Graffiti Detection Using Deep Learning

Sadia Yousafzai   
MS Software Engineering  
San Jose State UniversityCalifornia,USA  
sadia.yousafzai@sjsu.edu

line 1: 4th Given Name Surname  
line 2: *dept. name of organization*  
*(of Affiliation)*  
line 3: *name of organization   
(of Affiliation)*line 4: City, Country  
line 5: email address or ORCID  
  
Ali Saeidi Ashtiyani  
MS Computer EngineeringSan Jose State University  
California, USA  
ali.saeidiashtiyani@sjsu.edu

line 2: *dept. name of organization   
(of Affiliation)*  
line 3: *name of organization   
(of Affiliation)*line 4: City, Country  
line 5: email address or ORCID

***Abstract*—This electronic document is a “live” template and already defines the components of your paper [title, text, heads, etc.] in its style sheet. *\*CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract*. (***Abstract***)**

***Keywords—component, formatting, style, styling, insert (****key words****)***

# Introduction (*Heading 1*)

This template, modified in MS Word 2007 and saved as a “Word 97-2003 Document” for the PC, provides authors with most of the formatting specifications needed for preparing electronic versions of their papers. All standard paper components have been specified for three reasons: (1) ease of use when formatting individual papers, (2) automatic compliance to electronic requirements that facilitate the concurrent or later production of electronic products, and (3) conformity of style throughout a conference proceedings. Margins, column widths, line spacing, and type styles are built-in; examples of the type styles are provided throughout this document and are identified in italic type, within parentheses, following the example. Some components, such as multi-leveled equations, graphics, and tables are not prescribed, although the various table text styles are provided. The formatter will need to create these components, incorporating the applicable criteria that follow.

# Related Work

## *Selecting a Template (Heading 2)*

First, confirm that you have the correct template for your paper size. This template has been tailored for output on the A4 paper size. If you are using US letter-sized paper, please close this file and download the Microsoft Word, Letter file.

## *Maintaining the Integrity of the Specifications*

The template is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them. You may note peculiarities. For example, the head margin in this template measures proportionately more than is customary. This measurement and others are deliberate, using specifications that anticipate your paper as one part of the entire proceedings, and not as an independent document. Please do not revise any of the current designations.

# Our Approach

Our approach consists of implementing four models namely Faster RCNN with VGG16, ,Before diving deep into models, let’s discuss our data preprocessing method.

# 1).Human Graffiti

## *Training Dataset and Preparation*

We have prepared data set of 1000 images from google search engine with manually checking each one if it contains a person or not and then if it is a graffiti or not. Out of these 400 images were positive samples that contained human graffiti and other 400 images were negative samples that did not contain human graffiti and 200 images were reserved as test set. The train-test split ratio is 80:20. The main challenge faced was that it was very difficult to differentiate between graffiti or street art by an artist.

First, we rescaled the images to 200x200px keeping the aspect ratio constant. That will allow the deep learning model to train faster and produce more accurate results. After that, I annotated them manually using rectangular bounding box and creating json file for annotations details. The annotation contained label,xmin,xmax,ymin and ymax co ordinates which will be used by deep learning model. After that, we created json\_to\_csv converter which will create a compiled csv sheet of image annotations to be fed to deep learning model. Labelme screenshot is shown in figure below.

****

*Labelme tool*

Another big challenge in the data set was the rotation and angle of pictures that was taken from the camera. Our initial deep learning model was unable to detect that. So we applied data augmentation technique which generates new images for training and hence improving the model performance.There are various data augmentation techniques, the following are used in this approach :

1. Horizontal flip

2. Scaling

3. Translation

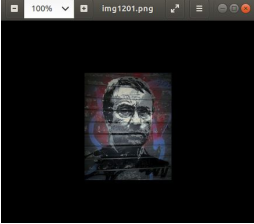
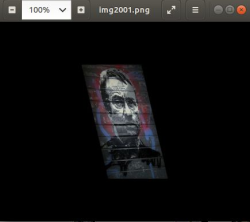
4. Rotation

5. Shearing

6. Various combinations of above

Data augmentation is different for object detection when compared to image classification. In image classification you only need to change the image accordingly, but in object detection you need to change and adjust the bounding boxes as well. For example, if you are flipping the image horizontally then you also need to flip the bounding boxes.

Example of sheared augmentation is given below.

