UO MS Project Report

Adib Mosharrof

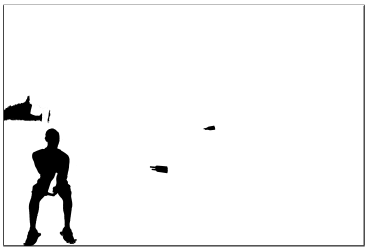
March 2021

Contents

Abstract

Digital image manipulation has become a major security concerns as image editing has become very simple due to user friend image editing software, image generation softwares and other simple image manipulation techniques like image splicing, translations, distortions, copy-move, removals and many more. A lot of research in detecting and localizing image manipulation has produced an abundance of algorithms. Most algorithms specialize in detecting particular manipulations, thus a comprehensive detection is lacking. In this project we embark on the process of ensembling the results of different image manipulation algorithms to produce a single localized prediction of image manipulation. Different types of image preprocessing have been applied to traditional machine learning and deep learning architectures to produce the ensembled prediction. We performed experiments on the Media Forensics Challenge 2018 Dataset and found that traditional machine learning techniques performed better.

(a) Probe Image



(b) Target Image

Figure 1: Probe and Target image of a row of data

1 Introduction

2 Data

We used the Media Forensics Challenge 2018 Dataset [4] to perform our experiments. Each entry in the dataset is uniquely identified using a 32 character unique string. Each entry has 3 parts in it, a probe image, target image and a indicator images. The probe image, shown in Figure 1a, is the original image with the manipulation in it. The target image, shown in Figure 1b, is a black and white image, in which the black pixels represent manipulated areas and white pixels represent non-manipulated. As mentioned before, the algorithms that perform image manipulation localization are called indicators and the output image from these algorithms are called indicator images. The indicator images, a few shown in Figure 2, are the output images produce from the indicators that ran on this probe image. To reduce the computational costs of our experiments, we downscaled the original dataset by a factor of 40.

2.1 Exploratory Data Analysis

To better understand the data and its properties, some simple data exploration steps have been performed. Since the core task is localization of manipulations,

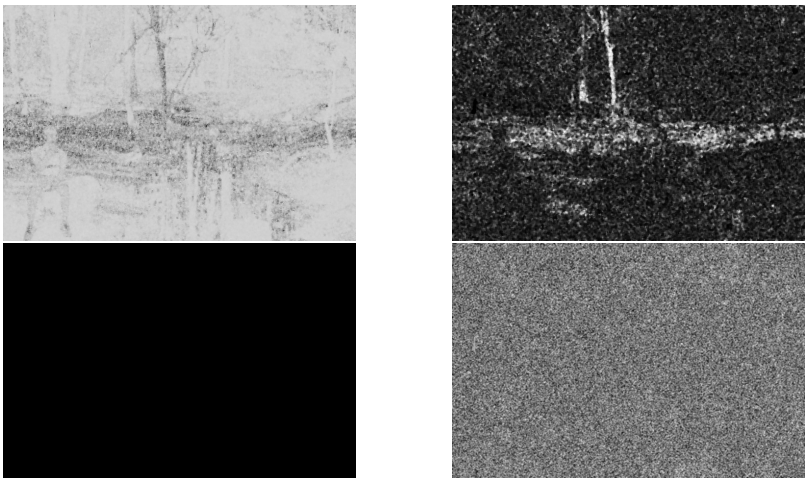


Figure 2: A few indicator images

it is very important to get an idea of the distribution of the percentage of manipulated pixels in the probe images. To visualize this, for each image we calculated the fraction of image pixels that were manipulated with respect to the whole image and plotted a pie chart of the distrubtion in Figure 3, which shows that most of the images have very little manipulation in them. The skewness in the amount of manipulation is a major factor that we have to take in consideration when designing the models and interpreting the results.

The distribution of the size of images is another important factor to consider and to visualize this a graph of the frequency of the image dimensions have been plotted in Figure 4, which shows that there is skewness here as well.

