Identifying Area of Rooftops in Denver, CO, for Solar Policy

Milestone Report

Problem Statement

Climate change as a crisis that decisive action. Now. A huge part of that solution involves shifting our productions of energy away from fossil fuels and towards renewable sources of energy. Paul Hawken's famous Project Drawdown lists rooftop solar as a key strategy of meeting our ambitious climate goals. Their analysis suggests that rooftop solar can grow from 0.4% of global electricity generation to 7% by 2050, and "that growth can avoid 24.6 gigatons of emissions."

Rooftop solar is an increasingly appealing option for those looking to not only contribute to building a better world, but those hoping to save money in the long term through reduced energy bills and net metering. Policies exist to incentivize the application of rooftop solar, including several in Denver: City and County of Denver Elevation Loans, Solar* Rewards through Xcel, and the RENU Loan through the Colorado Energy Office.

But how would rooftop solar would it take to essentially "run" Colorado, the state where the sun shines 300 days a year? What about just single-family residential electricity needs?

In true engineering fashion, we can begin to estimate this answer by taking it to the extreme: if every rooftop in Denver was outfitted with PVs, how much electricity could we produce?

Since this is a data science project and not an engineering project, our process will be simplified. We'll use a standard rate of electricity produced per area of PV on a rooftop. One (very basic) rule of thumb is to say that every square foot of roof space generates 15 watts of solar energy (based on a solar panel with 15.8% efficiency).

According to the US Energy Information Administration, Coloradans use an average of 706 kilowatt hours in their homes each month. According to the U.S. Census, there are 320,545 houses in Denver County. With some basic math, we can estimate that the residential sector of Denver County uses roughly 225 GW of solar energy.

Therefore, we're looking to identify roughly 1.5 x 10^10 square feet of rooftops in Denver, or 538 square miles.

Solution
Given satellite imagery of Denver, we can train a convolutional neural network to correctly identify rooftop area.

- Label 75+ images of rooftops with the tool provided generously by Dr. Tony Szedlak
- Build a data generator to apply random cropping of size (256,256) of the (800,800) images
- Build a CNN
- Train the CNN given the generated data
- Fine tune the model's hyperparameters to increase model accuracy
- Predict given all 1,800+ satellite images for Denver, CO

Data
The City and County of Denver generously provides open source data via https://www.denvergov.org/opendata. For this project, I am using their 2004 dataset of satellite imagery of Denver County. These data consist of almost 2,500 JP2 files.

From the Open Data website:

> Color, 6 inch resolution aerial photography for the City and County of Denver acquired in 2004. True-ortho correction is applied to all structures 4 stories and above in designated areas including all bridges and highway overpasses.
>
> Coverage includes a 359 square mile area encompassing the City and County of Denver, City of Glendale, City of Littleton, and the Denver Water Service Area. This project does not include DIA.
>
> Spatial reference is NAD 1983 HARN StatePlane Colorado Central FIPS 0502.

For the benefit of the analysis, I converted these JP2 files into JPEGs using XnConverter for Linux.

Exploratory Data Analysis Results

You can view the full results and code of this EDA in my jupyter notebook.

Each image has 2,640 rows and columns and three channels (RGB) and there are a total of 2,553 images in the data set. The max value in an RGB image is 255 (and a min of 0).

Type of the image : <class 'numpy.ndarray'>
Shape of the image : (2640, 2640, 3)
Image Height 2640
Image Width 2640
Dimension of Image 3

We can plot the channels against each other and look for any kind of correlation. Below I plot the first chanel (0) against the third channel (2) for the first image in the sest (Figure 1) and the fifth image (Figure 2. The histogram shows that there are a lot of pixels have an equal value of R and B (grey).