** **

*Normal Image*  *Sheared Image*

For data validation, please use the link below:

*https://drive.google.com/open?id=1jg6KzzBYwqp2OYByG8\_gQ\_XIyvI-iu01*

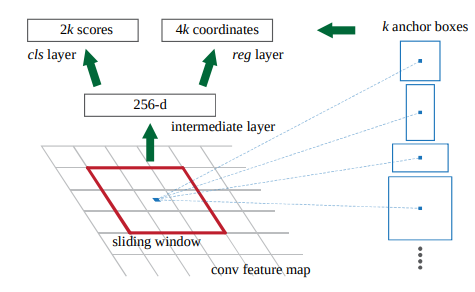
## *Deep Learning Models*

Let’s discuss about each about Faster RCNN(one from each team member) alongside with object detection, feature extraction and machine learning models and adaptive neural schema and convolutional layers of depicted models one after one in the below context; which are then applied on the existing dataset.

***Faster RCNN-Regional Proposal Network:***

A Region Proposal Network (RPN) takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an objectness score.1 We model this process with a fully convolutional network [1], which we describe in this section. Because our ultimate goal is to share computation with a Fast R-CNN object detection network [2], we assume that both nets share a common set of conv layers.

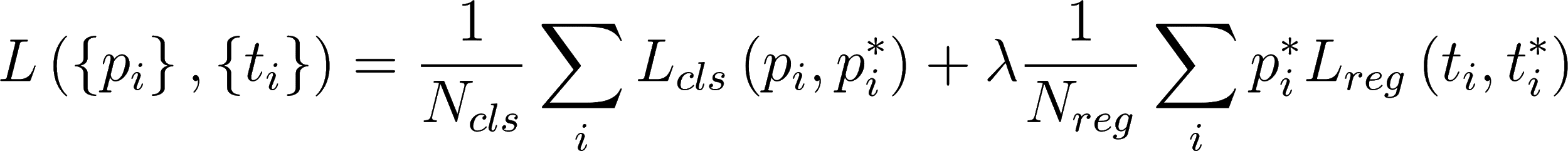
To generate region proposals, we slide a small network over the conv feature map output by the last shared conv layer. This network is fully connected to an n × n spatial window of the input convolutional layer. This vector is fed into two sibling fully-connected layers—a box-regression layer (reg) and a box-classification layer (cls).This mininetwork is illustrated at a single position in Fig. 1.3 . Note that because the mini-network operates in a sliding-window fashion, the fully-connected layers are shared across all spatial locations. This architecture is naturally implemented with an n × n conv layer followed by two siblings 1 × 1 conv layers (for reg and cls, respectively). ReLUs [3] are applied to the output of the n × n conv layer [4].

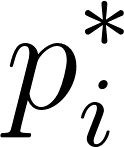
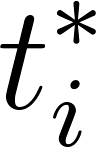
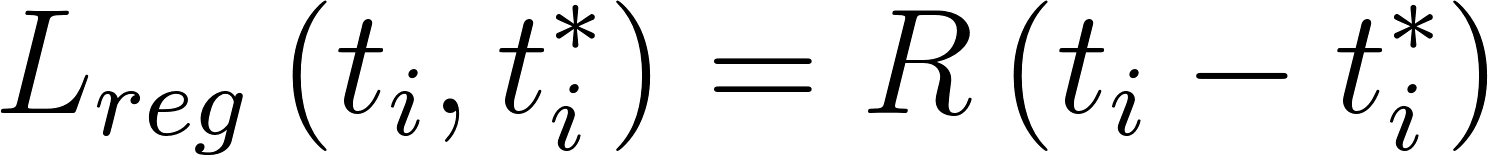
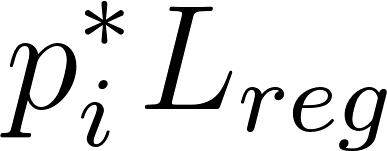
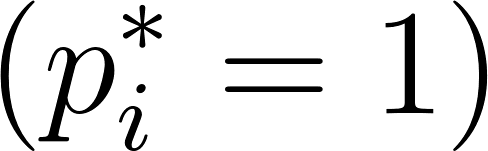
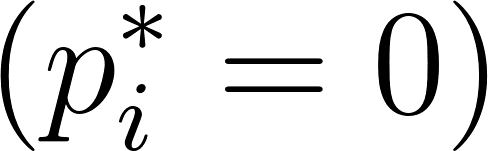
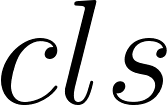
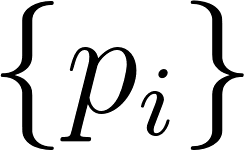
****

*Regional Proposal Network-RPN*

*Translation-Invariant Anchors:* At each sliding-window location, we simultaneously predict k region proposals, so the reg layer has 4k outputs encoding the coordinates of k boxes. The cls layer outputs 2k scores that estimate probability of object / not-object for each proposal.2 The k proposals are parameterized relative to k reference boxes, called anchors. Each anchor is centered at the sliding window in question, and is associated with a scale and aspect ratio. We use 3 scales and 3 aspect ratios, yielding k = 9 anchors at each sliding position. For a convolutional feature map of a size W ×H (typically ∼2,400), there are W Hk anchors in total. An important property of our approach is that it is translation invariant, both in terms of the anchors and the functions that compute proposals relative to the anchors [4].

