Machine Learning Advanced Nanodegree

Capstone Proposal for Classifier for Kaggle's Amazon from Space competition

Vivek Pothina

December 21st, 2018

I. Definition

Project Overview

A few decades ago, Image Classification was a difficult task for computers, but now using Deep Learning we can empower satellites to label satellite images. "Planet: Understanding Amazon from Space" is competition in Kaggle. This is a multi-label classification problem too. The dataset has been collected by Planet, designer and builder of the world's largest constellation of Earth-imaging satellites, will soon be collecting daily imagery of the entire land surface of the earth at 3-5-meterd resolution. Using a simple neural network to solve this problem would be slow and ineffective. CNN/Convnet has been used to use the spatial information to capture features of the satellite images such as cloudy or rain forest features. The predictions and analysis of this data about the location of deforestation and human encroachment on forests can help governments and local stakeholders respond more quickly and effectively.

Project Statement

The goal of this project is to create a model that can label all the satellite image chips, there are 17 labels and this is multi label classification, the end result would be given a satellite image chip, the model would classify the image into one or more labels. Deep learning techniques have become popular in the last few years as they have beat several other machine learning algorithms on various machine learning challenges. Convolutional Neural Networks are one such example, dominates in the field of image classification and object detection.

This specific problem is a competition in Kaggle. The dataset is in Kaggle and can be downloaded from link: https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/. The process is simple, data is preprocessed, a convolutional neural network model is built, then trained on the dataset and evaluated on the test set.

Metrics

To measure the performance of the model, a validation set was created initially from the training set. Performance was measured on validation set every epoch and both log-loss and accuracy were measured.

$$accuracy = \frac{number\ of\ correct\ predictions}{size\ of\ dataset}$$

The loss function is,

$$-\frac{\mathbf{1}}{\boldsymbol{N}}\sum_{\boldsymbol{n}=\mathbf{1}}^{\boldsymbol{N}}[\boldsymbol{y_n}\,\mathbf{log}(\widetilde{\boldsymbol{y_n}}) + (\mathbf{1}-\ \boldsymbol{y_n})\,\mathbf{log}(\mathbf{1}\ -\ \widetilde{\boldsymbol{y_n}})]$$

Where N = 17 i.e. the number of classes/labels in the dataset, $\boldsymbol{y_n}$ is the label vector and $\widetilde{\boldsymbol{y_n}}$ is the prediction vector.

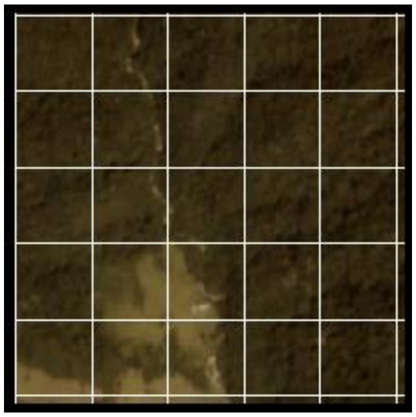
This particular loss function has been chosen as this problem is multi-label classification. Categorical cross entropy is used for multi class problem. But this is a multi-label classification problem.