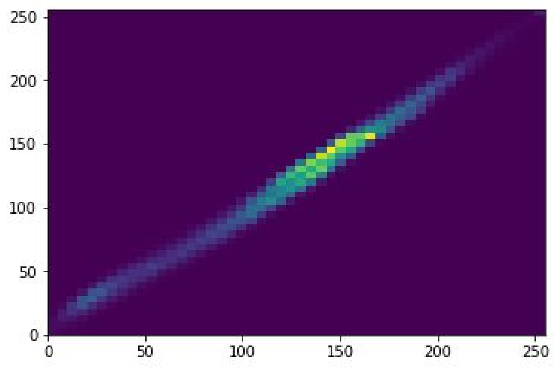


Figure 1. Histogram of R and B values in the first image

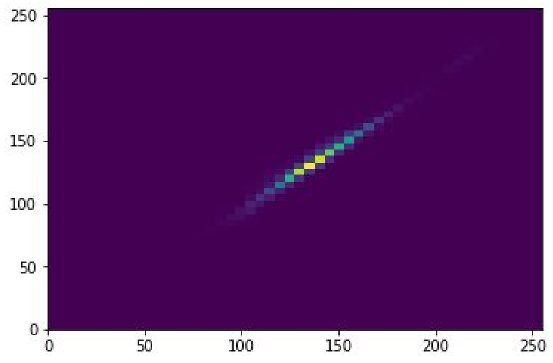


Figure 2. Histogram of R and B values in the fifth image

We can compare channels (in this case, the third channel) between the two images. The histogram below shows that the most common shared values lie in the middle... again, grey.

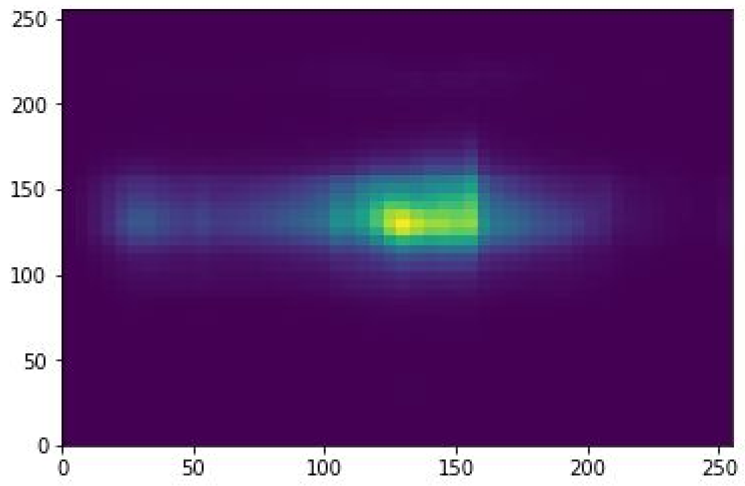


Figure 3. The comparison of the B channel between the 1st and 5th image.

We can experiment with compressing an image by applying a quick k-means clustering on the image with a k of 5. This gives us an impression of what kinds of features we might be able to extract from the image set.

From the result below, we see a dark purple for the roads, yellow for the green spaces, and blues and greens for the built environment.

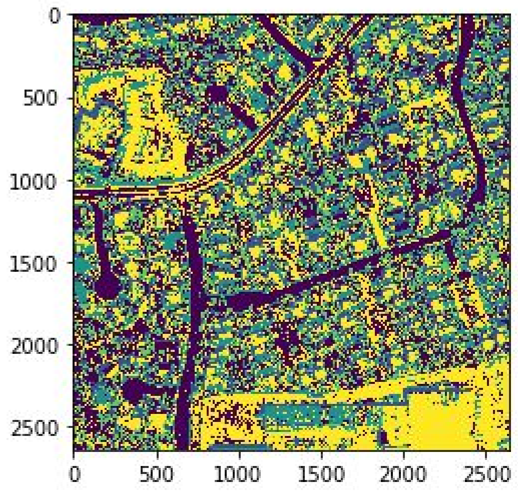


Figure 4 K-means clustering with k of 5 on the first image.

We can plot the histogram of the color values for each channel in the image. All three channels seem to correlate strongly with each other, indicating a lot of grey values.

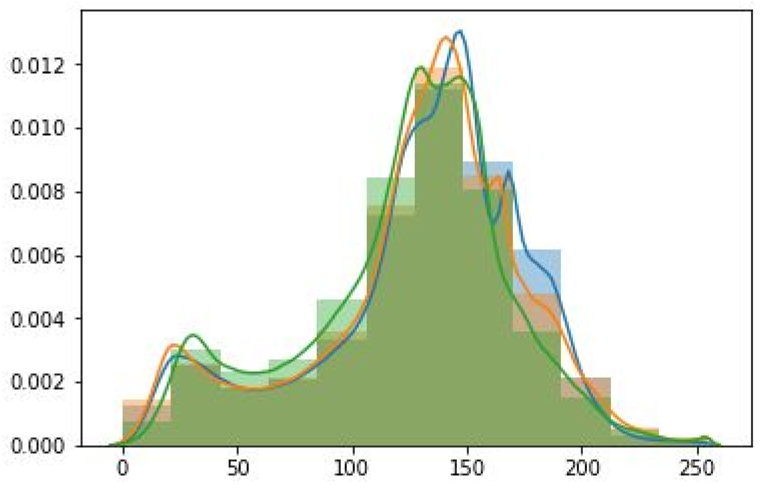


Figure 5. Histogram of 3 color channels for the first image

We can play more with the RGB channels and split the image accordingly.