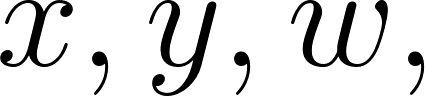
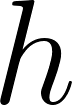
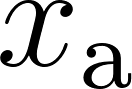
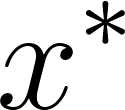
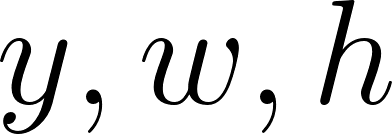
*Loss Function:* For training RPNs, we assign a binary class label (of being an object or not) to each anchor. We assign a positive label to two kinds of anchors: (i) the anchor/anchors with the highest Intersectionover-Union (IoU) overlap with a ground-truth box, or (ii) an anchor that has an IoU overlap higher than 0.7 with any ground-truth box. Note that a single ground-truth box may assign positive labels to multiple anchors. We assign a negative label to a non-positive anchor if its IoU ratio is lower than 0.3 for all ground-truth boxes. Anchors that are neither positive nor negative do not contribute to the training objective. With these definitions, we minimize an objective function following the multi-task loss in Fast RCNN [2]. Our loss function for an image is defined as:

[](https://www.codecogs.com/eqnedit.php?latex=L%5Cleft(%5Cleft%5C%7Bp_%7Bi%7D%5Cright%5C%7D%2C%5Cleft%5C%7Bt_%7Bi%7D%5Cright%5C%7D%5Cright)%3D%5Cfrac%7B1%7D%7BN_%7Bc%20l%20s%7D%7D%20%5Csum_%7Bi%7D%20L_%7Bc%20l%20s%7D%5Cleft(p_%7Bi%7D%2C%20p_%7Bi%7D%5E%7B*%7D%5Cright)%2B%5Clambda%20%5Cfrac%7B1%7D%7BN_%7Br%20e%20g%7D%7D%20%5Csum_%7Bi%7D%20p_%7Bi%7D%5E%7B*%7D%20L_%7Br%20e%20g%7D%5Cleft(t_%7Bi%7D%2C%20t_%7Bi%7D%5E%7B*%7D%5Cright)%0)

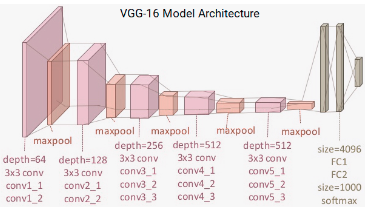
Here, [](https://www.codecogs.com/eqnedit.php?latex=%5Cdot%7B%5Cboldsymbol%7Bi%7D%7D%0) is the index of an anchor in a mini-batch and [](https://www.codecogs.com/eqnedit.php?latex=p_%7Bi%7D%0) is the predicted probability of anchor [](https://www.codecogs.com/eqnedit.php?latex=%5Cdot%7B%5Cboldsymbol%7Bi%7D%7D%0) being an object. The ground-truth label [](https://www.codecogs.com/eqnedit.php?latex=p_%7Bi%7D%5E%7B*%7D%0) is 1 if the anchor is positive, and is 0 if the anchor is negative. [](https://www.codecogs.com/eqnedit.php?latex=t_%7Bi%7D%5E%7B*%7D%0) is a vector representing the 4 parameterized coordinates of the predicted bounding box, and [](https://www.codecogs.com/eqnedit.php?latex=L_%7Bc%20l%20s%7D%0) is that of the ground-truth box associated with a positive anchor. The classification loss [](https://www.codecogs.com/eqnedit.php?latex=L_%7Bc%20l%20s%7D%0) is log loss over two classes (object vs. not object). For the regression loss, we use [](https://www.codecogs.com/eqnedit.php?latex=L_%7Breg%7D%5Cleft(t_%7Bi%7D%2C%20t_%7Bi%7D%5E%7B*%7D%5Cright)%3DR%5Cleft(t_%7Bi%7D-t_%7Bi%7D%5E%7B*%7D%5Cright)%0) where R is the robust loss function (smooth L1) defined in [5]. The term [](https://www.codecogs.com/eqnedit.php?latex=p_%7Bi%7D%5E%7B*%7D%20L_%7Br%20e%20g%7D%0) means the regression loss is activated only for positive anchors [](https://www.codecogs.com/eqnedit.php?latex=%5Cleft(p_%7Bi%7D%5E%7B*%7D%3D1%5Cright)%0) and is disabled otherwise [](https://www.codecogs.com/eqnedit.php?latex=%5Cleft(p_%7Bi%7D%5E%7B*%7D%3D0%5Cright)%0). The outputs of the [](https://www.codecogs.com/eqnedit.php?latex=cls%0) and [](https://www.codecogs.com/eqnedit.php?latex=reg%0) layers consist of [](https://www.codecogs.com/eqnedit.php?latex=%5Cleft%5C%7Bp_%7Bi%7D%5Cright%5C%7D%0) and [](https://www.codecogs.com/eqnedit.php?latex=%5Cleft%5C%7Bt_%7Bi%7D%5Cright%5C%7D%0) respectively. The two terms are normalized with [](https://www.codecogs.com/eqnedit.php?latex=N_%7Bcls%7D%0) and [](https://www.codecogs.com/eqnedit.php?latex=N_%7Breg%7D%0) For regression, we adopt the parameterizations of the 4 coordinates following [6]:

