Machine Learning Engineer Nanodegree

Capstone Project

Mehdi Khodayari
November 9st, 2017

I. Definition

Project Overview

In this project a labeled set of images is used to develop an algorithm which can automatically predict if there is iceberg(s) in new observations or not. These images are collected by the Sentinel-1 satellite and are used to find icebergs in areas like offshore of the East Coast of Canada. The following picture shows the Sentinel-1 satellite.

Illustration 1: the Sentinel-1 satellite

The moving icebergs present threats in that area and this is why locating them is highly important.

Illustration 2: a drifting iceberg which present threats to navigation and activities

The images received from the satellite can show solid objects which can be either icebergs or ships. In fact, the categorization of the images requires manual interpretation of the images and here we want to use a computer algorithm for the image categorization, which ultimately can be used to predict drifting icebergs.

Problem Statement

The problem here is that the images received from the satellite requires interpretation and cannot be directly used to predict if the object is iceberg or not (this is a binary classification). The following figure shows a set of images received from the satellite.

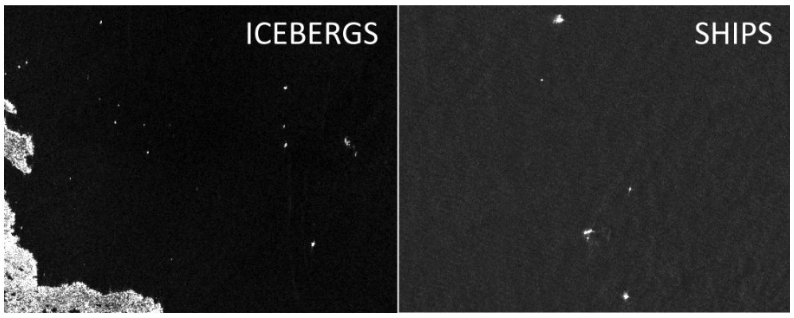


Illustration 3: images of iceberg and ships received from the Sentinel-1 satellite

However, there are some subtle differences between these two images that are not obvious to human eyes but can be detected by computers with the aid of right tools (algorithms).

To better understand the characteristics of the images received from the satellite, here we explain how these images are collected. The Sentinel-1 satellite sends a signal to an object and collect the echo referred to as backscatter which is then converted to an image. The backscatters from any solid objects such as ship, iceberg, and land is stronger than that from water, making it possible to distinguish between solid objects and water. In fact, solid objects in the resultant images appear brighter than their surrounding (water). The emitted signals from the satellite always hit the objects horizontally, but the backscatters are received either horizontally or vertically. Hence, two sets of information (channels of images) are obtained from objects; the first set, here referred to as HH (transmit/receive horizontally), corresponds to the horizontal backscatters and the second set, referred to as HV (transmit horizontally and receive vertically), corresponds to the vertical backscatters. In the following figure, two samples (each with two channels of HH and HV) of the datasets are shown.

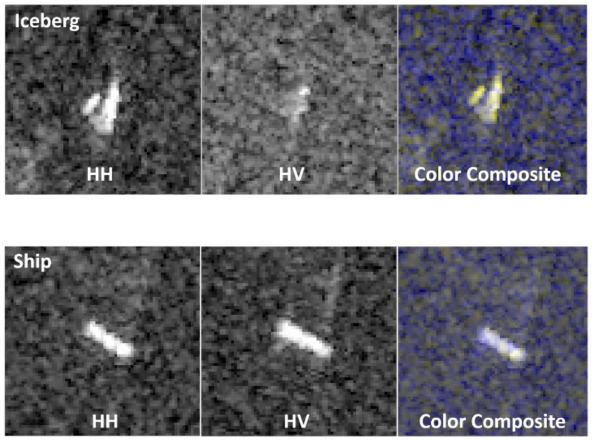


Illustration 4: images received from the satellite with the HH and HV channels

As obvious, the HH images of both iceberg and ship show bright solid objects. However, the HV images differ significantly; the ship HV image shows a bright object as well as its HH image does, but the iceberg HV image hardly shows the presence of a solid object. Here, we try to construct a Convolutional Neural Network (CNN) that can capture this difference.

