Roof Style Classification using Convolutional Neural Networks

Mohan Kishore, Krishnamurthy Srikanth

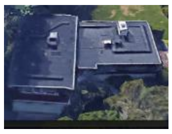
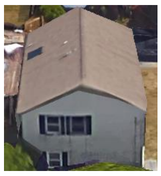
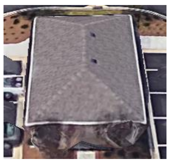
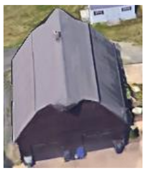
Department of Information Systems, Northeastern University, Boston 02115
Email: s.krishnamurthy@northeastern.edu

***Abstract*—** *Realistic 3D building modeling and rendering can be achieved by accurately recognizing building roof style. This also has potential impact in risk analysis due to natural catastrophes and plays a crucial role in underwriting sector of insurance industry. In this paper, we propose a novel system for aerial image based roof style classification using deep learning techniques. Our system is trained to recognize four individual roof styles. First, we train the system with regular deep nets which enable our system to flatten the image input and predicts the roof style using the flattened information. Second, to better characterize a roof image, we design convolutional neural network that takes the raw input image in three dimensions and predicts appropriate roof style of a building. We demonstrate that the second approach has much better performance compared to conventional deep nets. Finally, we propose the convolutional neural network with optimal parameters that provides maximum accuracy without overfitting .*

***Keywords**: Roof style, Deep neural network, Convolutional neural network, Aerial images, Google Earth,*

I. Introduction

The primary objective of the application is to assess the appropriate roof style. There are four primary roof styles consistently spread across US:

Roof type-0: Flat	
Roof type-1: Gable	
Roof type-2: Hip	
Roof type-3: Gambrel	

The application has two aspects:

(I) Deep Neural Networks – High resolution aerial images (either tiff format or png format) downloaded from Google Earth is fed to the network. Extracted features – Detected edges are again passed to the deep neural networks to benchmark the performance of raw images against feature extracted images[1].

(II) Convolutional Neural Networks – High resolution images are fed to convolutional layer, pooling layer and dense layer [2]. The accuracy obtained is compared with Deep networks.

Problem Type: Supervised Classification

II. Steps Involved

Following are the steps involved in model build:
(I) Read Images
(II) Regular Deep net construction for benchmarking
(III) Convolutional neural network construction and training
(IV) Accuracy and Analysis.

III. Reading Images

Package Used - Opencv

Opencv has higher gaussian performance [3] over scipy and hence chosen as our primary package to read and process images. In case of regular neural network, to perform dot product of input values and weights:

Pred_Y = X.W + b

Since this is matrix multiplication (X.W), it should be in the "mxn" and "nxj" format to get "mxj" form. To enable this multiplication, the image must be flattened to 1 dimension. In our case, all our images are first reshaped to 128x128x3. Using opencv, all the images are converted to Grayscale thereby reducing third dimension to 128x128x1.

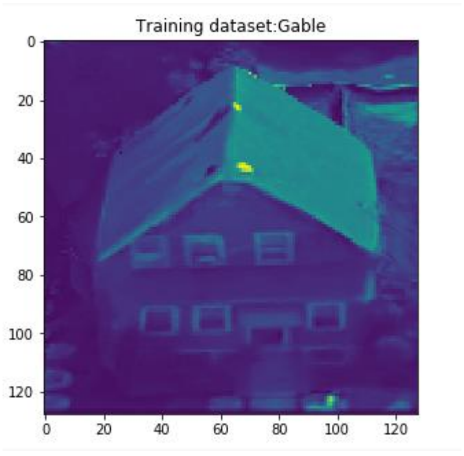


Fig 1: Grayscale image

This converted image is further reduced to get a one dimensional matrix of 16384 and fed to a regular deep network.
Another deep net with edges detected was ran to benchmark performance against regular deep net[2].

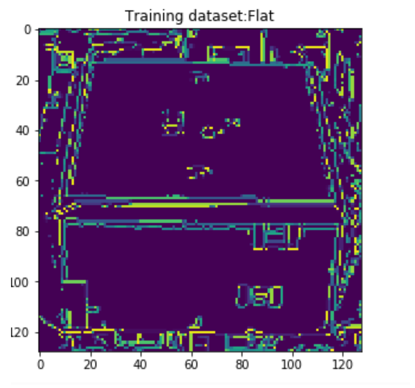


Fig 2: Edge detected image

Note: In case of convolutional neural network, images of dimension (128x128x3) are directly fed to the model and tested.