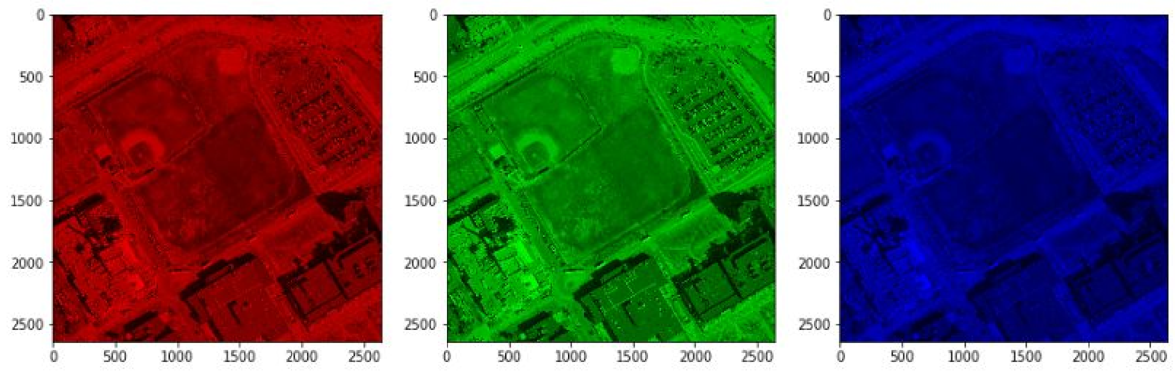


Figure 6. The split R,G, and B channels for a random image (1002 image).

Now we can get into image convolution. First, we can "blur" the image with a convolution matrix that averages the nearby pixels,

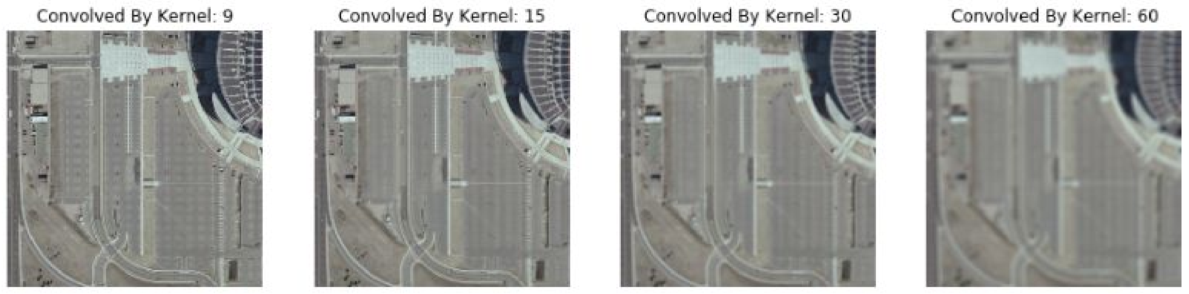


Figure 7.The 1006th image convolved by 9, 15, 30, and 60

Figures 8-11 show varying convolution filters applied to the same image. We see that throughout these filters, we're starting to segment out important features. In this image, we see the outline in the parking lot, the buildings, and the road.