Metrics

The dataset (images) are split into the training and validation datasets and the validation *accuracy* is used as the model performance metric. The validation *accuracy* is the ratio of the number of images correctly classified over the total number of images in the validation dataset. There are other metrics such as *log loss* and *F1 Score*. The *log loss* metric has too much emphasis on the prediction probabilities and *F1 Score*, which is a harmonic average of precision and recall, has more emphasis on positive predictions and requires some sort of interpretation. However, in this project, the objective was to obtain an optimal model that has a higher number of true predictions (positive and negative), and hence accuracy was chosen over *log loss* and *F1 Score* metrics.

While training the model, whenever the validation accuracy is improved the model parameters (weights) are saved and the last saved weights will be used to construct an optimal model.

II. Analysis

Data Exploration

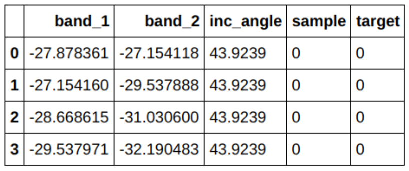
This project is an ongoing kaggle competition (at the time of this capstone project submission) and the data is given in json format. There are train (there are 1604 images in this dataset) and test datasets. The datasets have the following fields:

1. id: the image id

2. band_1, band_2: the flattened image data corresponding to HH and HV respectively each with 5625 elements (75x75 pixels)

3. inc_angle: the incident angles of which the image was taken

4. is_iceberg: the train dataset labels which is either 1 (if there is iceberg(s) in the image) or 0 (if the detected objects corresponds only to Ship(s))



	band_1	band_2	inc_angle	sample	target
0	-27.878361	-27.154118	43.9239	0	0
1	-27.154160	-29.537888	43.9239	0	0
2	-28.668615	-31.030600	43.9239	0	0
3	-29.537971	-32.190483	43.9239	0	0

Illustration 5: the pandas dataframe head of the training dataset

The band_1 and band_2 fields have dB unit. The is_iceberg field only exists in the train dataset. The following figure shows four first values of the band_1, band_2, inc_angle, and is_iceberg (target) fields of the first image (sample 0) of the training dataset.

There are also 851 samples with 0 labels (target=0) and 753 samples with 1 labels (target=1). The following bar chart shows the distribution of the target. Hence, the dataset is balanced.

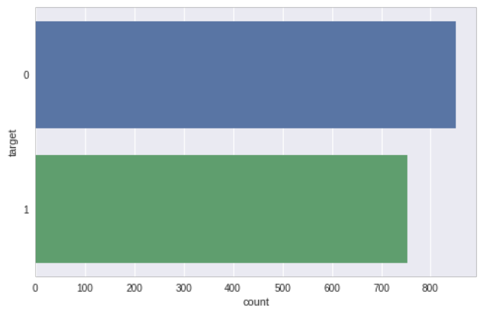


Illustration 6: target distribution

Exploratory Visualization

In this section, to better understand the dataset characteristics, the histograms of the band1, band2, and inc_angle fields are shown in the following figure. These histograms help us to better understand the differences between the band1 and band2 values and also see how the inc_angle distribution is.