II. Analysis

Data Exploration

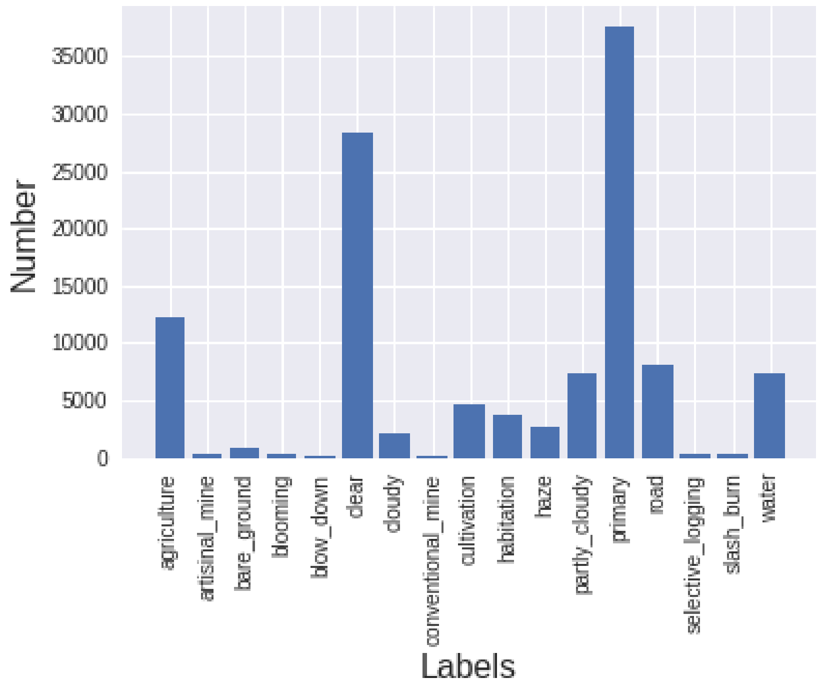
The data provided in Kaggle competition is present in train-jpg.tar.7z and the labels of these images are present in the train.csv file. The dataset consists of around 40479 images which belong to one or more classes among 17 classes. The images are in jpg format, each of shape (256, 256, 4). The color space is RGB + near infrared band. Each image has a unique id and the labels are given in train.csv file. The validation set has been created with a split of 80% from training set.

```
Label: agriculture clear habitation primary road
Shape: (256, 256, 4) Max: 255, Min: 0
```



Distribution of target class

```
{'haze': 2696, 'primary': 37512, 'agriculture': 12314, 'clear': 28430,
'water': 7410, 'habitation': 3659, 'road': 8070, 'cultivation': 4546,
'slash_burn': 208, 'cloudy': 2088, 'partly_cloudy': 7260,
'conventional_mine': 99, 'bare_ground': 861, 'artisinal_mine': 338,
'blooming': 331, 'selective_logging': 339, 'blow_down': 100}
No. of Labels: 17
```



The above plot shows the number of images labelled for each class. We can see that the labels which tell the condition of weather (clouds mostly) have been labelled for most images.

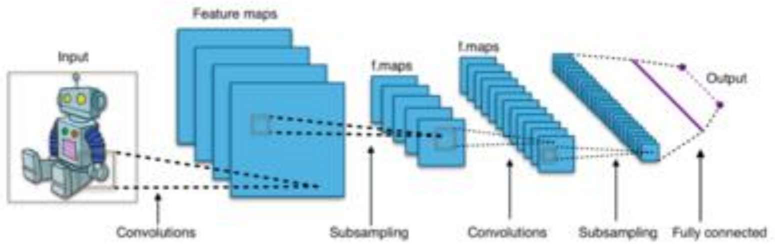
The dataset has no null values. And so far, there has been no reported abnormalities in Kaggle.

Algorithms and Techniques

A neural network is made of input, output layers and many hidden layers. Output of one-layer passes to the next layer; an operation can be done in each layer. Convolutional Neural Network is no different, it consists layers of convolution, maxpooling, dropout and fully connected layers (typical neural network layer)

Below is the brief description of CNN and different layers of CNN as specified in a Wikipedia article.

"In machine learning, a convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery."

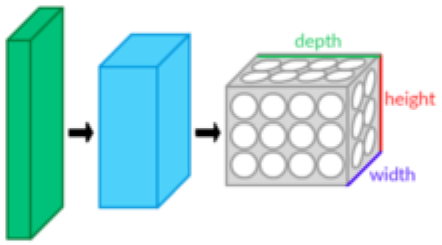


A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers.

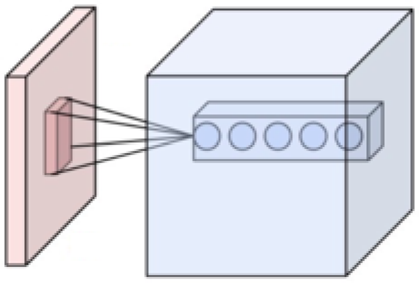
Description of the process as a convolution in neural networks is by convention. Mathematically it is a cross-correlation rather than a convolution. This only has significance for the indices in the matrix, and thus which weights are placed at which index.

Convolutional

Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli. Each convolutional neuron processes data only for its receptive field.



Although fully connected feedforward neural networks can be used to learn features as well as classify data, it is not practical to apply this architecture to images. A very high number of neurons would be necessary, even in a shallow (opposite of deep) architecture, due to the very large input sizes associated with images, where each pixel is a relevant variable.