[](https://www.codecogs.com/eqnedit.php?latex=t_%7B%5Cmathrm%7Bx%7D%7D%3D%5Cleft(x-x_%7B%5Cmathrm%7Ba%7D%7D%5Cright)%2F%20w_%7B%5Cmathrm%7Ba%7D%7D%2C%5Cquad%20t_%7B%5Cmathrm%7By%7D%7D%3D%5Cleft(y-y_%7B%5Cmathrm%7Ba%7D%7D%5Cright)%2F%20h_%7B%5Cmathrm%7Ba%7D%7D%2C%20%5Cquad%20t_%7B%5Cmathrm%7Bw%7D%7D%3D%5Clog%20%5Cleft(w%2F%20w_%7B%5Cmathrm%7Ba%7D%7D%5Cright)%2C%20%5Cquad%20t_%7B%5Cmathrm%7Bh%7D%7D%3D%5Clog%20%5Cleft(h%20%2F%20h_%7B%5Cmathrm%7Ba%7D%7D%5Cright)%0)

[](https://www.codecogs.com/eqnedit.php?latex=t_%7B%5Cmathrm%7Bx%7D%7D%5E%7B*%7D%3D%5Cleft(x%5E%7B*%7D-x_%7B%5Cmathrm%7Ba%7D%7D%5Cright)%20%2Fw_%7B%5Cmathrm%7Ba%7D%7D%2C%5Cquadt_%7B%5Cmathrm%7By%7D%7D%5E%7B*%7D%3D%5Cleft(y%5E%7B*%7D-y_%7B%5Cmathrm%7Ba%7D%7D%5Cright)%2Fh_%7B%5Cmathrm%7Ba%7D%7D%2C%5Cquadt_%7B%5Cmathrm%7Bw%7D%7D%5E%7B*%7D%3D%5Clog%5Cleft(w%5E%7B*%7D%2Fw_%7B%5Cmathrm%7Ba%7D%7D%5Cright)%2C%5Cquadt_%7B%5Cmathrm%7Bh%7D%7D%5E%7B*%7D%3D%5Clog%5Cleft(h%5E%7B*%7D%2Fh_%7B%5Cmathrm%7Ba%7D%7D%5Cright)%0)

where [](https://www.codecogs.com/eqnedit.php?latex=x%2C%20y%2C%20w%2C%0) and [](https://www.codecogs.com/eqnedit.php?latex=h%0) denote the two coordinates of the box center, width, and height. Variables [](https://www.codecogs.com/eqnedit.php?latex=x%0), [](https://www.codecogs.com/eqnedit.php?latex=x_%7B%5Cmathrm%7Ba%7D%7D%0), and [](https://www.codecogs.com/eqnedit.php?latex=x%5E%7B*%7D%0) are for the predicted box, anchor box, and ground-truth box respectively (likewise for [](https://www.codecogs.com/eqnedit.php?latex=y%2C%20w%2C%20h%0)). This can be thought of as bounding-box regression from an anchor box to a nearby ground-truth box [4].

*VGG16:* In VGG16, what we should pay attention to is that all the convolutional layers is 3x3 kernel size with 1 padding, and all the pooling layers is 2x2 kernel size with 2 stride. Since we have four conv+relu+pooling processes, the output size from every convolutional layer is the same as its input size, and only the pooling layer turn the height and width of the input image into half of its original size in the output. After four such processes, the height and width become 1/16 of its original size (note: 1/2^4). We have totally five conv+relu process, but it does not change the image size. Thus, the convolutional layer will give a size fixed output, and this will help us connect the feature map with original image.