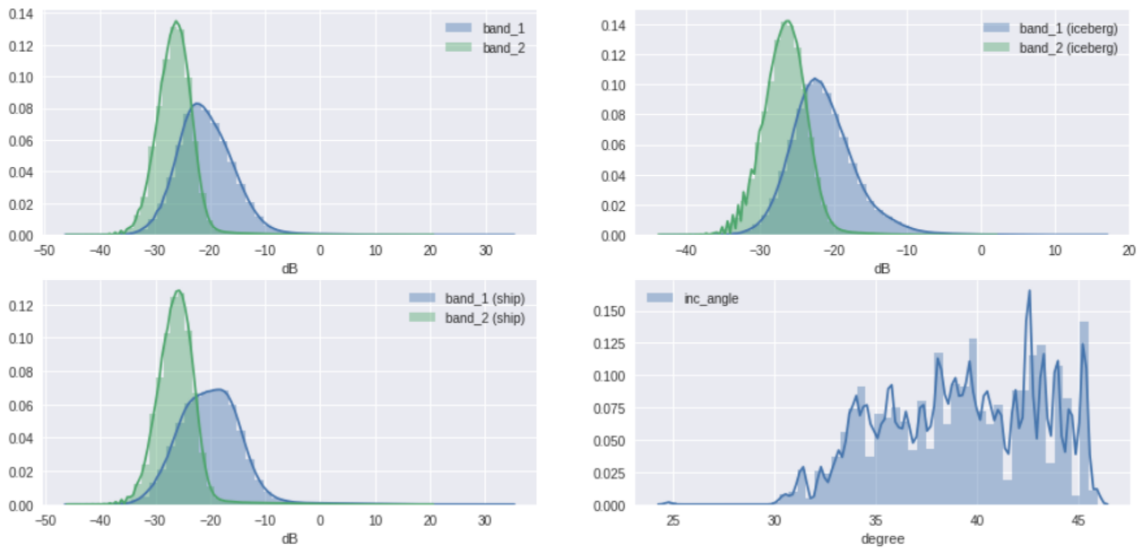


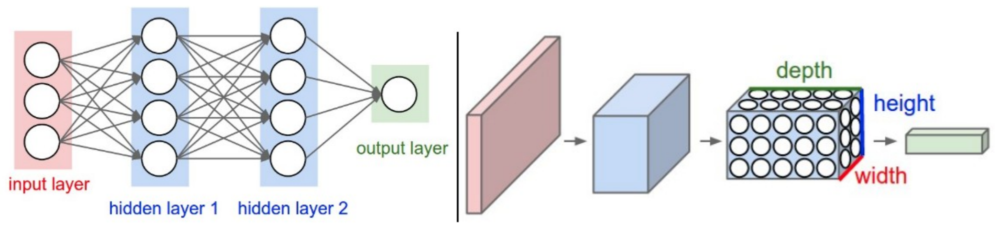
Illustration 7: histograms of the band1, band2, and inc_angle

As obvious from the distributions, the band_1 values tend to be higher than the band_2 values and there no abnormalities in these distributions. However, the inc_angle values do not form a normal distribution.

Algorithms and Techniques

In this project, Convolutional Neural Network (CNN) architectures (which have been successfully applied for image classification problems) are employed for the binary classification of iceberg images. The models are initiated by assigning random numbers to the models weights.

CNNs are similar to regular neural networks, in the sense that they have neurons (and their associated weights), activation functions, and they also train the same way through feed-forward/back-propagation iterations. However, due to the shape of their neurons (typically with three dimensions), the CNN architectures can scale to large images; this is not practically feasible with the regular neural nets for large images because the number of parameters gets too high, so they either have high computational cost and/or would suffer from an overfitting problem. In the following graph a comparison between a regular neural network and a CNN is shown.



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

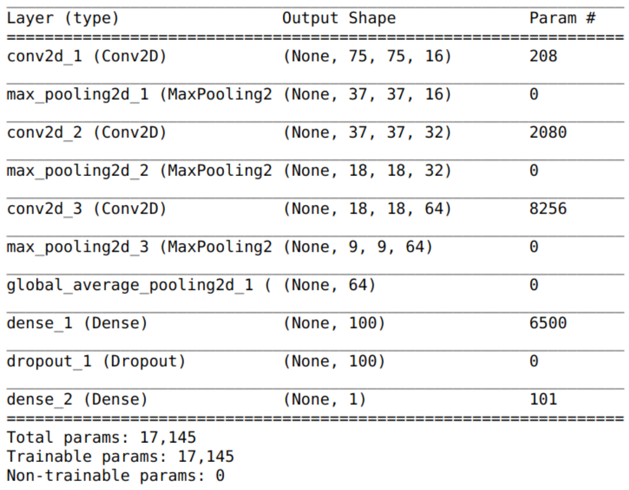
Illustration 8: taken form http://cs231n.github.io/convolutional-networks/

