer with a dropout rate of 0.5 was inserted after the flatten layer. The dropout layer randomly zeros out a portion of the output features of the prior layer during training, and is used as a regularization tool to prevent overfitting.

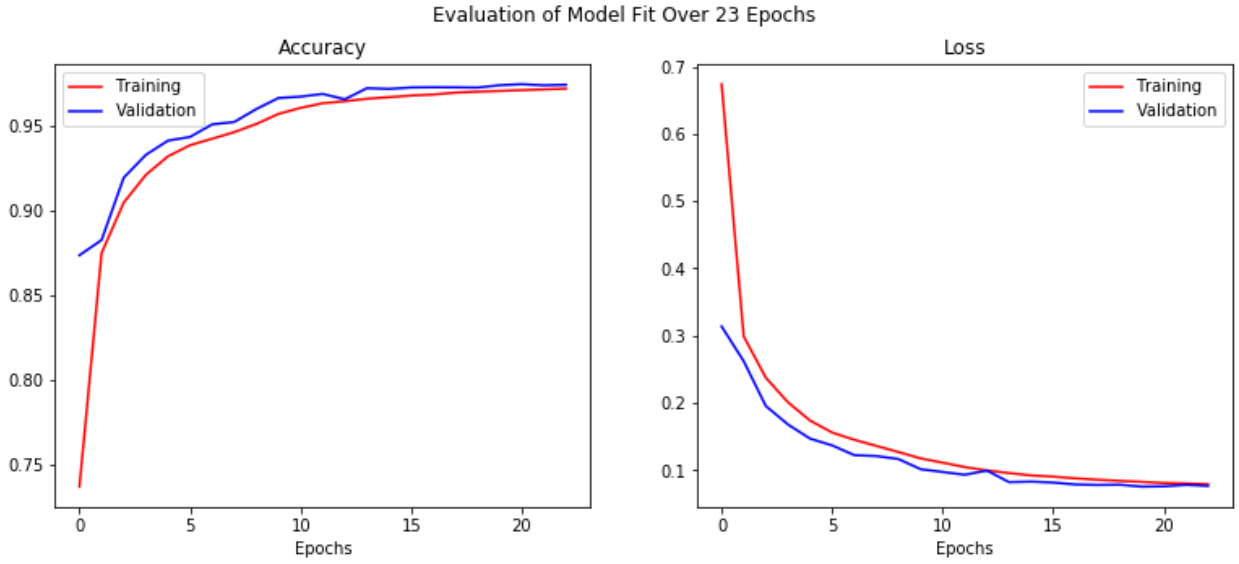
This model has a total of 842,470 parameters. Approximately 20,000 of these parameters are the weights to be trained in the convolution filters and the bias values to be applied after the convolutions. The majority of parameters are the weights of the first fully-connected dense layer of the classifier.

Compiling a model in Keras involves selecting a loss function, optimizer function, and output metrics to be reported during training. Since this is a multi-class classification problem, the categorical cross-entropy loss function was selected, which penalizes misclassifications based on the differences in log-probabilities between predicted and actual outcomes.

The baseline model was fit by backpropagation using the stochastic gradient descent (SGD) optimizer with a learning rate of 0.01 and batch size of 512. SGD was chosen by trial and error by testing different optimizers (RMSprop, Adam) and learning rates. SGD performs gradient descent using small batches of training samples. The Keras ImageDataGenerator class was used to create python generators to supply batches of pre-processed images during the model fitting process. The pre-labeled training data was randomly split into training and validation datasets, comprised of 280,000 and 64,800 images, respectively. Each generator was given a batch size of 512 images. Given the large size of the dataset, data augmentation was not needed to prevent overfitting.

The baseline model was fit using the fit_generator() method applied to the training and validation generators. The steps_per_epoch value was set to 545 and 126 for the training and validation sets to ensure the full set of image tiles would be used for training over each epoch (iteration over the full dataset). The number of epochs was set to 30 with early stopping enabled in the event the validation loss stabilized before reaching 30 epochs.