******

*VGG1616*

# MODEL IMPROVISATION

***A.Faster RCNN Implementation-Human Graffiti***

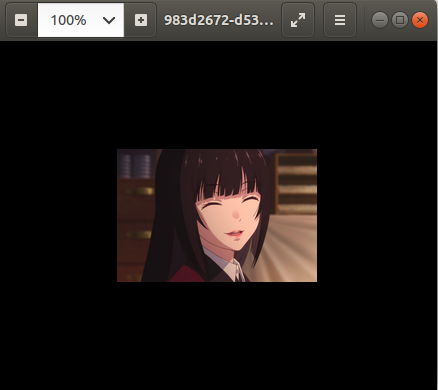
Faster RCNN was implemented on dataset of 1000 images to classify and detect the location of Graffiti Human in an image. However, the model predict a lot of false positives on the Graffiti dataset. Example is shown in the image below.



*False Positive*

***B. Improved negative hard mining of Human Graffiti***

### We removed the regression algorithm for negative dataset and assignment a background class for it.Also, we changed the Batch Normalization mode to Local Response Normalization(LRN) which greatly impacted the negative dataset. Furthermore, we added diversity of types of images in the negative dataset(in this example the anime humans) and evaluated the model. The results showed improvement from previous Faster RCNN model.



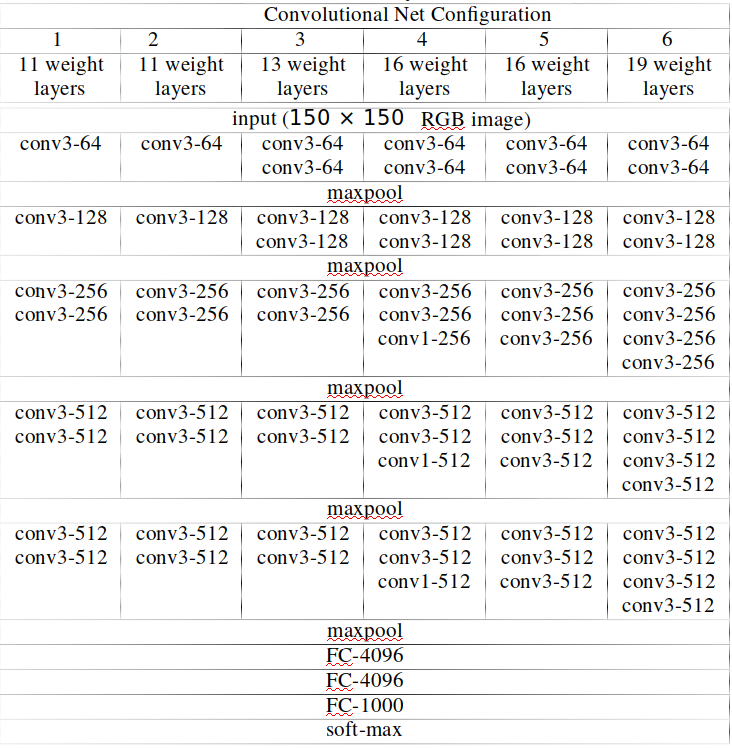
*True Negative*

# Data Driven Deep Learning Model Results

We trained our deep learning models (Faster RCNN,,,) on our dataset and below results were extracted.

***Faster RCNN with VGG16:***

Given label [‘Human Graffiti’].



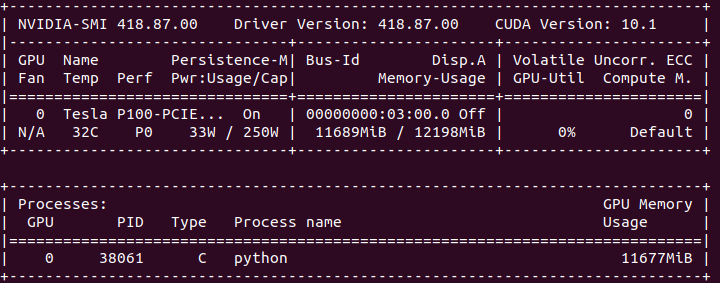
Number of parameters in k:



# PERFORMANCE

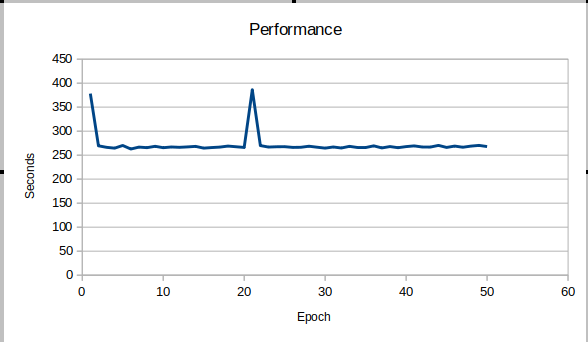
## *Faster RCNN - Human Graffiti*

The training process was done on HPC server which used GPU node and a time limit of four hours per instance. Details of hardware spec is given below.

