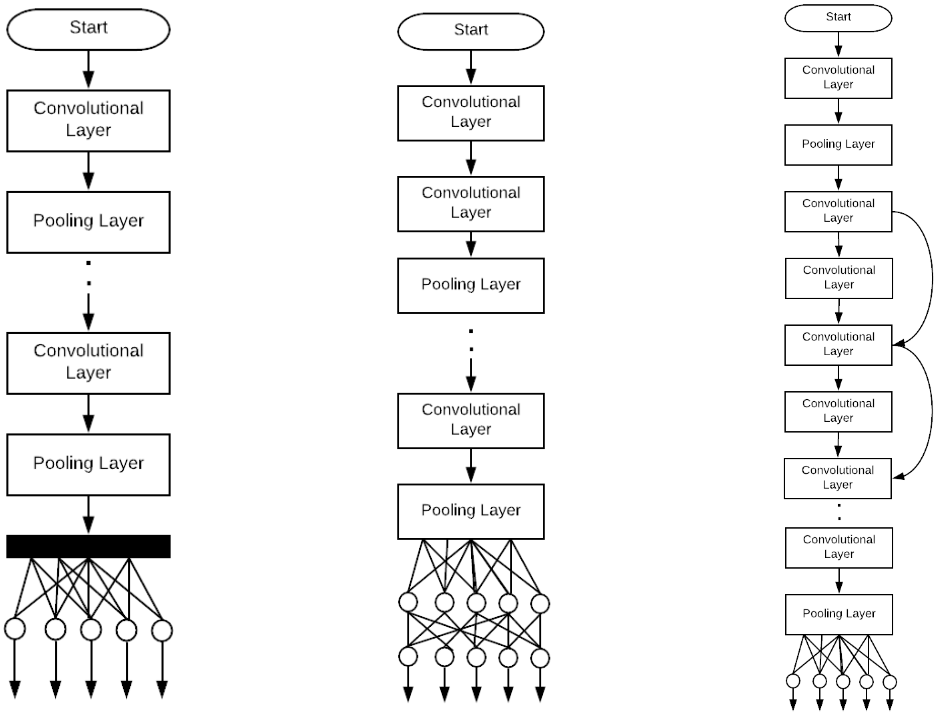
James Hurt<br>
CS 8780<br>
9 May 2018

# Comparison of Convolutional Neural Networks for Remote Sensing

## 1 Technical Description

Remote Sensing is a lucrative application for computational intelligence, with much of the recent research being in Convolutional Neural Networks. Within this discussion of neural networks for remote sensing, Deep Convolutional Neural Networks (DCNN) are the most popular. The popularity of convolutional neural networks can be attributed to the ability of a convolutional network to learn the features of the image, thus eliminating one of the greatest challenges in all of machine learning: feature selection and extraction. While CNN have proven very useful, one of the issues with DCNN is the high training time, even with GPU acceleration. In this project, I attempt to compare a shallow convolutional network



(a) Custom Shallow CNN (b) VGG19 (c) ResNet-50

Figure 1: Architectures of the Compared CNN

with one of the first deep convolutional networks. I then take my comparison one step farther, by looking at the relative cost and benefit of residual neural networks.

I chose to make my comparison of CNN for the application of Remote Sensing because I am very interested in Remote Sensing. I think the ability to detect object from satellite imagery has many applications in every day life and can better the lives of common people as well as aid industry in understanding what is around their facilities.

## 1.1 Shallow Convolutional Neural Network

The shallow convolutional neural network used in this project combines many techniques of fully connected networks previously used for pattern recognition. The architecture of the network can be see in Figure 1a.

The architecture of the network has a total of 12 layers. The first 10 layers are alternating 2D Convolutional layers layers with a 3x3 kernel size and 2D Max Pooling layers with a 2x2 Pooling Window. After these 10 layers, feature extraction is finished, and a flattening layer is introduced to put the data into a single vector for the final fully connected softmax layer.

## 1.2 Deep Convolutional Neural Network

The Deep Convolutional Neural Network used for comparison in this project is VGG 19 [1]. The VGG 19 network, as the name suggests, has 19 total layers. This count, however, does not include the pooling layers. The basic architecture can been seen in Figure 1b.

VGG19 has 16 convolutional layers and 4 pooling layers. The interesting quality about VGG19 is that the final pooling layer does not connect directly to the softmax output layer. Instead, the final pooling layer is a 3-layer multilayer perceptron (MLP) [2] The final layer of this MLP is a softmax layer for classification.

## 1.3 Deep Residual Convolutional Neural Network

The final neural network used for comparison in this project is a Deep Residual Convolutional Neural Network (DRNN). I chose to use ResNet-50 [3], a standard DRNN for comparison developed by Microsoft Research. The architecture of this network can be seen in Figure 1c.

ResNet-50 begins with a single convolutional layer, followed by a single pooling layer, followed by 49 convolutional layers that are residual, meaning that there is a connection from layer *N* to both layer *N+1* as well as layer *N+2*. This connection exists so that the error from the two connections to the *N+2* layer can be summed when performing backpropagation, thus slowing the disappearance of the gradient. The disappearing gradient is one of the greatest challenges of using the gradient descent method.