The accuracy and loss values for the training and validation datasets were recorded over the course of training. The training process achieved more than 95% accuracy on the validation data after only a few epochs. As shown below, additional epochs beyond 10 did not produce significant gains in performance, and early stopping was triggered after 23 epochs. There was no evidence of overfitting to the training data.



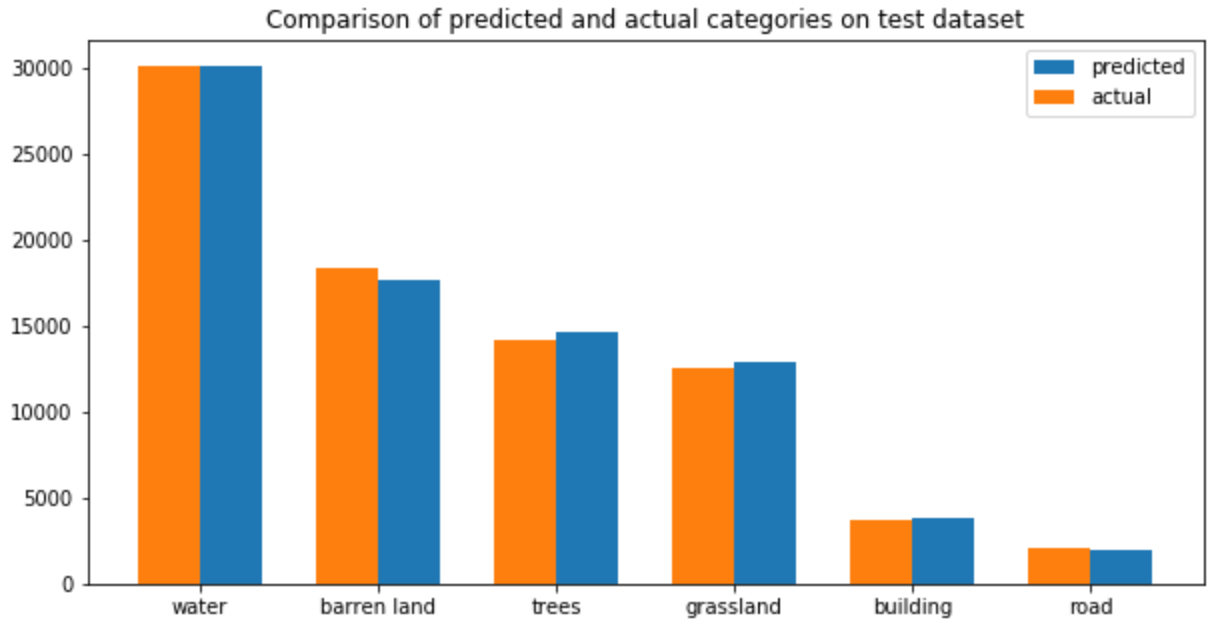
The training history plots suggested that training loss was slightly higher than the validation loss after each epoch. This counterintuitive result may be the result of applying regularization with a dropout

layer. Regularization increases the ability of the model to generalize to unseen data at the expense of the accuracy of individual training predictions. Dropout is only applied during the training stage and not during validation, this can result in validation scores that exceed training scores. Also, training loss is continuously measured over an epoch, while validation loss is measured at the end of an epoch. This can produce higher training losses when the model fit improves over the epoch.

Baseline model evaluation on held-out data

The results for the validation set during training suggested that the model would generalize well to unseen data. This was tested by using the baseline model to predict the land use classification on a set of 81,000 labeled training images provided with the DeepSat-6 dataset.

The model predicted the image classes of the held-out set with 97.5% accuracy. The following bar chart compares the frequency of the labelled and predicted classes, and demonstrates that the classes are similarly distributed, with some misclassifications of barren land, trees, and grassland.



