INTERNSHIP REPORT

IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

Under the guidance of Dr Akshay Pai and Dr Fabian Giseke

At the Image Group, University of Copenhagen

From May 2017 to July 2017

Submitted By:
Kanakvi Aggarwal

Objective:
To classify transient images of the sky as instances of a supernova or of a bogus object using Convolutional Neural Networks developed using Python and Keras with TensorFlow backend.

With the increase in the amount of data to be processed and subsequently classified, it is highly desirable to eliminate the need of manual labour and automate the task. Deep Learning utilises a multi-layer structure of non-linear processing units (neurons) to learn features from pre-existing classified dataset and then make predictions on new, unseen data. We used a specific type of neural network called Convolutional Neural Network for our learning problem.

Motivation to use Convolutional Neural Networks (CNNs):
CNNs are developed to specifically handle image data to make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.
Regular Neural Networks receive an input (a single vector), and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the output layer and in classification settings it represents the class scores. In case of images, it is observed that full connectivity is wasteful and the huge number of parameters quickly lead to overfitting.
On the other hand, in CNNs, the neurons in a layer will only be connected to a small region of the layer before it which greatly reduces the number of parameters to be learned by the network. By the end of the CNN architecture, we reduce the full image into a single vector of class scores arranged along the depth dimension.

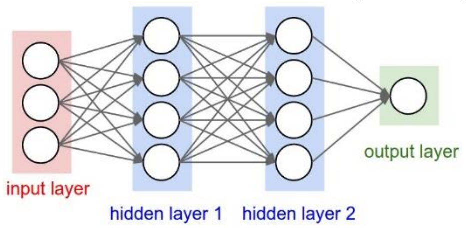


Figure 1 Regular Neural Network with two Fully Connected Hidden layers

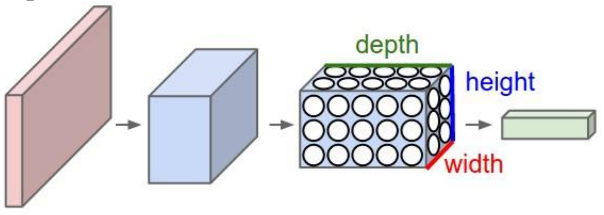


Figure 2 Convolutional Neural Network (CNNs) Architecture

Typical Architecture of a CNN:
A simple CNN is a sequence of layers, and every layer of a CNN transforms one volume of activations to another through a differentiable function. We use three main types of layers to build CNN architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer.

Overview of the layers used:

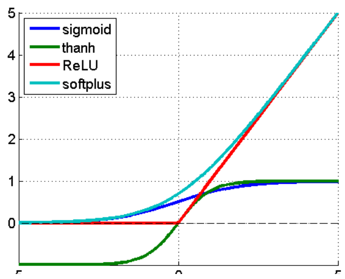


Figure 3 Plots of some commonly used activation functions

- *Convolution (CONV) Layer* computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.
- *Activation Layer* introduces non-linearity in the network, so that it can learn and represent arbitrary complex functions. It is particularly important when the network is handling complicated data such as images, videos,

audio and speech. The main purpose is to convert an input signal of a neuron into an output signal by applying a fixed elementwise activation function.
A commonly used activation function is Relu (Rectified Linear Units): max(0,x).

- *Pooling (POOL) Layer* is periodically inserted in-between successive Convolutional layers in a CNN architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation, in case of Max Pooling. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. Every MAX operation would in this case be taking a max over 4 numbers. The depth dimension remains unchanged.
- *Fully Connected (FC) layer* will compute the class scores. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

The CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). The parameters in these layers will be trained with gradient descent so that the class scores that the CNN computes are consistent with the labels in the training set for each image. A measure of discrepancy between the predictions of the network ($f(x_i)$) and the desired output (y) is the loss function. The objective of the learning problem is then to minimize the loss function. For binary classification problems, logistic loss and hinge loss are commonly used.

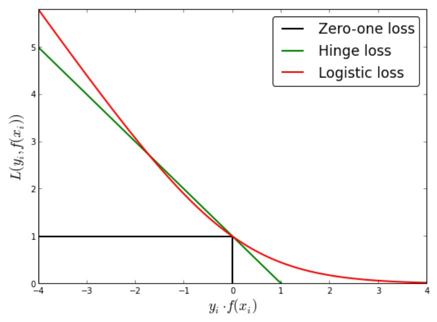


Figure 4 Plots of Common Loss Function

It is also necessary to ensure that the depth of the network satisfies the problem at hand. A very deep a network would lead to overfitting in case of simple tasks, whereas a shallow network might not to able to adapt to the learning task at hand and thus, underfit the data. The process of fine-tuning the network hyperparameters after deciding the structure involves a hit-and-trial approach.

Classification problems utilise feedforward type of neural network, that is, the flow of information is unidirectional. There are no feedback loops and a unit sends information to a unit from which it does not receive any information. Neurons in one layer are also not connected to each other.