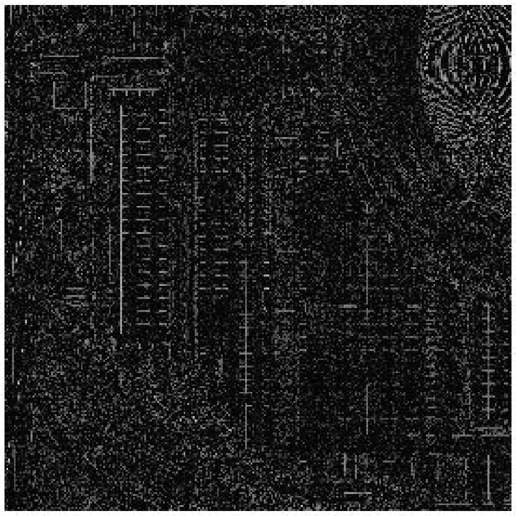


Figure 8. Outline kernel

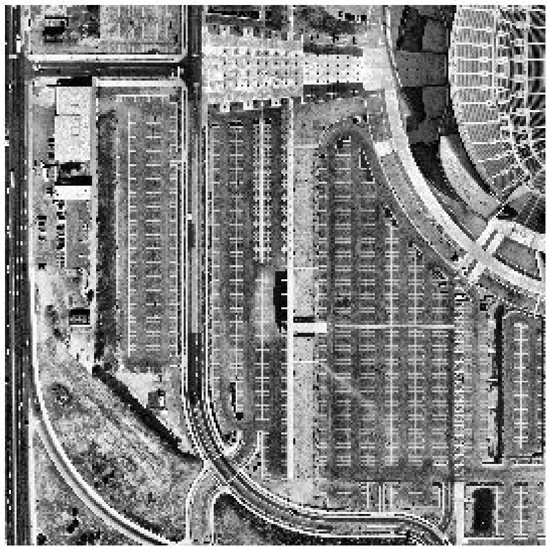
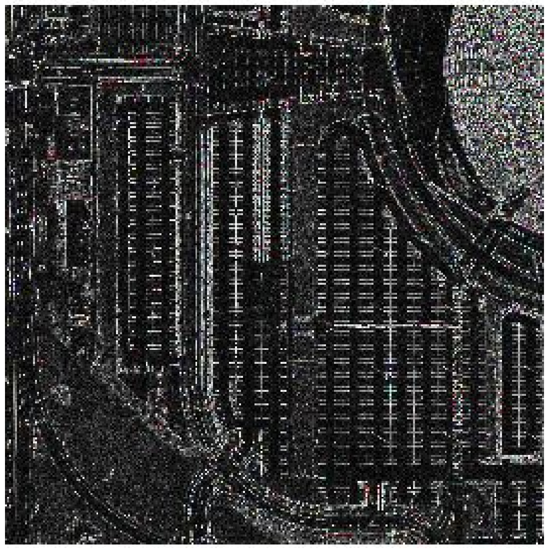


Figure 9. Gaussian window



Sobel 10. Sobel kernel

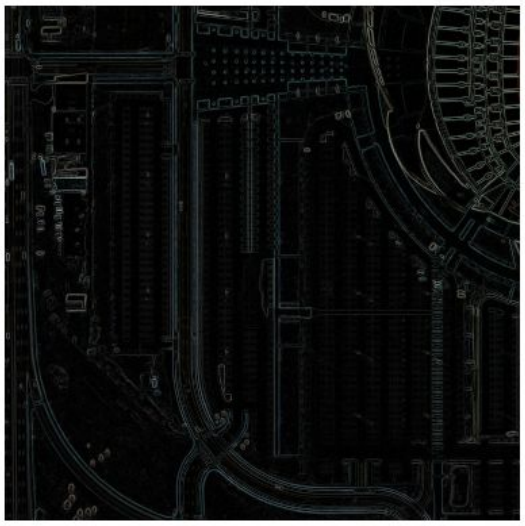


Figure 11.Gaussian Filter

Conclusions

I am so lucky to be working with such high quality images. With a 6 inch resolution, our EDA shows that even basic image manipulation can generate building outlines. These images are mostly grey, and I don't expect color to be a big deciding factor in determine what is and what is not a rooftop. We could convert these images to black and white.

Data Generation

You can view the code of my data generator here.

A neural net needs way more than 83 images to train on, so standard practice is to create an image generator to, you guessed it, generate more images.

The base of my data generator takes an (1000, 1000) px image and it's corresponding mask and:

- randomly rotates the images between 0 and 360 degrees
- randomly crops the images to size (256,256).
- randomly flips the images along the vertical axis
- randomly rotates the images an additional 90 degrees

