Aerial Land-Use Image Classification Using Convolutional Neural Networks and Transfer Learning

Alexander Archer, Linda Li, Brody Vogel & Jinghao Yan
Georgetown University

Abstract

This project explores applications of Convolutional Neural Network (CNN) models in classifying aerial land-use images. Models were trained with and without transfer learning to classify images into one of 21 categories. Overall, the methods detailed here achieved relatively good results, with some models attaining over 90% accuracy on the validation set.

1 Introduction

The ability to distinguish between various land-uses using images could be valuable in many fields. Accurate drone imagery classification would have applications ranging anywhere from damage estimation in insurance to national defense measures. Here, the goal is to do just that – classify images into one of 21 land-use categories using deep learning methods. The goals are two-fold: (1) To achieve the best accuracy possible, and (2) To understand what image features each layer and each activation in the models are picking up in the data.

2 Related Work

The first work on this dataset comes from the same group who extracted the images - Yang and Newsam. Working before the popularity of deep learning, Yang and Newsam used complicated Spatial Co-Occurrence Kernels (SCKs) to achieve accuracy scores around 78%. Since then, many other classification methods have been applied to this dataset. In 2017, Scott et al. were able to achieve 97.6% to 98.5% accuracy using deep CNNs and transfer learning with CaffeNet, GoogLeNet, and ResNet.

3 Data

The dataset consists of 2,100 images belonging to one of 21 classes. The images were manually extracted by the University of California, Merced, from the USGS National Map Urban Area Imagery collection. A few sample images are below:

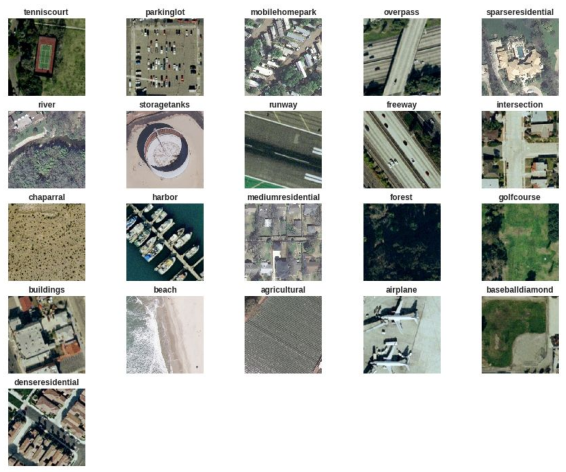

Figure 1: Land-Use Categories

Each RGB image measures 256x256 pixels, with the resolution corresponding to one square foot.

4 Methods

4.1 Training Data

The land-use dataset was divided into a training set with 80% of the images and a validation set with 20% of the images. Real-time data augmentation was used to generate batches of 100 images as the models were being trained. The training images were normalized so the values fell between 0 and 1. They were randomly sheared, zoomed with a range of 0.2, and flipped horizontally as the models were being trained.

4.2 Models

In total, five models were trained on the training images: a CNN built end-to-end, from scratch; a CNN using the frozen VGG16 weights as the base; a CNN using VGG16 weights that could be fine-tuned as the base; a CNN using the frozen weights of Google's InceptionV3 model as the base; and a CNN using Google's InceptionV3 model weights that could be fine-tuned as the base.

The CNN built from scratch consisted of 2 convolutional layers: the first had 32 ReLU-activated filters with 2 by 2 kernels and 1 by 1 strides; the second was similar but used 4 by 4 kernels. Next, 5 by 5 patches of the output were max-pooled and then flattened with a 0.25 dropout rate. The output was then fed into a dense layer with 256 ReLU-activated hidden nodes. Finally, this was used as input for the dense, 21-node softmax output layer. This model was trained for 50 epochs.

The final two layers of all the transfer learning models consisted of similar ReLU and softmax layers, but without dropout. These models were trained for 5 epochs.

At a high level, each model takes an input image, runs it through either a transfer learning base or stack of convolutional layers, then runs it through two dense layers, and finally outputs a vector of 21 probabilities that sum to 1; the image is assigned the label of the class corresponding to the highest predicted probability. A representative model blueprint is shown below:

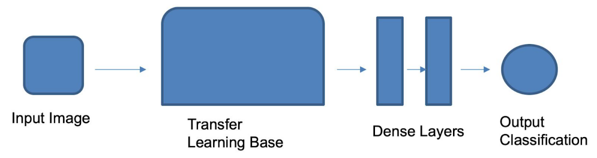

Figure 2: CNN Blueprint

Each model was then tested on the held-out validation images. This corresponds to goal (1) listed in the introduction – finding the models that perform the best on this classification task.