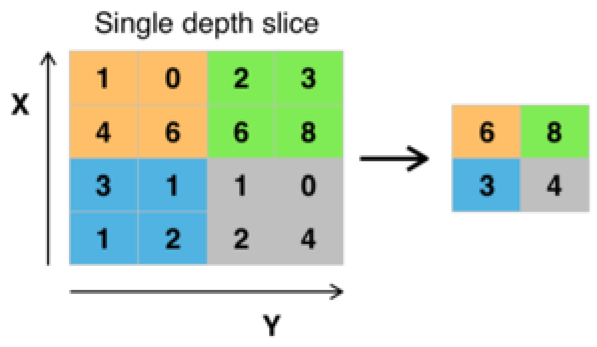


For instance, a fully connected layer for a (small) image of size 100 x 100 has 10000 weights for each neuron in the second layer. The convolution operation brings a solution to this problem as it reduces the number of free parameters, allowing the network to be deeper with fewer parameters. For instance, regardless of image size, tiling regions of size 5 x 5, each with the same shared weights, requires only 25 learnable parameters. In this way, it resolves the vanishing or exploding gradients problem in training traditional multi-layer neural networks with many layers by using backpropagation.

Pooling

Convolutional networks may include local or global pooling layers, which combine the outputs of neuron clusters at one layer into a single neuron in the next layer.

For example, max pooling uses the maximum value from each of a cluster of neurons at the prior layer.



Another example is average pooling, which uses the average value from each of a cluster of neurons at the prior layer].

Fully connected

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP).

Weights

CNNs share weights in convolutional layers, which means that the same filter (weights bank) is used for each receptive field in the layer; this reduces memory footprint and improves performance.

Using these layers, convolutional neural network identifies features from images. One or more layers of each type is used, the model is trained and evaluated on a validation set. Based on the accuracy and loss, the hyper parameters are tweaked.

Benchmark

One kaggler achieved accuracy of 93%. The link to his notebook is: https://gist.github.com/EKami/33ec0172590ab9f2e3a6b757c9f9dcb4

The model was run on 20 epochs and has tried learning rates = 0.01, 0.001.

The objective of this project is to get as close as possible to this model. But it will be considered success if accuracy of 80% is achieved.

III. Methodology

Data Preprocessing

Before building the model, preprocessing of data needs to be carried out. These are the steps that were carried out during this project:

1. The images were resized to 128 x 128 pixels.
2. All three channels were used during training process as the satellite images are color images.
3. The images were normalized by dividing every pixel in every image by 255.
4. Lastly, the images were split into training, validation and test sets.
5. Fortunately, the dataset didn't have any null values or abnormalities so far, this based on the forums in Kaggle.