Typically used layer pattern:
The most common form of a CNN architecture stacks a few CONV-RELU layers, follows them with POOL layers, and repeats this pattern until the image has been merged spatially to a small size. At some point, it is common to transition to fully-connected layers. The last fully-connected layer holds the output, such as the class scores.

Input to the network:
The input layer was comprised of sets of three input images:

- *The template image* which was taken long before the detection an activity in the sky

- *The target image* which captured the activity
- *The difference image* which is obtained on pixel-by-pixel subtraction of template image from the target image, followed by a few correction techniques. The difference image thus captures the transient events and removes any constant objects.

The input images can thus capture either real instances of supernovae or of a bogus object such as a satellite, a result of bad image processing or atmospheric effects. The training set contains 2,237 instances (2,010 "bogus" and 227 "real" instances) and the test set 2,169 (1,942 "bogus" and 227 "real" instances).

The network used:
Similar to the typical CNN architecture, in our network, we used stacks of 2D Convolution layer, 2D MaxPooling layer, LeakyReLU activation function layer and Dropout layer, followed by stacks of Dense (fully connected) layer, Leaky ReLU activation function layer and Dropout layer. The loss function used was Binary Cross Entropy and the optimizer used was Adam.

Layers in the network:
Input Layer → Convolution2D_1 → MaxPooling2D_1 → LeakyReLU_1 → Dropout_1 → Convolution2D_2 → MaxPooling2D_2 → LeakyReLU_2 → Dropout_2 → Convolution2D_3 → MaxPooling2D_3 → LeakyReLU_3 → Dropout_3 → Convolution2D_4 → MaxPooling2D_4 → LeakyReLU_4 → Dropout_4 → Flatten → Dense_1 → LeakyReLU_5 → Dropout_5 → Dense_2 → LeakyReLU_6 → Dropout_6

Apart from the MaxPooling layers, dropout layers are also used to prevent overfitting in which a neuron is only kept active with a fixed probability given as the hyper parameter.

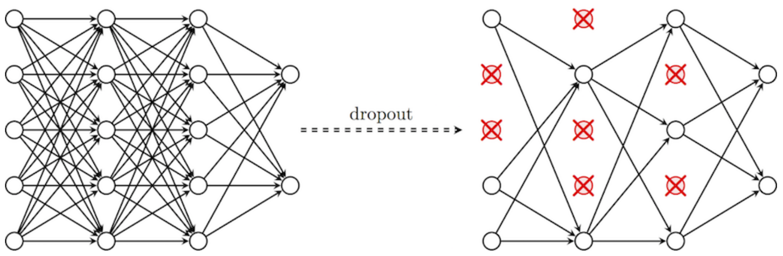


Figure 5 Working of Dropout Layer

The activation function used is Leaky ReLU, which was observed to work better. Leaky ReLU is an attempt to fix the dying neuron problem, that is, the neuron always outputs zero irrespective of the input due to learning a large negative bias. Once a ReLU ends up in this state, it is unlikely to recover, because the function gradient at 0 is also 0, so gradient descent learning will not alter the weights. Leaky ReLU with a small negative slope when x<0 provides a chance to recover.

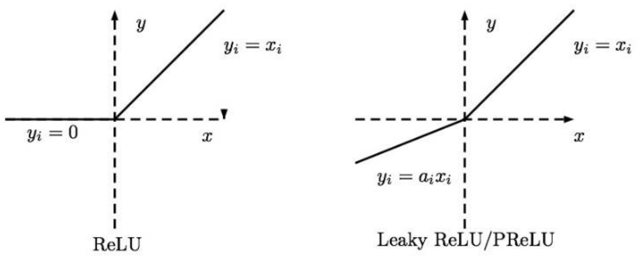


Figure 6 Plot of ReLU and Leaky ReLU activation functions

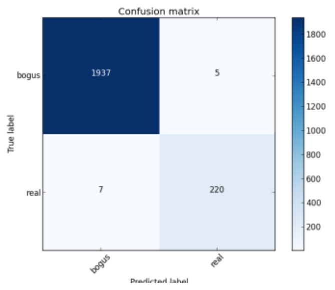
The number of "bogus" cases outnumber the "real" cases. There are approximately 8 times more "bogus" cases than "real" cases. Thus the dataset is a skewed one and a good metric for evaluating the accuracy of the network is to plot the confusion matrix.

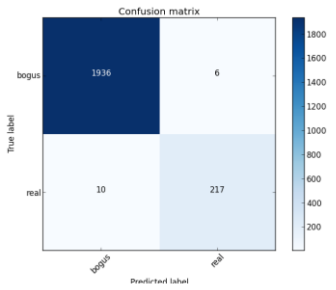
Results:
Without using any pre-processing step or data augmentation, the network achieved an accuracy of 99.32%, that is, 15 misclassifications out of a total of 2169 in the test set. In order to improve the accuracy, several techniques were tried:

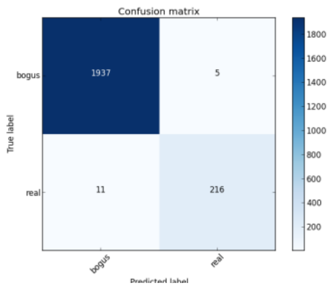
- A data generator was used for data augmentation of both “real” and “bogus” cases. It is observed that it lowers the accuracy of the network to 98.294%.
- Next we tried the augmentation of only “real” cases by randomized flipping, rotation and cropping of images. The accuracy of the network was slightly less than the one with no data augmentation or pre-processing step at 99.262%.
- A simple Variational Autoencoder network was used to remove noise from the images, which was observed to not give good results.