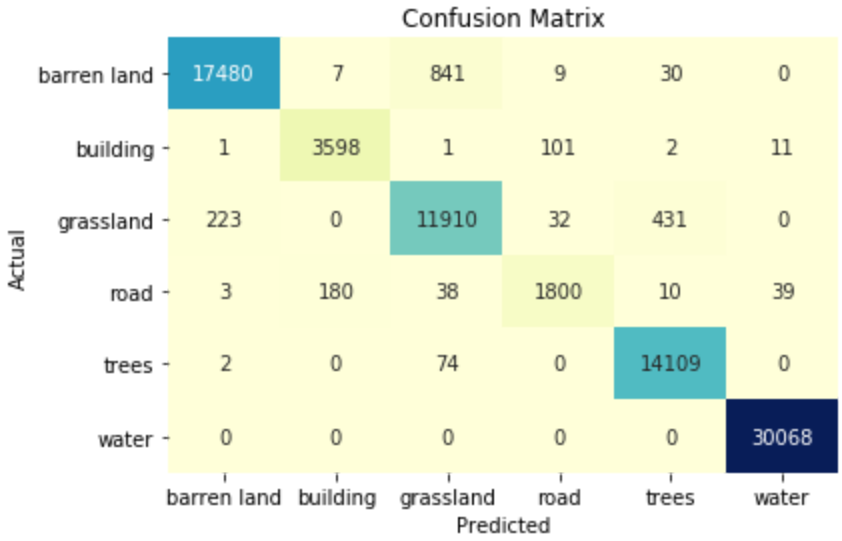
The classification report presents the precision (proportion of classifications into that class that were correct), recall (proportion of items in the class that were identified), and f1-score (harmonic mean of precision and recall) of each class. The results show slightly lower precision and recall for the grassland and road classes.

Classification Report:

	precision	recall	f1-score	support
barren land	0.99	0.95	0.97	18367
building	0.95	0.97	0.96	3714
grassland	0.93	0.95	0.94	12596
road	0.93	0.87	0.90	2070

trees	0.97	0.99	0.98	14185
water	1.00	1.00	1.00	30068
accuracy			0.97	81000
macro avg	0.96	0.96	0.96	81000
weighted avg	0.98	0.97	0.97	81000

A confusion matrix was also used to assess the baseline model's performance. The most common classification errors are barren land misclassified as grassland, and grassland misclassified as trees.

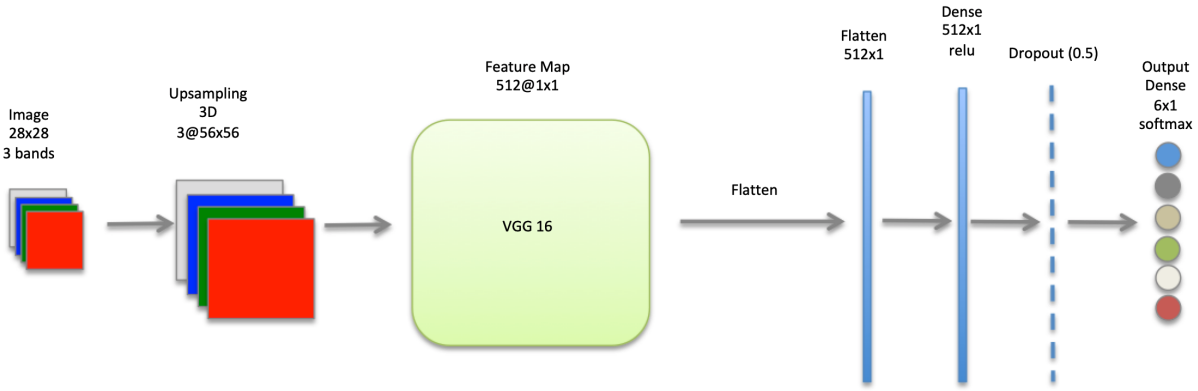


The following set of images demonstrate misclassifications made by the model. Some misclassifications occur when multiple classes are present in the same image, resulting in ambiguous class assignments.