Implementation

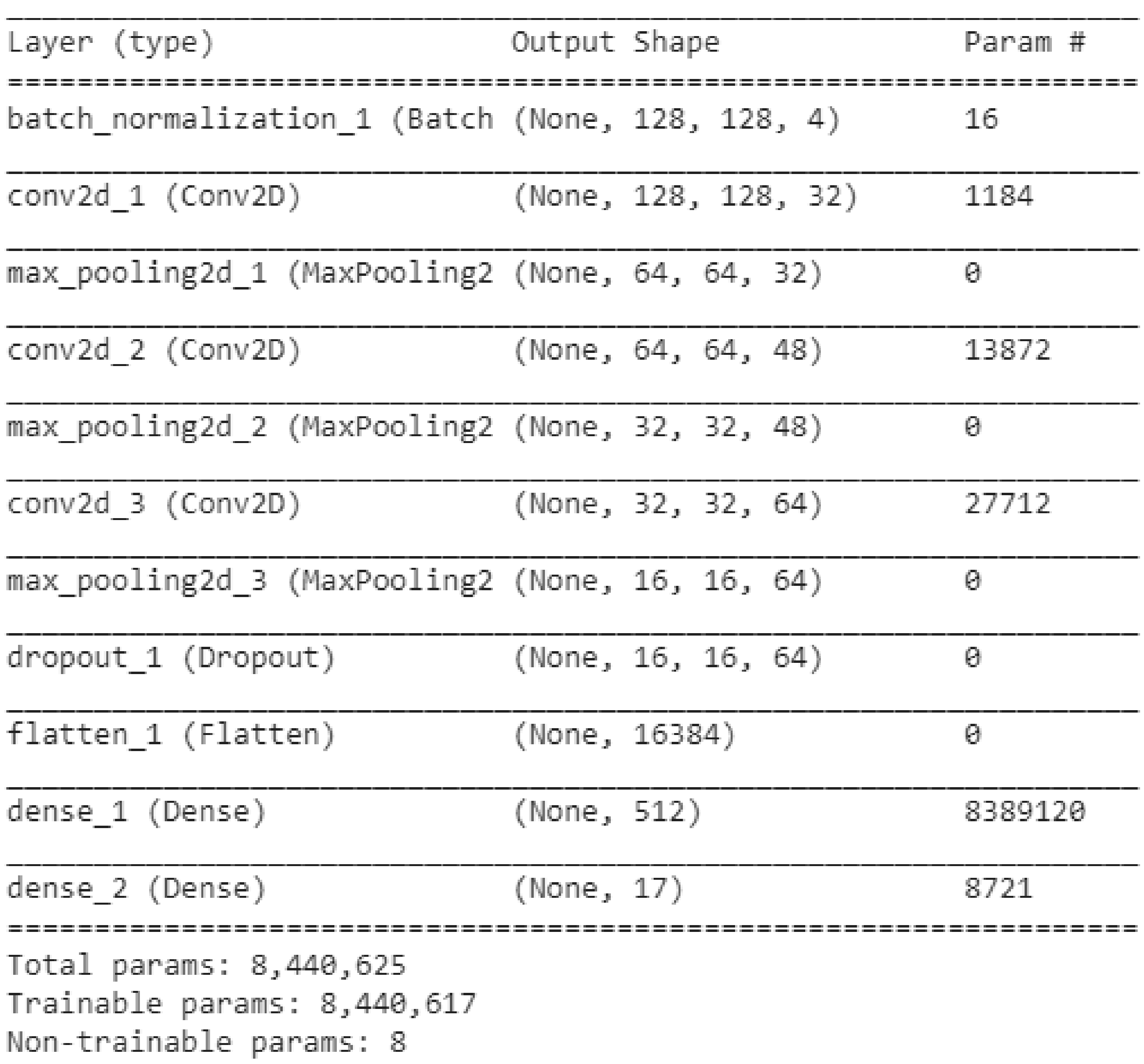
For implementing this project, Google Colab has been chosen. Google Colab offers a customized jupyter notebook which is powered by GPU. Many deep learning models are pre-installed. Firstly, using Kaggle CLI, the dataset has been downloaded into the runtime. The dataset has been preprocessed and then, split into training, validation and test set. Using Keras, CNN model has been built.

Complications that occurred during the implementation are figuring out the inputs for each layer of CNN is more of a trial and error process. It was also important to check how the data was being modified during the pre-processing. Also, going through documentation thoroughly was helpful.

Following are the layers of the CNN model:

1. The entire dataset is pre-processed and split into train, validation and test sets.
2. A simple CNN model is created.
3. First layer is Convolutional layer with inputs 32 and filter size as 3,3
4. Next layer is maxpooling layer with pool size as 2,2
5. It is followed by another Convolution layer of inputs 48 and filter size of 3,3
6. Next layer is maxpooling layer with pool size as 2,2