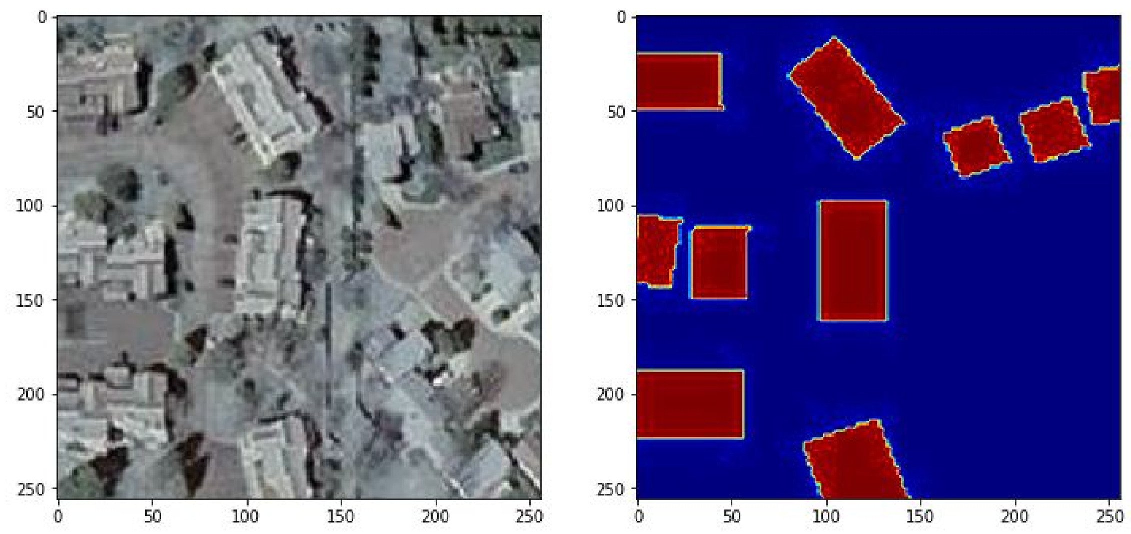


Figure 12. A cropped image and it's associated mask.

This data generator's batch size is 4, which will produce:

4 images/step * 100 steps/epoch *50 epochs = 20,000 images

Building the model

The model's architecture is based off of the U-NET architecture U-NET is a artificial neural network based on ConvNets that is widely used for image segmentation work. The U-NET architecture is shown below in Figure 13, which features a set of encoding layers (on the lefT) then decoding layers (on the right). Each encoding chunk, as shown by the set of three lines in each row below, has a convolution block followed by a maxpool downsampling. The decoding section features upsampling and concatenation followed by more convolution.