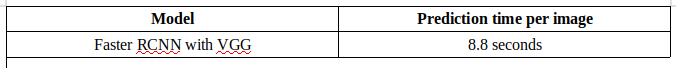
**

*HPC server info*

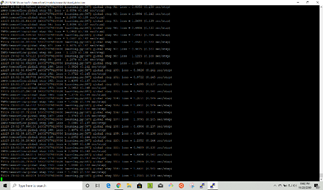
Average number of seconds taken for 1 epoch was 265. If we plot number of epochs and time, we get the following graph which can be used as a performance measure of the model. The spike is due to the four hour limit on HPC cluster.



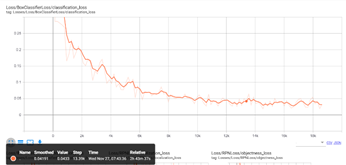
Below is the prediction time for one test image.



*B.*  *Faster RCNN inception V2 – Animal Graffiti*

**

Above figure shows that Faster RCNN with inception v2 is very fast. It only takes only 0.147 sec/steps. Within 3 hours loss of the model becomes 0.05.

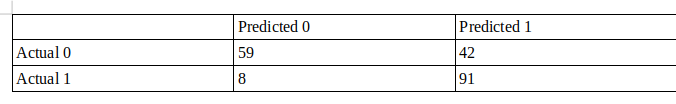


# RESULTS

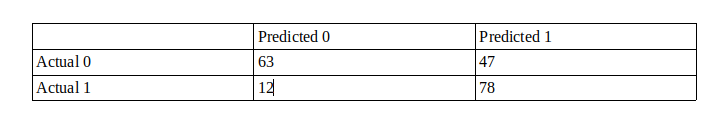
1. *Faster RCNN - Human Graffiti*

We use a very simple measure to compute which base model is better for Faster RCNN-Confusion matrix. Following are the comparison results when we ran Faster RCNN with VGG16 and Resnet50.

VGG16:



Resnet50:



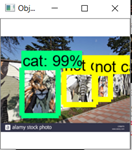
We can easily deduce from the confusion matrix that VGG16 is better with Faster RCNN as compared to Resnet50. Somehow, both models are not good in accuracy when it comes to false positive data.

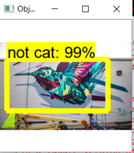
Below is the result of choosing different size of pre-weights trained on Imagenet database. We can see very clearly that full weights has better accuracy as compared to other ones. So we download the full weights for our model.

Here is the graph of training loss and accuracy of the model on the data set during training.

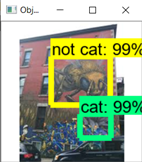
*B.* *Faster RCNN inception v2 – Animal Graffiti*

Around 20,000 steps loss comes to 0.05. At this loss we get accuracy of 99%. From the images below we can see that our model is predicting images with different animals perfectly and successfully able to discriminate between cat species and other animals*.*

* *

* *

As we further train our model even after loss becomes 0.05, model will try to start overfitting our data. We tried to train our model till 35,000 and at this step our loss was 0.03. We tried to predict images using this model and the result of this, our model was taking trivial details in images and tried to predict them as “cat” or “not cat”. Below is the output of overfitted model.

# Conclusion

## *Faster RCNN - Human Graffiti*

Using Faster RCNN with VGG16 as base model we were able to achieve an accuracy of 93 measured

through RPN. More epochs and fine tuning Regions of Interest (ROI) parameter may lead to achieve even higher accuracy.

# 2). Context of the Graffiti

Graffiti images are usually found in the streets. They could be anywhere in the street such as on the walls, on the ground, on vehicles or traffic signs and etc. In this project, we are going to enable our model to detect 2 contexts which are more common for graffiti. We are going to train our model on at least 1200 images of street walls and underpasses. Therefore, our model besides detecting the graffiti type, it will also detect whether the graffiti is drawn on a wall or in an underpass. To begin, Ali will use google image search to collect 600 images of walls and 600 images of underpasses. Later on, we may decide whether we need to collect more images or not depending on our model’s performance.



*Graffiti on a Wall*



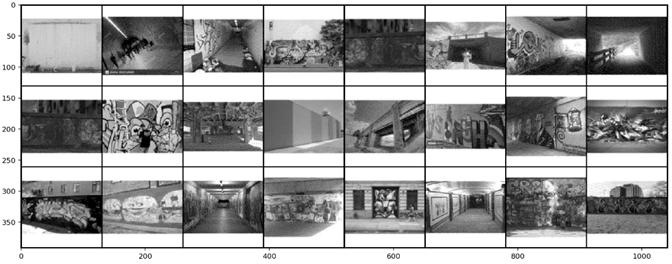
*Graffiti in an Underpass*

## *Training Dataset and Preparation*

Using a tool to download google images, 1000 images which consist of 400 wall images, 400 underpass images and 200 images of other contexts such as trains and signal signs, were downloaded for this part of the project. Initially, more than 2000 images were downloaded, however, certain measures were taken to remove the images that may not be useful for our model. For example, images that described the most difference between a wall and underpass were given a priority to be kept and images with multiple irrelevant objects in the image were ought to be removed.