IV. Regular Deep net for benchmarking

Package Used: Tensorflow

Layers:

Layers	**Dimensions**
Input_layer	16384
Output_layer	4

Processed Image (nx16384) is fed to the model.

Computation:

Y_ = SoftMax(Pred_Y = X.W + b)

Cross Entropy: $$-\sum_{i=0}^{n} \ln\left(o_i\right) * t_i$$

Optimizer: Gradient Descent to minimize cross entropy

Optimizer: Gradient Descent to minimize cross entropy

Softmax provides values between 1 and 0 for all the classes associated with the image. The negative sign in cross entropy is to get rid of negative value we get while applying log on predicted value computed using softmax function. Weights and biases are adjusted till local optima is reached using Gradient descent algorithm.

V. Convolutional neural network construction and working

Package Used: Tensorflow with CUDA - Bitfusion Amazon Cloud (AWS)

Layers Involved:

(I) Convolutional Layer – This is the layer where mxn filter is convolved over 128x128 image. The convolved image is then fed to pooling layer.

(II) Pooling Layer – This is the layer where spatial size of the representation is reduced, thereby reducing the amount of features and computational complexity of the network.

(III) Dense Layer – This is the layer where flattened images are passed through multiple fully connected layers. Activation function used is ReLU – Rectified Linear Unit. ReLU is commonly preferred function mainly because of its similarity to biological neurons, where a signal is triggered only when signal strength exceeds threshold.

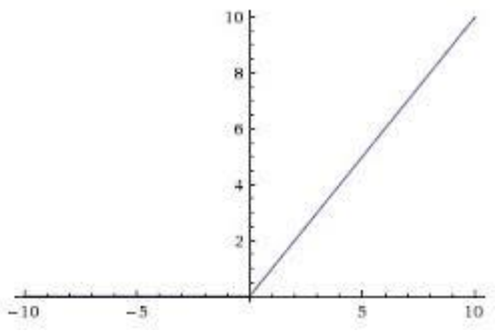


Fig 3: ReLU Activation

(IV) Dropout Layer – This is the layer where specified amount of neurons are dropped to avoid regularization. Overfitting can be avoided by shooting down specific number of neurons during every iteration.

(V) Logits Layer – This is the layer where sigmoid function is applied to get values in the boundary (0,1) for every class label.

Logit(X) = (1/1+exp(-X))

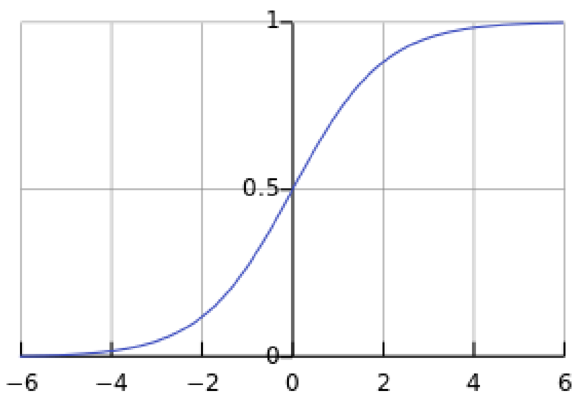
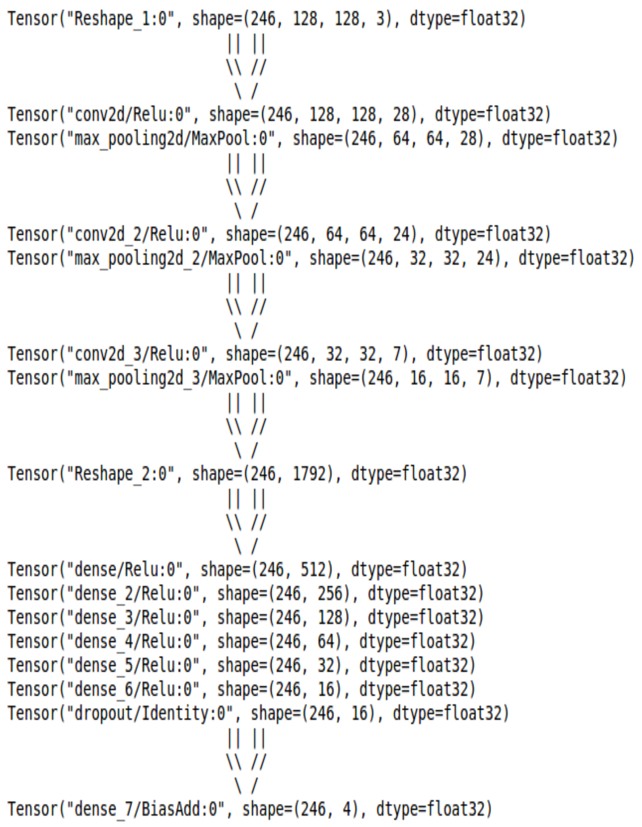