Transfer Learning with VGG16

Transfer learning involves repurposing a convolutional base that was previously trained on a large imagery dataset for a new computer vision problem. Pre-trained CNNs can recognize general spatial features at different scales. By adding a custom backend (such as a dense classifier) to the pretrained-CNN, the combined model can be applied to solve new computer vision tasks.

For this project the VGG16 model pre-trained on ImageNet was applied to classify the DeepSat-6 dataset. The following diagram shows the architecture used to construct the model, which was again implemented in Keras:

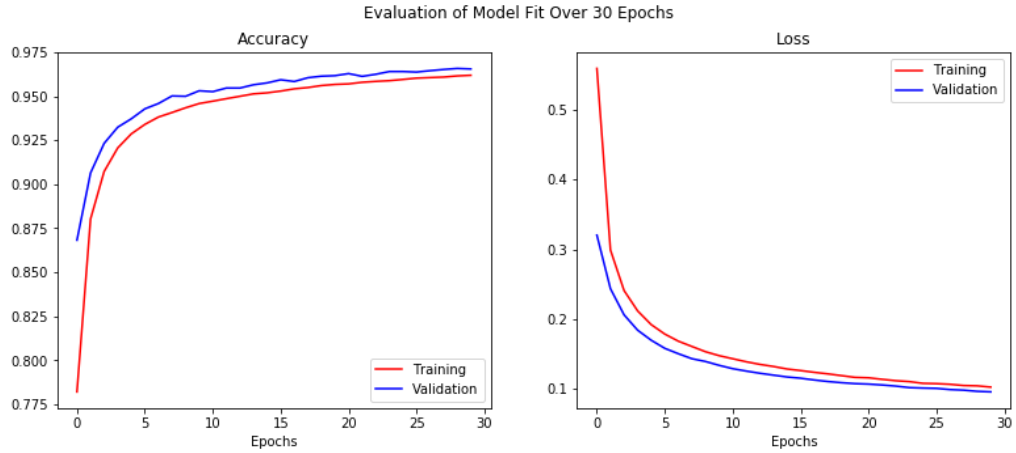


The VGG16 layer required some modification of the input data pipeline used for the baseline model. The CNN from the baseline model was trained on all 4 bands (R,G,B,IR) of the DeepSat-6 image tiles. Since VGG16 was only trained on images with 3 bands, the IR band was dropped from the input data. In addition, VGG16 has a minimum image size of 32x32 pixels. Since the image tiles in the dataset are 28x28, the Upsampling3D layer was added to upsample the rows and columns by a factor of two, transforming each image to dimensions of 56x56x3 for input to VGG16. The dense classifier on the backend was similar to that used in the baseline model.

There are two approaches for feature extraction using transfer learning: 1) running the convolutional base over the data and saving the resulting numpy array, then training a dense classifier on the array; or 2) extending the convolutional base by directly connecting a dense classifier, then running the entire model on the input data. Although more computationally intensive, the second approach was selected, due to its simplicity and potential to enable image augmentation and/or fine tuning of the top layers of VGG16 if necessary (neither was needed).