7. It is followed by another Convolution layer of inputs 64 and filter size of 3,3
8. Next layer is maxpooling layer with pool size as 2,2
9. Till now the layers have used relu as their activation function and padding is ‘same’.
10. Next layer is dropout layer with 0.7 as probability.
11. Next the model as flatten as layer and then, fully connected layer of inputs 512 is present with activation function as relu.
12. The output layer is a fully connected layer with sigmoid as activation function.
13. The model is compiled with Adam optimizer and ‘binary_crossentropy’ as the loss function.
14. The model is trained for 10 epochs with batch size of 128.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
batch_normalization_1 (Batch (None, 128, 128, 4)       16        
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 32)      1184      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 48)        13872     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 48)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 64)        27712     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 64)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 16384)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               8389120   
_________________________________________________________________
dense_2 (Dense)              (None, 17)                8721      
=================================================================
Total params: 8,440,625
Trainable params: 8,440,617
Non-trainable params: 8
```

Refinement

To get the initial result simple CNN architecture was built and evaluated. This resulted in decent loss of around 0.256. Batch normalization was included after couple of runs. The hyperparameters were tuned a couple of times, layers inputs were changed from (16, 32, 48) and (64, 128, 256) to (128, 64, 32). Drop out layer was added to account for overfitting. Training was initially carried out with 20 epochs which was later reduced to 10 epochs because there was no much variation in the loss values. The accuracy on validation set during the initial runs were around 56% and 72%, after changing the layer inputs, accuracy on validation sets increased to 90%, the exact value in the notebook is `0.9054`. Batch size was not changed, using Google Colab for this project helped a lot in the computation aspect of this project.

IV. Results

Model Evaluation and Validation

During model generating and development, a validation set was used to evaluate the model. A test set was also created initially, finally the model was evaluated on this test set. Accuracy of `0.905385` was achieved on this test set. As a validation set was used during training and the data in test set is completely new for the model, it is safe to say that the model will perform great on another new dataset.

At the end of last epoch,

```
Epoch 10/10
 - 39s - loss: 0.2567 - acc: 0.9050 - val_loss: 0.2564 - val_acc: 0.9054
```

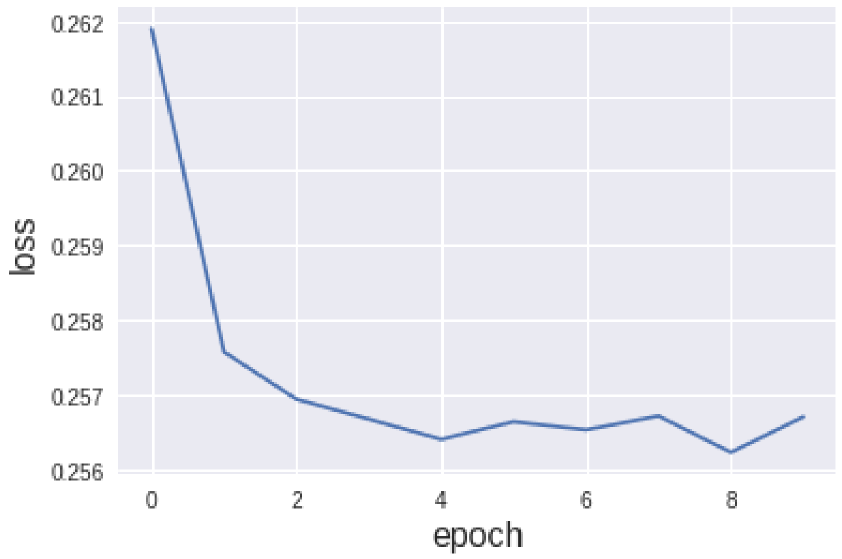
Here, training accuracy is 0.9050 and validation accuracy is 0.9054

Justification

The final solution has been thoroughly analyzed. The results of this project didn't cross the project discussed in Benchmark section. But the final solution is significant enough to have solved the problem. The model mentioned in benchmark had an accuracy of 93%, the accuracy of my model is around 90%. The model has a decent accuracy (90%) based on leaderboard of Kaggle where the accuracy varies from 88% to 94.15%.

V. Conclusion

Free-Form Visualization



This is the loss per each epoch graph. This graph almost looks like the ideal curve expected of loss function.

Reflection

In summary,

1. Google Colab was used to implement this project
2. Dataset was downloaded from Kaggle using Kaggle CLI
3. Data was pre-processed and split into training, validation and test set
4. CNN model was built using Keras
5. Training was done using Adam optimizer and loss function 'binary_crossentropy', validation set was used during training
6. The model achieved 90.54% on the test set

It was a good experience trying this project on Google colab, it provided a great advantage, it helped a lot with computational aspect of this project.

Improvement

Following are areas of improvement,

- TIF images were also provided as additional dataset in the Kaggle competition, training can be improved with more data.
- Also, the satellite images have a fourth band apart from RGB bands, near IR spectrum was included in the dataset, maybe using that might increase efficiency of the model.