5 Results

For a classification task with 21 possible choices, all the models performed adequately. However, only a few reached accuracies above 92%. The full results are below:

Model	**Validation Accuracy**
VGG16 Static	0.92619
VGG16 Dynamic	0.68095
InceptionV3 Static	0.77857
InceptionV3 Dynamic	**0.94761**
Full CNN	0.70476

Figure 3: Model Results

As the table shows, the model built using Google's InceptionV3 base, where weights were fine-tuned during training (dynamic), produced an accuracy of almost 95% on the classification task.

6 Discussion

6.1 Accuracy Results

The static VGG16 model was able to achieve relatively high accuracy results with little training. However, if allowed to fine-tune the VGG16 base weights, the model quickly overfit to the training data. On the other hand, the static InceptionV3 model overfit to the data very quickly but allowing these weights to be fine-tuned prevented the model from overfitting to the data too drastically. These models therefore suggest that fine-tuning does not always improve performance.

It is unclear why the dynamic InceptionV3 model outperformed the other models, but one possible explanation is that it is a very complex and deep model. The VGG16 model is very shallow in comparison, so each backpropagation step is more likely to change all the weights to fit to the training data more so than in a deep network like InceptionV3.

6.2 Misclassifications

The image below shows what the best model - the dynamic InceptionV3 - tends to misclassify. The confusion matrix seems to indicate that this model has the most difficulty differentiating between sparse, medium, and dense residential images.

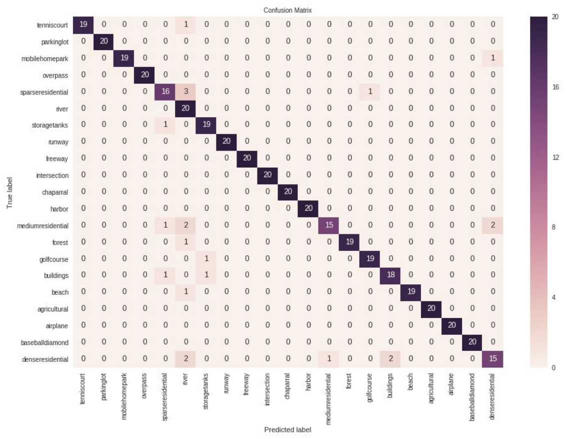

Figure 4: Dynamic InceptionV3 Confusion Matrix

The validation images that this model failed to classify correctly can be seen in the following figure. Many of these were mistakenly classified as rivers or the wrong residential category.

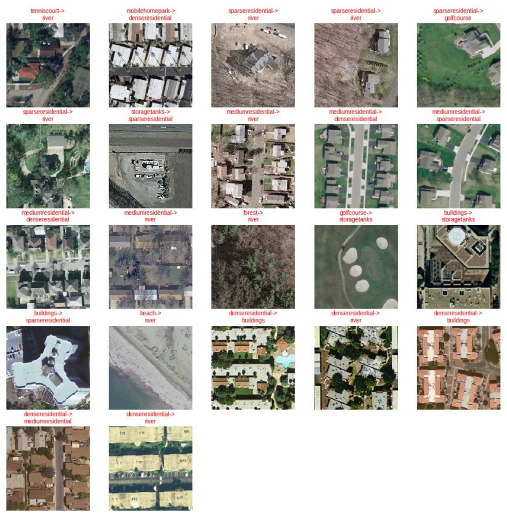
tenniscourt-> river
mobilehomepark-> denseresidential
sparseresidential-> river
sparseresidential-> river
sparseresidential-> golfcourse
sparseresidential-> river
storagetanks-> sparseresidential
mediumresidential-> river
mediumresidential-> denseresidential
mediumresidential-> sparseresidential
mediumresidential-> denseresidential
mediumresidential-> river
forest-> river
golfcourse-> storagetanks
buildings-> storagetanks
buildings-> sparseresidential
beach-> river
denseresidential-> buildings
denseresidential-> river
denseresidential-> buildings
denseresidential-> mediumresidential
denseresidential-> river

Figure 5: Dynamic InceptionV3 Misclassifications

6.3 Understanding the Models

It can be quite difficult to understand how the CNNs work. To better understand what exactly the layers are “learning” from the data, experiments were run on the model built from scratch (because it allowed us the most control over the tests). The two convolutional layers and the max-pooling layer should pick out the details in the image that warrant its classification (like the lines on tennis courts, for example).

