UO MS Project Report

Adib Mosharrof

March 2021

Contents

Model	Self	Mediscore
LR	0.095	0.235
LGB	0.22	0.23
MLP	0.007	0.004
XGB	0.195	0.238
LR (tuned - log loss)	0.075	0.239
LGB (tuned)	0.19	0.242

Figure 1: Previous Results

1 Introduction

2 Data

3 Related Work

The initial work on this data focused on using traditional machine learning algorithms to perform image localization. In these models one row of data was considered to be a single pixel in an image. A data preprocessing step was applied to convert each image into a 1 dimensional array, thus enabling the use of a single pixel as a row of data. The number of features in a model is equal to the number of indicators that have been selected to be used. As an example of a single row of data, we consider the top left pixel of an image, where we take the value of that pixel from each of the indicators and pass that into the model as input.

One of the major challenges in the previous work was to get a consistent scoring for the algorithms. As mentioned in the previous section, Mediscore was used to evaluate the performance of the models. Since it is a very time consuming step, a local scoring system was created which would provide a score for the models. However, there was no direct correlation between mediscore and the local scoring method, which proved to be a major drawback when fine tuning models to increase performance. Tweaks in hyperparameters that would increase the score in the local scoring, would not produce the same behavior in from the results returned from mediscore. A major step in our current work was to create a new local scoring system that would be fast and also be consistent to what mediscore would produce, thus enabling us to perform experiments without having a tight coupling with mediscore.

Previous work mainly focused on models with decision trees, regression and boosting. The scores produced from the different models are shown in

Initial work started with a vanilla Logistic Regression model that produced a decent score in mediscore. Other

1

4 Scoring

The scoring mechanism in the Mediscore project has a concept of no score region that was not considered in our previous work. The idea behind the no score region is to ignore the regions on the boundary of manipulated and non-manipulated regions. If an algorithm can identify the manipulated region in the image, it could still mislabel a lot of pixels in the boundary region. This could incurr a high penalty to the algorithm and give it a score that does not accurately reflect on its performance. In order to produce the no score region, two of the core morphologic operations; erosion and dilasion, are applied to the image.

4.1 Dilation

Applying dilation $\oplus$ to an image expands the shape of the image.

4.2 Erosion

The general output of erosion is that it shrinks the shape of the image.
[1]

5 Methodology

5.1 Data Pre processing

The format of data expected as the input layer varies depending on the type of model being used. The raw image files cannot be served directly as input, thus we need to process these images into the format expected by the appropriate model. The highest level of distinction is whether the model expects 1 dimensional data, or it expects a 2 dimensional data, an image of a certain size. The data preprocessing step exists as an individual module, in which the input is the raw images and the output is either an image or a csv file depending on the configuration specified.

5.1.1 Pixel

Models like Logistic Regresssion, Random Forests and other traditional machine learning algorithms, use a single pixel as a row of data. Images are read in transformed from a 2 dimensional array into a 1 dimensional array, so that it can be fed into the model as input. The output of the model will be a 1 dimensional array of the same size as the input array. A very important step has to be taken next, which is converting this 1 dimensional array back to a 2 dimensional array, so that we can generate an image of the output of the model. The dimension reduction can be done by stacking rows or columns together, so when we are generating the output image from the model prediction, we have to be consistent with the method used previously.

5.1.2 Image Resizing

Neural Network models expect input images to be of a certain dimension, and so in this data pre-processing step, we resize all the input images to a certain size. Since there is a big variation in the image sizes, depending on the resize resolution, some images might be scaled up whereas some might be scaled down.

5.1.3 Image Patches

When resizing images to a fixed size, a lot of information in images are lost. When a high resolution image is shrunken down considerably, there is a lot of blur in the image and fine grained details are lost in the process. To mitigate this problem, we break down a whole image into patches of a fixed size. One main challenge here is that not all images break out into even patches. There are numerous pixels around the boundary of an image that are not enough to fulfil the size of the image patch. To remedy this, we add padding to the image both horizontally and vertically, so that the image is of a size that can be broken down evenly according to the patch size specified.

A major step in this preprocessing is reconstructing the source image from the individual patches. We have to align the patches in the right order and also remove the padding we added. In order to create patches and reconstruct patches from an image, we used a library, patchify, which provides functions to produce patches from an image given a patch shape, and also function to reconstruct images, given a list of patches.

5.2 Architectures

We have used mixture of Machine Learning models and architectures, ranging from traditonal models like Logistic Regression, to simple Neural Network and even a bit more complicated neural architecture, U-NET. Models require data in certain formats, so each model is tied to a certain data preprocessing step, so that it can obtain the data in the format it wants.

5.2.1 Logistic Regression

Logistic Regression is a classical linear Machine Learning model that has proved to be a really simple but efficient model. We have included this to create a baseline score with which we can compare other models. This model has been implemented using the scikit-learn library. One of the major drawbacks of logistic regression is that it has to load the whole data for training, thus we are limited by the amount of memory we have. It does not have the ability to perform batch learning, thus creating this challenge. Due to this challenge, we had to perform Logistic Regression on data that had been scaled down by a factor of 32 or more so that we can avoid the memory challenges.

5.2.2 Neural Networks

We have explored with different types of Neural Network architectures to fit the data, ranging from simple single layer networks to multi layers and also the complex U-NET architecture. The neural architectures have the ability to work with both 1 and 2 dimensional data. Data preprocessors arecoupled to the type of architecture to feed the data in the required format. The main advantage of neural networks is that it can perform training in batches, thus allowing to train on a large dataset and also on images of larger resolution. For images with large resolutions, we had to minimize the batch size in order to fit the memory constraint.

The networks have been implemented using the Keras 2.0 library. Sigmoid and ReLU activation functions have been used when building the networks. To fine tune the models, different L1 regularization and learning rate values has been used.

5.2.2.1 Multi-layer Neural Networks

A sequential linear multi-layer network has been applied to the 1 dimensional data. The number of neurons in the input layer is equal to the number of indicators in the data. Each dense layer is followed by an activation layer. The final output layer has only one neuron, which predicts whether the current pixel is manipulated or not. The input layer has the maximum number of neurons, and the number of neurons in successive dense layers are halved every time. For example, if we start with an input layer with a dimension of $64x64$, and follow it with 3 dense layers, then the dimensions of these 3 layers would be $32x32$, $16x16$ and $8x8$. These dense layer would then be augmented with the output layer, which would have only 1 node.

Add an image of the architecture here

5.2.2.2 UNET

UNET is a special type of Convolutional Neural Network, that has been designed specifically to solve image segmentation and localization.

MORE DETAILS ABOUT UNET HERE

U-NET works with 2 dimensional data, specifially on the data produced after the preprocessing steps of image patches and image resizing. A single layer in the network is designed to have a convolutional layer, followed by a max pooling layer, which is subsequently followed by a dropout layer. The various hyperparameters have been tinkered with to find the set of values that produce the best results. More regarding hyperparamter tuning are included in the experiments section.

5.2.3 Weighted Ensemble

The different machine learning algorithms pick up different patterns from the input data, some of which might be very important, while others might be less. There is also the possibility that some patterns are picked up by certain algorithms, whereas other algorithms completely fail to pick them up. In order to tackle these cases, we have introduced a weighted ensemble of the algorithms. Each algorithm runs independently and produces a prediction. Next, a weighted average is performed on the predictions based on the weights provided to produce a final output.

6 Experiments

7 Results

8 Discussion

9 Future Work

References

[1] Tambe, S. B., Kulhare, D., Nirmal, M., and Prajapati, G. Image processing (ip) through erosion and dilation methods.