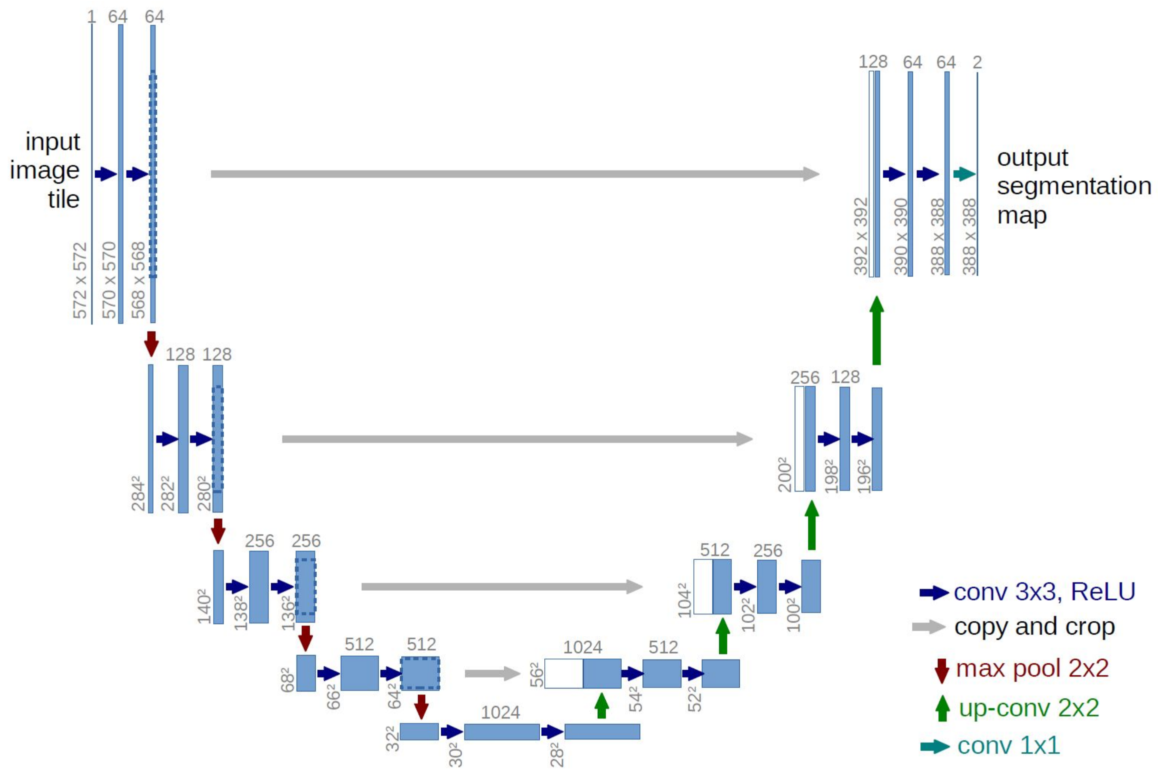


Figure 13.U-NET model architecture

The modified model architecture I chose for this project is based off of [Fabio Sancinetti's U-NET ConvNet for CT-Scan segmentation](). The full code for the model is presented below, [and you can see the full code here.]()

```
inputs = Input((256, 256, 3))
conv1 = Conv2D(8, (3, 3), activation='relu', padding='same')(inputs)
conv1 = Conv2D(8, (3, 3), activation='relu', padding='same')(conv1)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
drop1 = Dropout(0.5)(pool1)

conv2 = Conv2D(16, (3, 3), activation='relu', padding='same')(drop1)
conv2 = Conv2D(16, (3, 3), activation='relu', padding='same')(conv2)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
drop2 = Dropout(0.5)(pool2)

conv3 = Conv2D(32, (3, 3), activation='relu', padding='same')(drop2)
conv3 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv3)
pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
drop3 = Dropout(0.3)(pool3)

conv4 = Conv2D(64, (3, 3), activation='relu', padding='same')(drop3)
conv4 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv4)
pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)
drop4 = Dropout(0.3)(pool4)

conv5 = Conv2D(128, (3, 3), activation='relu', padding='same')(drop4)
conv5 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv5)

up6 = concatenate([Conv2DTranspose(64, (2, 2), strides=(2, 2),
    padding='same')(conv5), conv4], axis=3)
conv6 = Conv2D(64, (3, 3), activation='relu', padding='same')(up6)
conv6 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv6)

up7 = concatenate([Conv2DTranspose(32, (2, 2), strides=(2, 2),
    padding='same')(conv6), conv3], axis=3)
conv7 = Conv2D(32, (3, 3), activation='relu', padding='same')(up7)
conv7 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv7)

up8 = concatenate([Conv2DTranspose(16, (2, 2), strides=(2, 2),
    padding='same')(conv7), conv2], axis=3)
conv8 = Conv2D(16, (3, 3), activation='relu', padding='same')(up8)
conv8 = Conv2D(16, (3, 3), activation='relu', padding='same')(conv8)

up9 = concatenate([Conv2DTranspose(8, (2, 2), strides=(2, 2),
    padding='same')(conv8), conv1], axis=3)
conv9 = Conv2D(8, (3, 3), activation='relu', padding='same')(up9)
conv9 = Conv2D(8, (3, 3), activation='relu', padding='same')(conv9)

conv10 = Conv2D(1, (1, 1), activation='sigmoid')(conv9)

model = Model(inputs=[inputs], outputs=[conv10])
```