Typically, the CNN architectures consist of convolutional, pooling, and fully-connected layers. In the convolutional layer, the dot products of the neuron weights with the values of the pixels (each neuron is first connected to a small region of the image and then slides over the width and hight of the image) are computed and fed into activation functions (here, ReLu) and the computed output is a representation (filter) of the image; this process is repeated with different neurons holding different weights (the number of neurons is equal to the number of filters and this number is set when the model is built at each convolutional layer) that map the same image into different filters. In the pooling layer, a downsampling is implemented by averaging or finding the maximum of small regions of the filters and reducing the size of filters. For example, an image with a size of [32x32x3] can be first mapped into a space with the size of [32X32X12] at the convolutional layer (12 here is the number of filters) and then its size can be reduced to [16x16x12] at the pooling layer. The downsampling process at the pooling layer helps to minimize the number of parameters and overfitting problem. After the last convolutional layer, a Global Average Pooling (GAP) layer is typically used which maps the last set of filters into a one dimensional array by computing the average of all the values in every filter and returning an array with a length that is equal to the number of filters (the GAP layer is beneficial for the model generalization). Then, the resultant vector is fed into a fully-connected layer(s) that are similar to the hidden layers in the regular neural nets and compute the probabilities of classes (the last dense layer has to have a sigmoid activation function for binary classification problems). We can also add a dropout layer after each fully-connected layer in order to randomly removing some of the outputs of those layers to minimize overfitting.

Benchmark

We use a relatively simple CNN model as benchmark and try to see what happens if we add to the complexity of the benchmark model by adding more convolution layers.

The following image shows the benchmark CNN model architecture. In this architecture, three convolution layers with 16, 32, and 64 filters (2x2) are used. Between every two convolution layers, we use a max-pooling layer to minimize overfitting and computational cost by reducing the number of parameters. After the last max-pooling layer, a global average pooling (GAP) is used to minimize overfitting as well (*reference*). Then, the output vector is fed into a dense layer with 100 nodes and finally the results are passed to a dense layer with one node and sigmoid activation function which returns the classes probabilities. All the convolution and dense layers, except the last dense layer, have ReLU activation function.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 75, 75, 16)        208