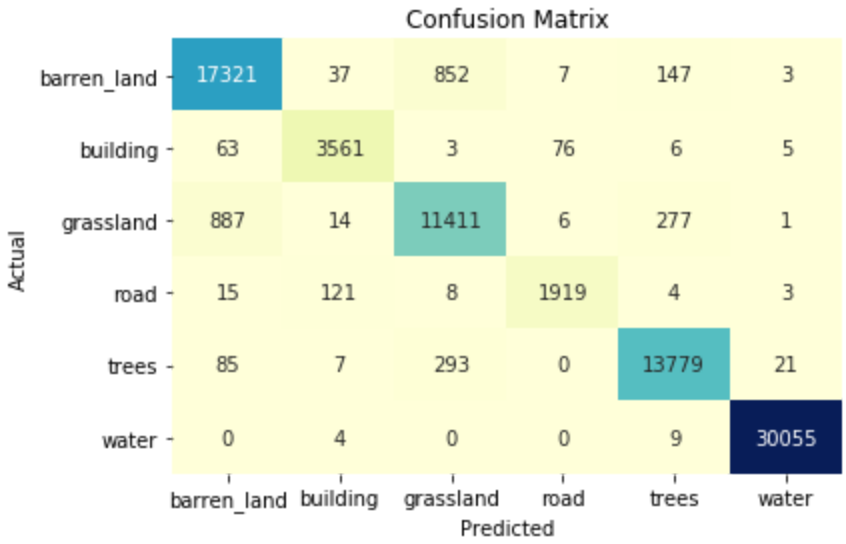
To implement the second approach, the convolutional base was frozen so the weights of the VGG16 layer would not adjust during training. The only trainable parameters in the model are in the two dense layers, roughly 270,000 parameters. The model was fit with the RMSProp optimizer (learning-rate=0.0001) over 30 epochs. Model training took approximately 23 minutes using the GPU in Colab, with a final validation data accuracy of 96.3%.

The accuracy and loss curves for fitting the transfer learning model, shown below, suggest no overfitting after 30 epochs. Additional iterations may have produced minor improvement in model fit.



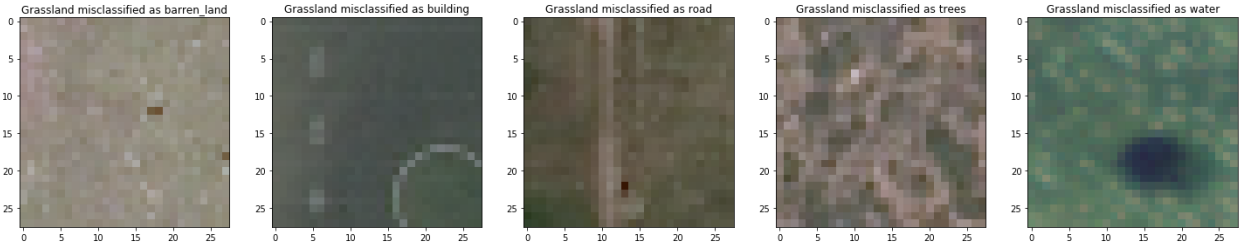
The transfer learning model was fit to the held-out data (DeepSat-6 test set). The resulting accuracy score was 96.4%, slightly lower than the baseline model (97.5%).

```
Classification Report:
              precision    recall  f1-score   support

 barren_land       0.94      0.94      0.94     18367
    building       0.95      0.96      0.95      3714
   grassland       0.91      0.91      0.91     12596
        road       0.96      0.93      0.94      2070
       trees       0.97      0.97      0.97     14185
       water       1.00      1.00      1.00     30068

    accuracy                           0.96     81000
   macro avg       0.95      0.95      0.95     81000
weighted avg       0.96      0.96      0.96     81000
```



The transfer learning model was slightly stronger at classifying water and road tiles in the test dataset compared to the baseline model, and slightly worse at the other categories. Additional training iterations may have reduced this difference.

The most difficult category for the transfer learning model to classify was grassland. The following examples of misclassifications suggest that there is substantial variation within the grassland class, and that a human classifier may not have been able to do much better:



A closer look at the class probabilities predicted in cases with errors shows that the correct class is often assigned the second highest probability predicted by the classifier. The following table shows the first 10 misclassifications in the test dataset. The only deviation from this pattern is the road/building mismatch, where the correct classification has the 3rd ranking probability.