The model's summary is:

```
Layer (type)                     Output Shape         Param #     Connected 
to
============================================================================
=======
input_1 (InputLayer)             (None, 256, 256, 3)  0
____________________________________________________________________________
_______
conv2d_1 (Conv2D)                (None, 256, 256, 16) 448
input_1[0][0]
____________________________________________________________________________
_______
conv2d_2 (Conv2D)                (None, 256, 256, 16) 2320
conv2d_1[0][0]
____________________________________________________________________________
_______
max_pooling2d_1 (MaxPooling2D)  (None, 128, 128, 16) 0
conv2d_2[0][0]
____________________________________________________________________________
_______
dropout_1 (Dropout)              (None, 128, 128, 16) 0
max_pooling2d_1[0][0]
____________________________________________________________________________
_______
conv2d_3 (Conv2D)                (None, 128, 128, 32) 4640
dropout_1[0][0]
____________________________________________________________________________
_______
conv2d_4 (Conv2D)                (None, 128, 128, 32) 9248
conv2d_3[0][0]
____________________________________________________________________________
_______
max_pooling2d_2 (MaxPooling2D)  (None, 64, 64, 32)   0
conv2d_4[0][0]
____________________________________________________________________________
_______
dropout_2 (Dropout)              (None, 64, 64, 32)   0
max_pooling2d_2[0][0]
____________________________________________________________________________
_______
conv2d_5 (Conv2D)                (None, 64, 64, 64)   18496
dropout_2[0][0]
____________________________________________________________________________
_______
conv2d_6 (Conv2D)                (None, 64, 64, 64)   36928
conv2d_5[0][0]
____________________________________________________________________________
_______
max_pooling2d_3 (MaxPooling2D)  (None, 32, 32, 64)   0
conv2d_6[0][0]
____________________________________________________________________________
```

```
-------
dropout_3 (Dropout)             (None, 32, 32, 64)   0
max_pooling2d_3[0][0]
-----------------------------------------------------------------------------
-------
conv2d_7 (Conv2D)               (None, 32, 32, 128)  73856
dropout_3[0][0]
-----------------------------------------------------------------------------
-------
conv2d_8 (Conv2D)               (None, 32, 32, 128)  147584
conv2d_7[0][0]
-----------------------------------------------------------------------------
-------
max_pooling2d_4 (MaxPooling2D)  (None, 16, 16, 128)  0
conv2d_8[0][0]
-----------------------------------------------------------------------------
-------
dropout_4 (Dropout)             (None, 16, 16, 128)  0
max_pooling2d_4[0][0]
-----------------------------------------------------------------------------
-------
conv2d_9 (Conv2D)               (None, 16, 16, 256)  295168
dropout_4[0][0]
-----------------------------------------------------------------------------
-------
conv2d_10 (Conv2D)              (None, 16, 16, 256)  590080
conv2d_9[0][0]
-----------------------------------------------------------------------------
-------
conv2d_transpose_1 (Conv2DTrans (None, 32, 32, 128)  131200
conv2d_10[0][0]
-----------------------------------------------------------------------------
-------
concatenate_1 (Concatenate)     (None, 32, 32, 256)  0
conv2d_transpose_1[0][0]

conv2d_8[0][0]
-----------------------------------------------------------------------------
-------
conv2d_11 (Conv2D)              (None, 32, 32, 128)  295040
concatenate_1[0][0]
-----------------------------------------------------------------------------
-------
conv2d_12 (Conv2D)              (None, 32, 32, 128)  147584
conv2d_11[0][0]
-----------------------------------------------------------------------------
-------
conv2d_transpose_2 (Conv2DTrans (None, 64, 64, 64)   32832
conv2d_12[0][0]
-----------------------------------------------------------------------------
-------
concatenate_2 (Concatenate)     (None, 64, 64, 128)  0
conv2d_transpose_2[0][0]
```

```

conv2d_6[0][0]
____________________________________________________________________________
_______
conv2d_13 (Conv2D)              (None, 64, 64, 64)   73792
concatenate_2[0][0]
____________________________________________________________________________
_______
conv2d_14 (Conv2D)              (None, 64, 64, 64)   36928
conv2d_13[0][0]
____________________________________________________________________________
_______
conv2d_transpose_3 (Conv2DTrans (None, 128, 128, 32) 8224
conv2d_14[0][0]
____________________________________________________________________________
_______
concatenate_3 (Concatenate)     (None, 128, 128, 64) 0
conv2d_transpose_3[0][0]

conv2d_4[0][0]
____________________________________________________________________________
_______
conv2d_15 (Conv2D)              (None, 128, 128, 32) 18464
concatenate_3[0][0]
____________________________________________________________________________
_______
conv2d_16 (Conv2D)              (None, 128, 128, 32) 9248
conv2d_15[0][0]
____________________________________________________________________________
_______
conv2d_transpose_4 (Conv2DTrans (None, 256, 256, 16) 2064
conv2d_16[0][0]
____________________________________________________________________________
_______
concatenate_4 (Concatenate)     (None, 256, 256, 32) 0
conv2d_transpose_4[0][0]

conv2d_2[0][0]
____________________________________________________________________________
_______
conv2d_17 (Conv2D)              (None, 256, 256, 16) 4624
concatenate_4[0][0]
____________________________________________________________________________
_______
conv2d_18 (Conv2D)              (None, 256, 256, 16) 2320
conv2d_17[0][0]
____________________________________________________________________________
_______
conv2d_19 (Conv2D)              (None, 256, 256, 1)  17
conv2d_18[0][0]
============================================================================
=======
Total params: 1,941,105
```

```
Trainable params: 1,941,105
Non-trainable params: 0
____________________________________________________________________________
_______
```

The model hyperparameters are presented in Table 1.

Table 1. Model hyperparameters

Hyperparameter	**Value**
Training batch size	4
Learning Rate	0.001
Steps per epoch	5
Epochs	200
Optimizer	Adam
Loss metric	Binary cross entropy

Training Results

After running for 200 epochs, the model's final accuracy was 0.91637 and the loss was 0.1560. The model's performance for loss and accuracy for both the test and train sets is plotted in Figure 14. From the loss plot, we see that while loss decreases through the training run, it appears that the model is not yet done training. The model's accuracy jumps around, indicating that we may need to tweak the model architecture or lower the learning rate.