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 37, 37, 32)        2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 18, 18, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 18, 18, 64)        8256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 9, 9, 64)          0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               6500
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 17,145
Trainable params: 17,145
Non-trainable params: 0
```

Illustration 9: the benchmark architecture

III. Methodology

Data Preprocessing

The provided dataset contains band_1 and band_2 data in the form of two separate one dimension arrays. However, in order to feed this information into CNN models, a data structure with the dimension of (1, width, height, number of channels) is required. Hence, we used the numpy.reshape method to convert the dimension of the original band_1 and band_2 arrays into (1, 75, 75, 2). Here there are two channels that represent the band_1 and band_2 arrays. Hence, in order to be able to feed the data into the ResNet50 architecture (to compute bottleneck features), we added a third channel to the data set which is simply the replicate of the second channel (band_2), after which each sample dimension turned into (1, 75, 75, 3). Because the pixel values are in dB unit, we rescaled their ranges to 0 to 255. The convert_data function available in the iceber_ship.ipynb file in the project repository does the data conversion task in this project.

Implementation

For all of the models, the optimizer used was *adam*. A ModelCheckpoit callback was also integrated into the model fitting process in order to save the best model during training. The best model is selected based on the validation dataset accuracy. A learning rate of 0.00015 was chosen for this model and training was completed after 200 epochs. The loss function for the model optimization was the binary_crossentropy which is commonly used for deep binary classification problems. After each model training the accuracy values variations for both the training and validation datasets are plotted. The following figure shows the corresponding plot for the benchmark model.

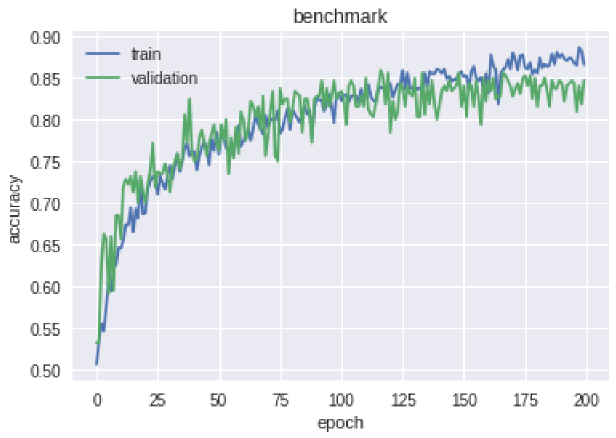


Illustration 10: accuracy variations of the training and validation datasets

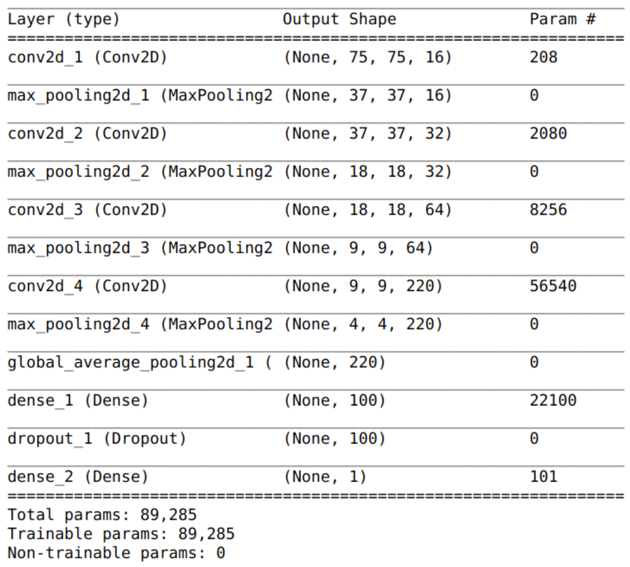
The validation accuracy is also computed after the model training is complete. To do so, the same model is reconstructed and the best weights, that were saved during training, are loaded into the model. Then, the validation accuracy is computed via model.evaluate method.

The packages used in this project are as follows:

1. *keras* with tensorflow backend
2. *json* to load the provided datasets
3. *matplotlib* and *seaborn* to plot the data analyses and the prediction results
4. *pandas* to convert the data to pandas dataframes
5. *numpy* to implement all the arrays manipulations

Refinement

After evaluating the benchmark model performance, one more convolution layer was added to the benchmark model. The following figure shows the architecture of the refined model.

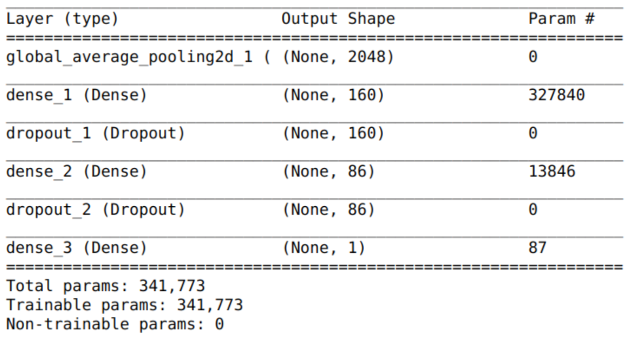