To see what these layers are picking out, a single image can be passed through the network. Tensors can then be extracted from layers within the network (in this case, the two convolutional and one max-pooling layers) to examine the features being learned at that stage:

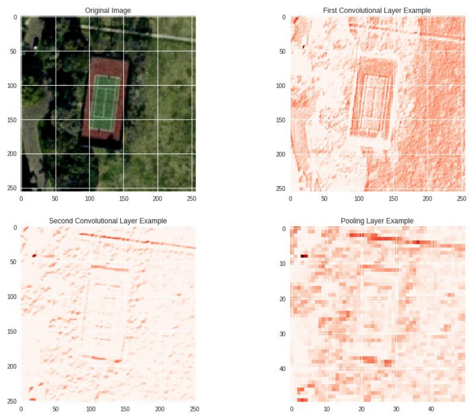

Figure 6: What do the Layers Learn?

The first layer looks for patterns like lines and clusters of similarly-colored pixels. The latter convolutional layer looks for finer details in the images, and the final layer highlights details that are unique to the predicted class (e.g. the outline of the tennis court).

In a slightly different vein, the *segments* of images that highly influence the class predictions – that is, the groups of pixels that caused the model to predict the class label it did – can also be examined:

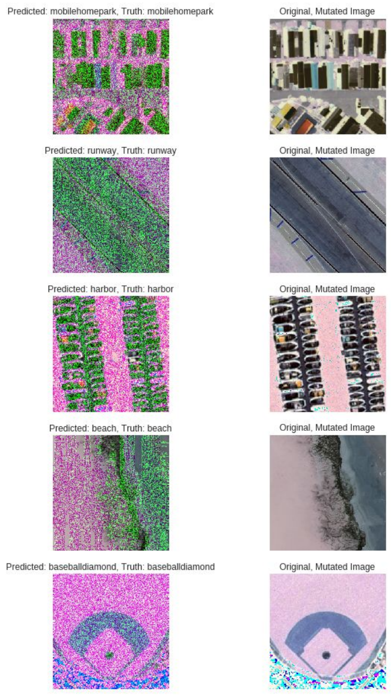
Predicted: mobilehomepark, Truth: mobilehomepark
Original, Mutated Image
Predicted: runway, Truth: runway
Original, Mutated Image
Predicted: harbor, Truth: harbor
Original, Mutated Image
Predicted: beach, Truth: beach
Original, Mutated Image
Predicted: baseballdiamond, Truth: baseballdiamond
Original, Mutated Image

Figure 7: What do the Layers Look For?

It would appear that the model is learning some complex things, like the outline of a baseball diamond and the general constitution of a harbor (boats along a dock). The model did not always perform well, though – it often mistook things like rivers and beaches that have similar features.

7 Conclusions

This project's best models are strong classifiers of land-use images; the most successful model reached nearly 95% accuracy on the task of classifying aerial land-use images into one of 21 categories, which is comparable to most of the results of previous work though slightly inferior.

If work on this project were to be continued, one might consider modifying some of the image categories. Images of medium and dense residential land-use, for example, are similar and can be difficult to distinguish even for a human. These two and potentially other classes could be merged to cut down on unimportant misclassifications. Also, these models can currently only classify individual images taken from a small range of altitude levels. These models could be made more robust by training on images of a wider range of scales. Similarly, additional data augmentation such as rotations and vertical flipping during training could be beneficial.

Regardless, this project achieved strong accuracy scores, has important applications, and could be used to propel future research.[1]

8 References

Scott, Grant J., et al. “Training Deep Convolutional Neural Networks for Land–Cover Classification of High-Resolution Imagery.” Geoscience and Remote Sensing Letters, IEEE, vol. 14, no. 4, IEEE, Apr. 2017, pp. 549–53, doi:10.1109/LGRS.2017.2657778.

Yi Yang and Shawn Newsam, "Bag-Of-Visual-Words and Spatial Extensions for Land-Use Classification," ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS), 2010.

Zhou, et al., “Learning Deep Features for Discriminative Localization,” Computer Vision Foundation (IEEE Xplore), 2016.

[1] All code and results from this project are available at the public GitHub repository: https://github.com/ll3091/ANLY-590-02-Deep-Learning-Project