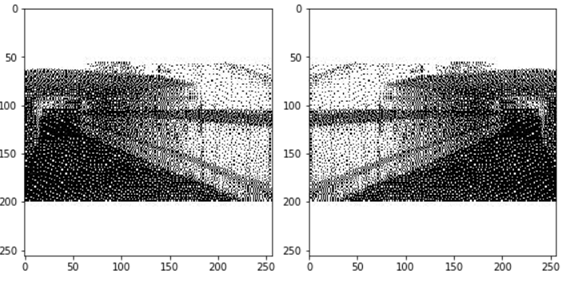
The very first step of pre-processing was to resize the dataset. All images were resized into 200x200 which is large enough to maintain image quality and small enough for machine learning models to process quickly. An alternative is to adjust the size of the dataset depending on the machine learning model within the model itself.

A crucial step in pre-processing was to convert images to black and white. Graffiti images are very intense in colors. The variety of colors can be confusing for any machine learning algorithms. Specially in context detection part of the project, the algorithm may keep on extracting features of the colorful part of the image instead of detecting the features that define the context of a wall or an underpass. Therefore, converting images to black and white can keep the model focused on edge detection and help extracting features that differentiate between walls and underpasses. This idea was tested during the training by training the model on colored images first. The results were very weak and inaccurate using the colored images. Converting images to black and white increased the training accuracy by at least 15%.



*Black and White Dataset*

Before proceeding with creating models and training, first we rescale the image data and then apply augmentation using ImageDataGenerator on the dataset for better learning. Augmentation creates modified copies of the images. In this case images will be rescaled to 1 by 255 and flipped.



*Data Augmentation by Flipping*

After completing the data pre-processing steps, we are confident to create a model for training. As images are resized to a standard shape that is not too large for the models to overrun and data is converted to black and white instead of the 3 channel RGB to enhance model learning and once data augmentation and data generation is completed we separate the training data from test data with a ratio of 0.2. More details will be provided about the training and test data separation in the following sections explaining each model.

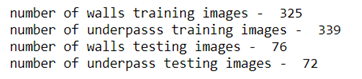
## *Deep Learning Models*

To find the best performance and achieving the highest accuracy multiple models have been tested to detect image context. Starting from simpler CNN models to other advanced CNN models and VGG16 which is a pre-trained advanced CNN model. However, the variety of models alone was not enough to achieve acceptable results. Each model needs to be tuned very specific to the type and shape of the data as well as the features that help classifying the data. Therefore, each model has been tested and tuned with various parameters before reaching a good outcome. In this section each of these models and experiments will be discussed.

In general, CNN models are known to perform excellent on visual data classification. Therefore, a simple CNN model was the first approach in this classification problem. The main purpose of this part is to classify the context of an image which could be a wall or an underpass. More advanced CNN models such as VGG16 were also used for training for comparison.

1. **Simple CNN Model**

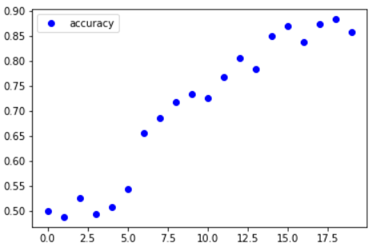
The context of the image is definitely less complicated than the graffiti itself. Therefore, the model is going to need a smaller number of features to classify the context. Thus, making the model too complicated could decrease the accuracy as the model may start fit to the graffiti instead of the context. Therefore, the first approach was to build a simple model and try to tune the parameters to achieve an acceptable accuracy.

 *Data Points*

Initially, a CNN model with three convolutional layers with each having a maxpooling and dropout layer with filter sizes of 64, 128 and 256 in order, and three dense layers with 32, 64 and 128 neurons in order were created for training. As we can see the model may be somewhat complicated with large number of filters and neurons. As expected, the accuracy was suffering to improve and stuck around 52% which is basically a flip of coin.

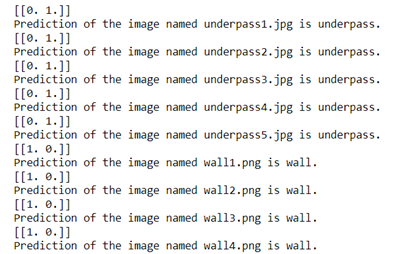
Making the model simpler improved the accuracy and achieved good results. The model was modified to a simpler version with two convolutional layers each having a maxpooling layer and dropout layer with filter sizes 32 and 32 in order, and two dense layers with 32 and 16 neurons in order.