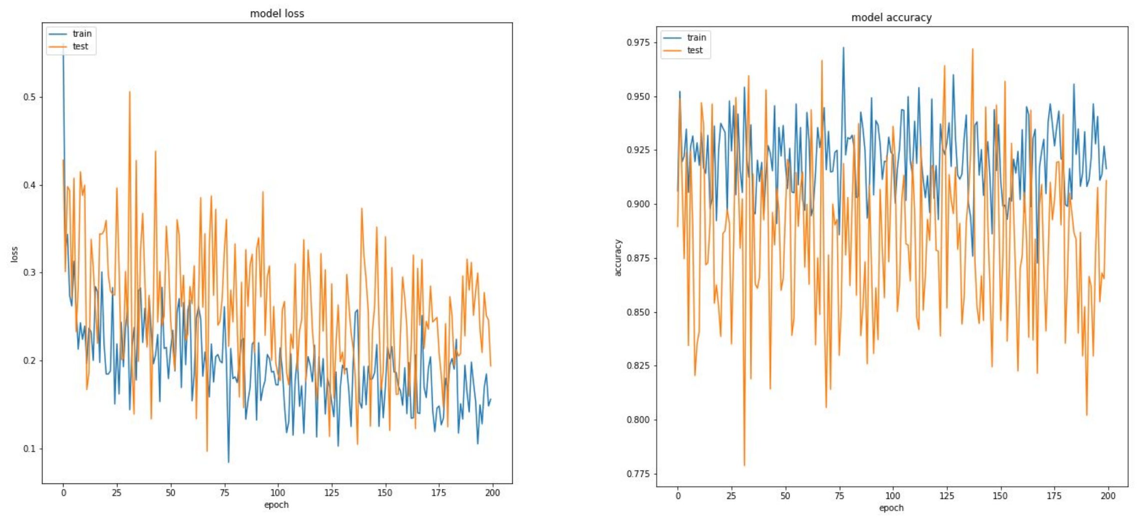


Figure 14. [Left] Training loss (blue) and test loss (orange) over the 200 epochs during training. [Right] Model accuracy over the 200 epochs.

We can do a “spot check” and apply the model to one of our images. When we predict rooftops on a random image, we see that the result is very noisy, although the predicted masks do correlate with rooftops.

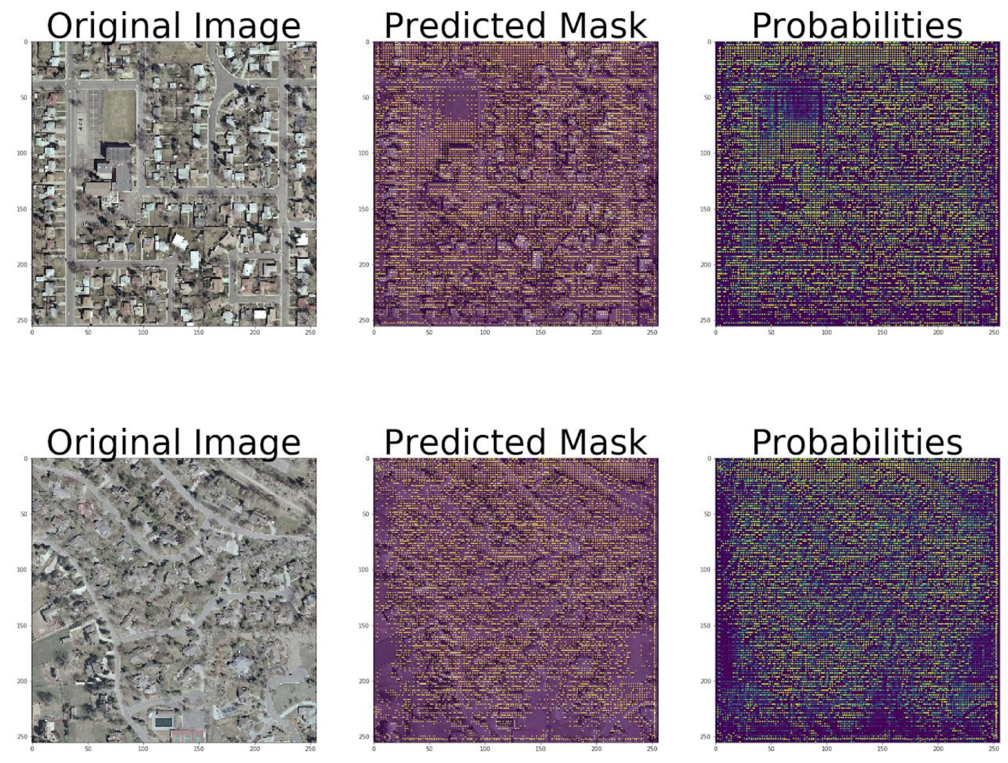


Figure 15. Two plots each showing a random image, the model’s prediction for the rooftops, and the raw probabilities for each prediction.

References

1.Novel Approach for Rooftop Detection Using Support Vector Machine
https://www.hindawi.com/journals/isrn/2013/819768/

(2) Automatic Rooftop Detection Using a Two-stage Classification
http://ijssst.info/Vol-15/No-4/data/4923a285.pdf