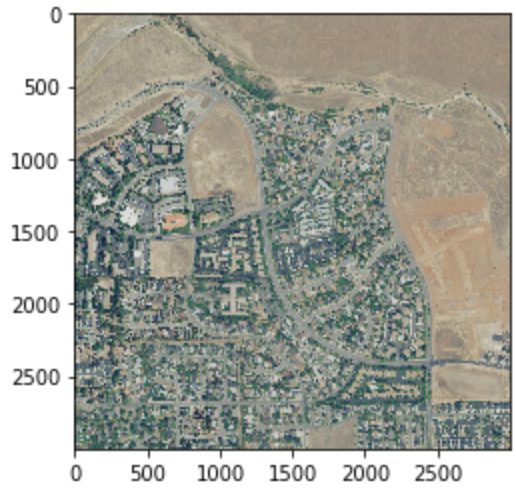
Class Probabilities							
building	**barren_land**	**trees**	**grassland**	**road**	**water**	**Actual**	**Predicted**
0.482	0.505	0.000	0.012	0.001	0.000	building	barren_land
0.000	0.151	0.022	0.827	0.000	0.000	barren_land	grassland
0.000	0.598	0.001	0.402	0.000	0.000	grassland	barren_land
0.000	0.125	0.003	0.872	0.000	0.000	barren_land	grassland

0.000	0.119	0.001	0.881	0.000	0.000	barren_land	grassland
0.101	0.891	0.000	0.000	0.007	0.000	building	barren_land
0.000	0.816	0.002	0.182	0.000	0.000	grassland	barren_land
0.763	0.000	0.000	0.000	0.237	0.000	road	building
0.000	0.113	0.059	0.828	0.000	0.000	trees	grassland
0.001	0.257	0.003	0.740	0.000	0.000	barren_land	grassland

Applying the Baseline Model to New Imagery

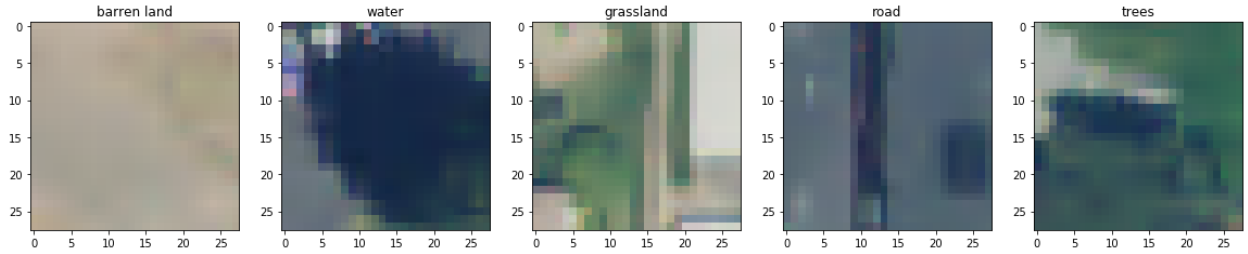
Since the DeepSat-6 dataset does not provide location data for the image tiles, the resulting classifications could not be ground-truthed against the original imagery or displayed concurrently with other spatial datasets. To get around this limitation, a larger extent of georeferenced NAIP imagery was obtained for a region of northern California using the USGS National Map Viewer application. A workflow was then developed to evaluate this more extensive image using one of the classifiers trained on the DeepSat-6 data. The baseline model was selected since it performed slightly better than the transfer learning model, and was capable of using all 4 color bands in the NAIP data.

To prepare the NAIP data for the classifier, the following square window (3000x3000x4) was sampled from the image and imported to a numpy array using the rasterio python package:

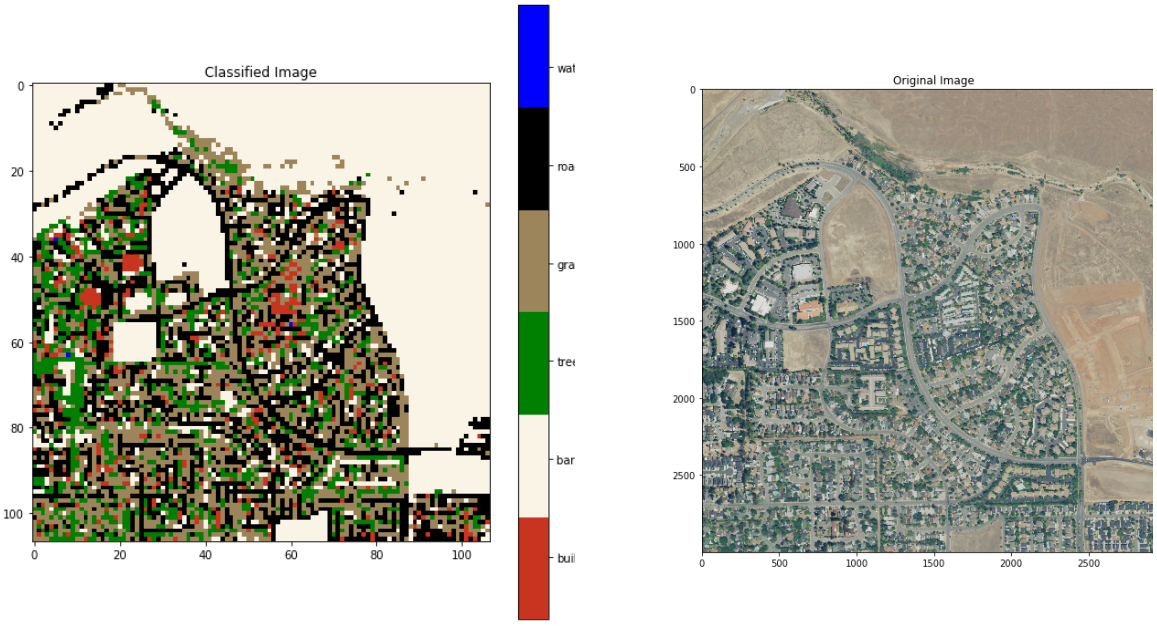


A python function was written to transform the image into a multidimensional array of image tiles with the channels-last format expected by Tensorflow. The resulting tile set has 11,449 tiles of 28 x 28 pixels. After rescaling the pixels to the range (0,1), the tiles were input to the predict method of the baseline model to classify the tiles.

A simple inspection of the results for a few tiles demonstrates that the model does a reasonable job of assigning classes:



The classification results were re-gridded to produce a “segmented” image that could be compared against the original image. The results show that the 28-pixel tile generally provides enough resolution to identify large-scale, natural features, but is not fine enough to identify land use classes at the scale of a house, street, or backyard. Since each pixel covers 1 meter of ground, the 28 pixel tile is approximately 90 feet.



Discussion and Conclusions

Two deep learning models have been developed to automate classification of satellite image tiles into 6 land use classes. Two approaches were implemented: building a CNN from scratch and using a pre-trained CNN. Both approaches achieved over 96% accuracy on held-out data. A workflow was also developed to apply classification models trained on DeepSat-6 to classify contiguous regions of NAIP data, and to produce gridded classification outputs that can be used for other spatial data workflows.

Follow-on work to extend this application could include developing an additional workflow to georeference (i.e. restore the spatial metadata) the reassembled land use classification output shown in the final figure. This enhancement would enable more complex spatial evaluations like

calculating statistics over different regions (like the distribution of classes by county), and provide the capability to develop user interfaces to interact with the classified output.

A limitation of the classification approach presented here is the loss of resolution that occurs when classifications are applied to tiles rather than individual pixels. A potential workflow to increase the resolution of the output classified image would be to classify partially overlapping tiles, and assign the classification result to the center-most pixels of each tile.

Another approach to improving resolution of the classifier would be to implement an image segmentation model using a U-Net architecture to classify image pixels at the original resolution. Training data was not readily available to implement this approach on this dataset. However, research groups have demonstrated that image segmentation models for satellite imagery can be trained using classifications derived from OpenStreetMap.