As expected, the training accuracy was improved significantly and model was able to reach an accuracy of 92% on the test set. The last two steps of the training is As We can see the loss was achieved to be 0.3718, training accuracy of 0.8571, validation loss of 0.2319 and the test accuracy of 0.9257.

 *Training Accuracy Steps*

As we can see, the simplified model performed way better than the complicated model. However, the number of layers and neurons were not the only parameters that were tuned. Other tuning measures were taken to improve the results. The number of epochs, number of steps, batch size and kernel sizes are some other parameters that were tuned for this part.

The model was further tested on a small separately created test set containing 5 underpass images and 4 wall images. The model has performed perfectly on the small test set and was able to classify all images correctly.



*Smaller Test Set is 100% Accurate*

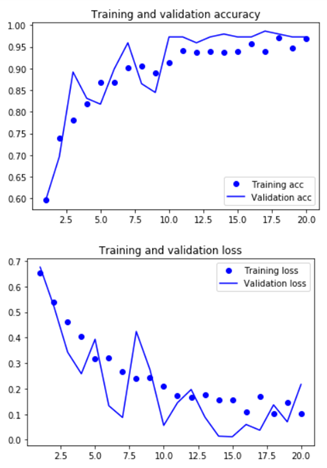
1. **Advanced CNN Model**

During the simple CNN model sparse categorical crossentropy was used as the loss function. In the advanced CNN models binary crossentropy loss function is used. As mentioned in simpler version of CNN, complicating the model decreased the learning accuracy. In the advanced version of CNN model for this part of the project, the CNN model was expanded to more layers as the number of epochs and parameters were increased as well to increase the training accuracy.

The training data consists of 339 wall images, 325 underpass images. The testing dataset consists of 72 wall images and 76 underpass images. Images are in 100 by 100 shape for this model which makes the training shape (664, 100, 100, 3) and testing shape (148, 100, 100, 3).

The advanced CNN was made of four convolutional layers each having a maxpooling and dropout layer with filter sizes 32, 64, 128 and 128 in order, and two dense layers with 512 hidden neurons and 1 output neuron.

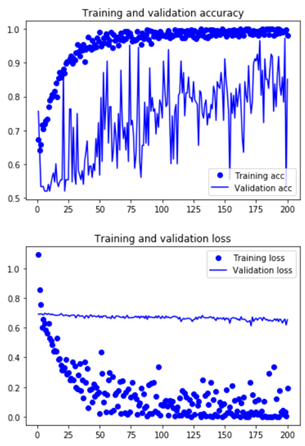
The training accuracy reached 0.9693, testing accuracy to 0.9730, training loss to 0.1032 and testing loss to 0.2167 after 20 epochs. As we can see, the accuracy is as good as the simpler CNN, however, the loss and accuracy does not seem to be very stable along the training epochs. Therefore, we’re achieving a similar accuracy with higher computational cost and less stability.



*Training and Test Loss*

Even though the advanced CNN we just discussed was more complicated than the simple CNN used previously, it is still mildly simplified. Other approaches were taken to try to improve the accuracy of the training with more complicated models. However, not only complicated models did not seem to improve learning but also the accuracy was decreased significantly.

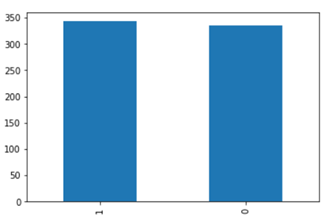
After trying data augmentation, the accuracy was not improved at all even by running it for 200 epochs.We can see that the training accuracy and training loss as well as validation accuracy and validation loss have been behaving very unstably and did not reach to an accuracy as good as a simpler model without augmentation.



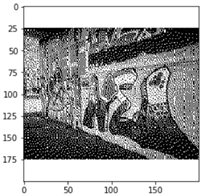
*Training and Validation Loss and Accuracy*

1. **VGG16 Model**

VGG16 is a pretrained CNN model. It is used for image classification for large amount of data. This model was used as a final approach to find the best accuracy and loss values for context classification.



*Data Distribution*



*Data Point Example*

VGG16 model performance was similar to the very first simple CNN model. The accuracy and loss were in an acceptable range. The accuracy and loss are in a good range in the last step of training after 100 epochs. Even though the model consists of 16 levels of convolutional layers, the performance was very good because of the fact that this model has pre-trained weights that are used for prediction in this part of the project which is context classification.



*Accuracy and Loss of VGG16 Model*

Acknowledgment

This work was supported by Dr Jerry Gao computer engineering department, San Jose State university.

##### References

1. [14] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In CVPR, 2015.
2. R. Girshick. Fast R-CNN. arXiv:1504.08083, 2015.
3. [15] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In ICML, 2010.
4. <https://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf>
5. <https://arxiv.org/pdf/1409.1556.pdf>