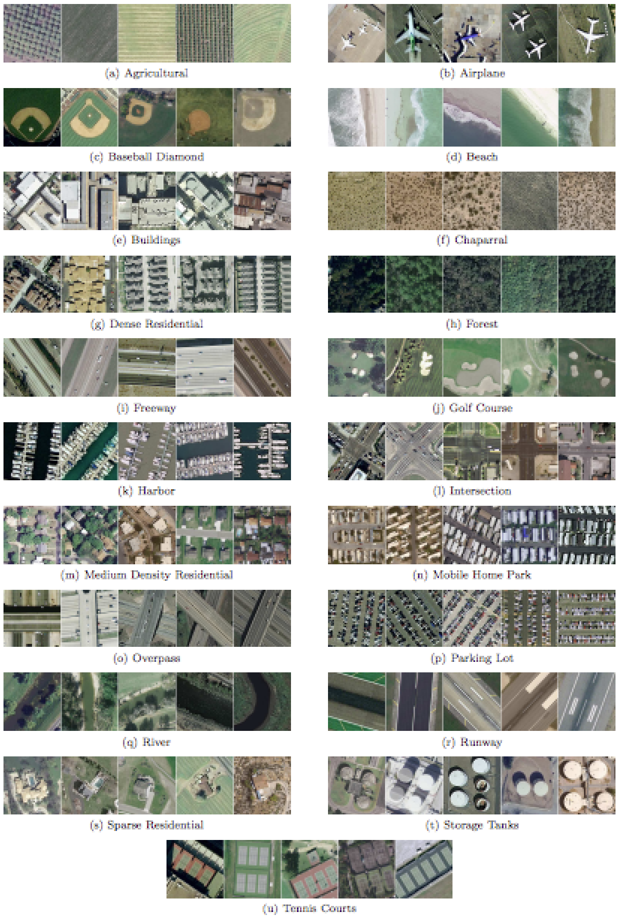

(a) Agricultural
(b) Airplane
(c) Baseball Diamond
(d) Beach
(e) Buildings
(f) Chaparral
(g) Dense Residential
(h) Forest
(i) Freeway
(j) Golf Course
(k) Harbor
(l) Intersection
(m) Medium Density Residential
(n) Mobile Home Park
(o) Overpass
(p) Parking Lot
(q) River
(r) Runway
(s) Sparse Residential
(t) Storage Tanks
(u) Tennis Courts

Figure 2: UCMerced Dataset with 21 classes

# 2 Algorithm Design

## 2.1 Dataset Selection

The algorithm I designed for comparing these various CNN involved 2 datasets: a 3 class dataset and the benchmark UC Merced Dataset [4]. The 3 class problem is a subset of the UCMerced dataset. I chose 3 classes from the UCMerced dataset: agricultural, airplane, and baseball diamond. While the test dataset is small, the UCMerced dataset, seen in Figure 2 is not. It has 21 classes, each with 100 images each. This dataset is not the largest available as far as benchmark Remote Sensing is concerned, but I felt it was a good representative for my comparison, because I didn't want too challenging a dataset.

I chose to use 2 datasets because I felt that using only one would not give me enough of a picture to draw conclusions. Perhaps, if I had chosen just one dataset, the dataset I picked was particularly suited to one of these methods, but if I chose 2, then I would have enough to draw somewhat of a conclusion. Certainly, only 2 datasets is not enough for a generalization, but it does allow me to eliminate the possibility of choosing a biased dataset.

I also chose to have a small and large dataset, because there are many instances of binary classification needed in everyday life, so looking at a 3 class problem as well as a large, 21 class problem, would allow me to see if one method was better for large but not small or vice versa. If I found that the shallow could better handle the 3 class problem or maybe if the shallow CNN was within 1% cross validation accuracy, for example, then I could say that binary classifiers could settle for a shallow CNN, thus relieving the long training times required for Deep CNN or RNN.

### 2.2 Parameter Selection

After choosing the datasets to use, I had to decide on the parameters for the algorithms. The choices I had to make specifically were the loss function and the optimizer. I needed to choose one of each that would be used in all 3 algorithms. I had to use the same one to ensure a fair comparison.

The loss function I decided to use was **categorical crossentropy** [5]:

$$H(q,p) = \int q(x) log(q(x)/p(x))\mathrm{d}x$$

Crossentropy, as discussed in the first Computational Intelligence class, is a useful loss function because it harshly punishes output that is very far from the correct value. It is more punishing than standard error and can help a network converge in less epochs.

For the optimizer, I started with knowledge of gradient descent from the previous Computational Intelligence class, and decided to choose a similar optimizer for my CNN, **Stochastic Gradient Descent** [6]:

$$w(t+1) = w(t) - \eta \frac{\partial}{\partial w} \ell(f(x_t), y_t)$$

The idea of this optimizer is the same as gradient descent in that we are attempting to move the opposite direction of the error function in search of a set of parameters such that there exists a global minimum for error.

### 2.3 Comparison Algorithm

## 3 Experiment Results

### 3.1 Test Dataset

### 3.2 UCMerced Dataset

## 4 Analysis

## 5 Code

The code for this project can be seen in the **Code** folder.

# References

[1] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Sept. 2014. arXiv: 1409.1556.

[2] J. M. Keller, D. Liu, and D. B. Fogel, *Fundamentals of Computational Intelligence: Neural Networks, Fuzzy Systems, and Evolutionary Computation*. Piscataway, NJ: Wiley-IEEE Press, 1 edition ed., July 2016.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015. arXiv: 1512.03385.

[4] Y. Yang and S. Newsam, "Bag-of-visual-words and spatial extensions for land-use classification," p. 270, ACM Press, 2010.

[5] J. Shore and R. Johnson, "Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy," *IEEE Transactions on Information Theory*, vol. 26, pp. 26–37, Jan. 1980.

[6] O. Bousquet and L. Bottou, "The Tradeoffs of Large Scale Learning," p. 8.