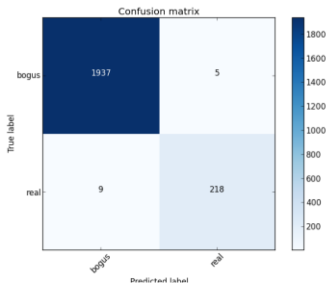
The accuracy remained unaffected or worsened due to data augmentation, possibly because the neural network was unable to learn any new features and because the new images generated using data generator were not a true representation of the the original cases. A major reason that affects the accuracy is the presence of noise in the images. Physical conditions such as moonlight affected the quality of the images and determines the amount of noise. Furthermore, the dataset is skewed. There are approximately 8 times more bogus training examples than real. Thus, there is a high probability that in the test dataset there are real cases which are fairly different in representation from all the cases seen by the network earlier leading to a higher number of misclassifications.

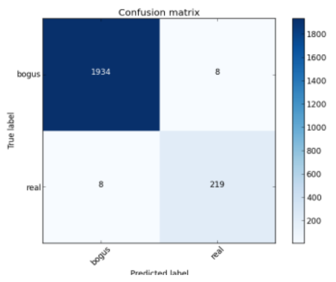
Confusion Matrices:

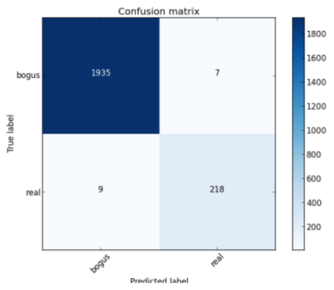












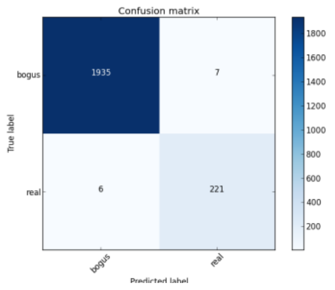


Figure 8 Confusion matrices obtained in case of no data augmentation. The matrices show that the network misclassifies 15 images out of a total of 2,169, equating to an accuracy of 99.37% on

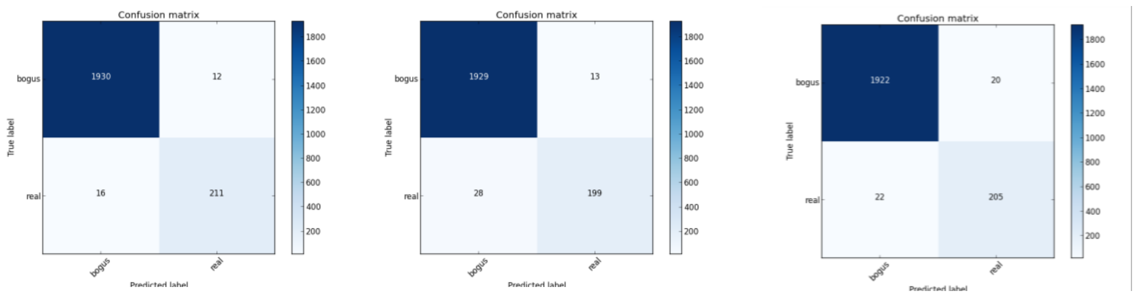


Figure 9 Confusion matrices obtained in case of using data generator for both "real" and "bogus" cases in the traning data set. The accruacy lowers down to 98.294%.

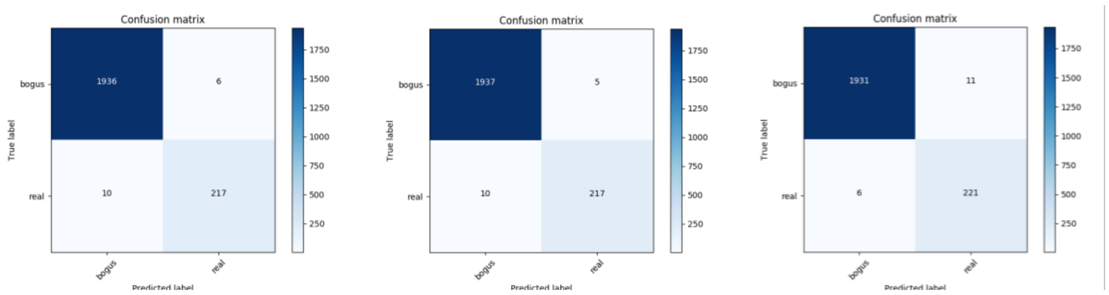


Figure 10 Confusion matrices obtained in case of data augmentation of only "real" cases in the training dataset. The network consequently misclassifies 6 out of 2169 images equating to an accuracy of 99.262%.