```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 75, 75, 16)        208
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 37, 37, 32)        2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 18, 18, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 18, 18, 64)        8256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 9, 9, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 9, 9, 220)         56540
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 220)         0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 220)               0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               22100
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 89,285
Trainable params: 89,285
Non-trainable params: 0
```

Illustration 11: architecture of the refined model

The optimizer and epochs are as those in the benchmark model, *adam* and 200.

In the last model, first bottleneck features of the ResNet50 architecture were extracted and saved. Then they were fed into a model with three fully connected dense layers. The following figure shows this model architecture.



```
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_1 ( (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 160)               327840
_________________________________________________________________
dropout_1 (Dropout)          (None, 160)               0
_________________________________________________________________
dense_2 (Dense)              (None, 86)                13846
_________________________________________________________________
dropout_2 (Dropout)          (None, 86)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 87
=================================================================
Total params: 341,773
Trainable params: 341,773
Non-trainable params: 0
```

Illustration 12: last architecture

For this model, we used *adam* optimizer with a learning rate of 0.0003 and the model training was implemented over 100 epochs. We achieved a validation accuracy of 0.863 which is similar to the benchmark validation accuracy.

IV. Results

Model Evaluation and Validation

The following bar chart shows the validations accuracies of the three models.

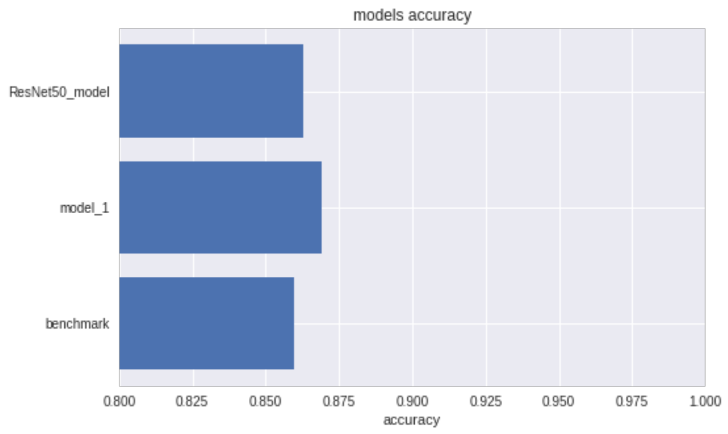


Illustration 13: validation accuracies

The model_1 architecture has the best validation accuracy which is 0.87. In this model we used three convolutional layers with 16, 32, 64, and 220 filters; after each convolutional layer a MaxPooling layer was used and, after the last MaxPooling layer, a GlobalAveragePooling layer was also used. The output of the GlobalAveragePooling layer was fed into a dense layer with 100 nodes, the output of which fed into a dense layer with one node and a *sigmoid* activation function.

We also predicted the category of the test images, which scored 0.2548 (log loss) in the kaggle Leaderboard and ranked around 500 among 1000 contributors.

Justification

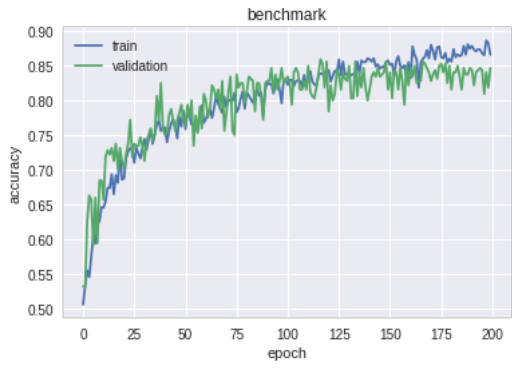
The validation accuracy of the benchmark model, ResNet50-based-model, and refined model are respectively 0.860, 0.863, and 0.869. Before documenting the final results, we tried to optimize the performance of these models by changing different parameters such as learning rate, number of epochs, changing the number of nodes and filters in the dense and convolution layers, and changing the number of layers; overall, the refined model can outperform other models and then the ResNet50-based-model is better than the benchmark.

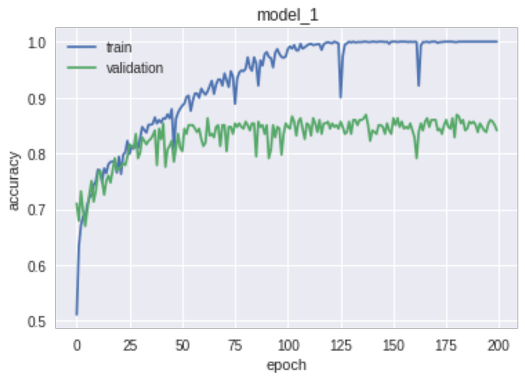
V. Conclusion

In conclusion, the refined model has performed better than the other models and the results indicate that this model performance can be yet improved by using other techniques.

Free-Form Visualization

The accuracy variation behaviors of the benchmark and refined models (as shown in the following figures) indicate that the benchmark model is not complex enough so the train accuracy hardly improves over epochs; on the other hand, the refined model (model_1) shows some behaviors that is indicative of over fitting. In fact, the model_1 has high learning capacity and overfitting is very probably happening due to the small size of the training dataset.





Reflection

The provided dataset is reshaped using numpy.reshape method, after which the dataset can be fed into CNN models. Then, the dataset is divided into training and validation datasets. Three CNN models, namely benchmark, model_1, and ResNet50-based-model, are constructed. The model_1 architecture has more convolution and dense layers than the benchmark does. In the ResNet50-based-model, bottleneck features of the ResNet50 architecture are incorporated and fed into a deep neural network with three fully connected dense layers. The models are trained with the train dataset and their performance is evaluated using the validation accuracy.

The model_1 architecture showed better performance than the other models. However, the accuracy variation plots indicate that the model_1 suffers from overfitting problem even though its performance is better than the other models and the benchmark is possibly not complex enough for this problem.

One interesting aspect of this project is that if we make the model more complex than model_1 by adding more convolutional layers, the validation accuracy score can decrease. Hence, we tried to find and optimal complexity for the models and the model_1 architecture is the optimal structure we found for this project and for the approach we have taken.

Improvement

Even though model_1 has shown better performance than the other models, its performance can be possibly improved by using a technique like *image augmentation* (even though it has not been incorporated in this project) to minimize the overfitting problem caused by the small size of the training dataset.