Fig 3: Logit function

Logit function is preferred over Softmax activation mainly because of its reduced comlexity. Having softmax for batch normalized CNN will increase the complexity[4].



```
Tensor("Reshape_1:0", shape=(246, 128, 128, 3), dtype=float32)
                          || ||
                          \\ //
                           \ /
Tensor("conv2d/Relu:0", shape=(246, 128, 128, 28), dtype=float32)
Tensor("max_pooling2d/MaxPool:0", shape=(246, 64, 64, 28), dtype=float32)
                          || ||
                          \\ //
                           \ /
Tensor("conv2d_2/Relu:0", shape=(246, 64, 64, 24), dtype=float32)
Tensor("max_pooling2d_2/MaxPool:0", shape=(246, 32, 32, 24), dtype=float32)
                          || ||
                          \\ //
                           \ /
Tensor("conv2d_3/Relu:0", shape=(246, 32, 32, 7), dtype=float32)
Tensor("max_pooling2d_3/MaxPool:0", shape=(246, 16, 16, 7), dtype=float32)
                          || ||
                          \\ //
                           \ /
Tensor("Reshape_2:0", shape=(246, 1792), dtype=float32)
                          || ||
                          \\ //
                           \ /
Tensor("dense/Relu:0", shape=(246, 512), dtype=float32)
Tensor("dense_2/Relu:0", shape=(246, 256), dtype=float32)
Tensor("dense_3/Relu:0", shape=(246, 128), dtype=float32)
Tensor("dense_4/Relu:0", shape=(246, 64), dtype=float32)
Tensor("dense_5/Relu:0", shape=(246, 32), dtype=float32)
Tensor("dense_6/Relu:0", shape=(246, 16), dtype=float32)
Tensor("dropout/Identity:0", shape=(246, 16), dtype=float32)
                          || ||
                          \\ //
                           \ /
Tensor("dense_7/BiasAdd:0", shape=(246, 4), dtype=float32)
```

Fig 5: Layers in the model

Optimizer: Stochastic Gradient Descent (SGD) with learning rate = 0.001

V. Accuracy and Analysis

Network type	**Accuracy**
Regular Net	34.92%
Regular Net - After Edge detection	34.92%
Convolutional Neural Net	69.84%

Inferences

- Performance with 1D array is low because we destroy all the features (color and 2D)

- Performance with edges extracted is same as Regular Net. Feature selection and normalization processes have zero impact on the performance of net.

- Performance in Convolutional nets has increased tremendously. Increasing number of records in training and test set will further increase accuracy.

VI. Conclusion

For close to 250 training images and 60 test images, CNN yields an accuracy of about 70%. If the number of images in training and testing set are increased, the network will have increased accuracy.

References

[1] Automatically Generating Roof Models from Building Footprints (2003) R.G. Laycock and A.M. Day

[2] Very Deep Convolutional Networks for Large Scale Image Recognition (2015) Karen Simmoniyan and Andrew Zisserman.

[3] Python for prototyping computer vision application (2010) Brian Thorne and Raphael Grasset

[4] Tensorflow and Deep Learning – Without a PhD Martin Gorner

[5] Tensorflow Tutorial Documents - https://www.tensorflow.org/versions/r0.10/tutorials/deep_cnn/