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

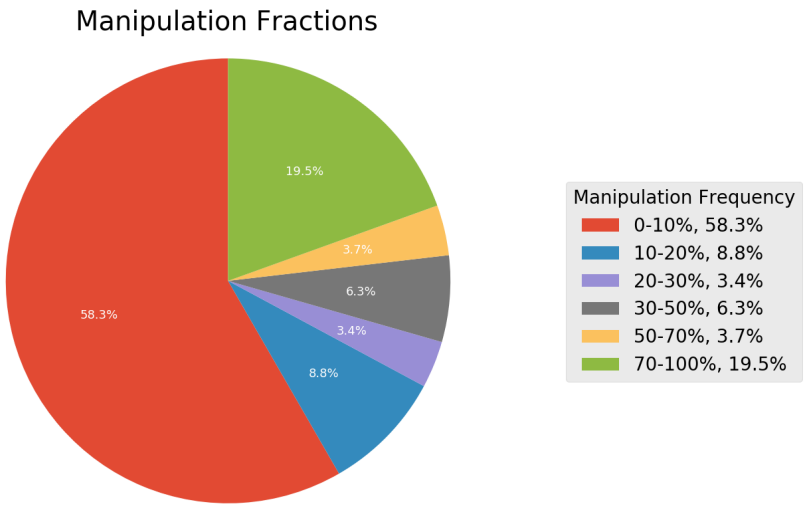


Figure 3: Manipulation Fractions

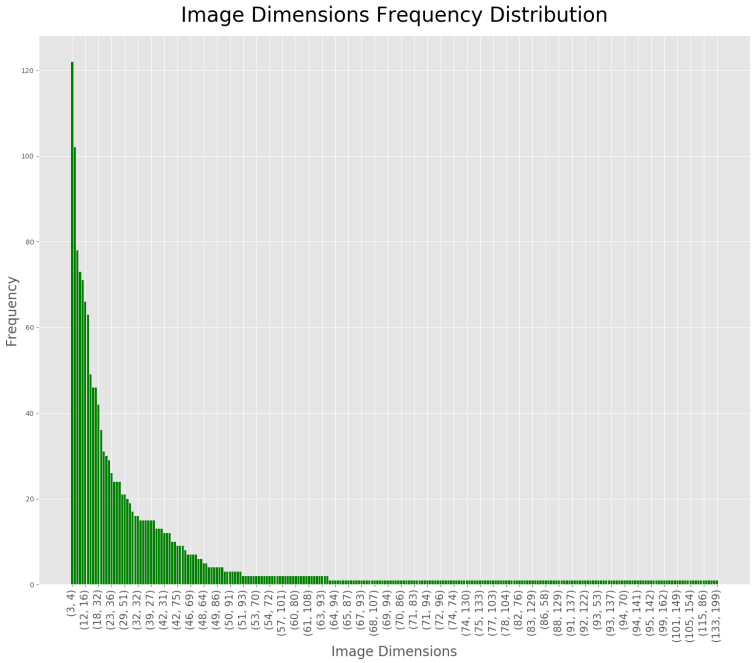


Figure 4: Image Dimension Frequency Distribution

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 5: Previous Results

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 figure 5

Get more ideas from Daniel about previous work

4 Scoring

As mentioned in the Data section, in the dataset, the proportion of manipulated pixels when compared to non-manipulated pixels is very small. Unblanced data generally causes algorithms to be biased towards the majority class, thus popular metrics like accuracy, which is the ratio between the number of correctly classified samples and the overall number of samples (for example [11]), will no longer be an accurate metric to use here. Accuracy would provide an overoptimistic estimation of the classifier's ability on the majority class [2]. Consider a data set that has 10% positive class and 90% negative class. A naive classifier that always outputs the majority class label will have a high accuracy of 0.90.

The problem caused due to this imbalance can be addressed by using the Matthews correlation coefficient (MCC), a special case of the ϕ phi coefficient [5]. MCC has been originally developed by B.W. Matthews for comparison of chemical structures [7]. However, it was re-proposed by Baldi and colleagues [1] in 2000 as a standard performance metric for machine learning with a natural extension to the multiclass scenario [3].

4.1 Matthews Correlation Coefficient

Matthews correlation coefficient is defined in terms of True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). These terms can be calculated from the confusion matrix as defined in Figure 6. MCC is a contingency matrix method of calculating the *Pearson product-moment correlation coefficient* [9] and can be calculated using Equation 1.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{1}$$

		Prediction	
Actual		P	N
	P	TP	FN
	N	FP	TN

Figure 6: The Standard Confusion Matrix

Positives (P) and Negatives (N). True Positives(TP) and True Negatives (TN) are the correct predictions, while False Negatives (FN) and False Positive (FP) are the incorrect predictions

4.2 Mediscore

The scoring mechanism in the Mediscore project has a concept of no score region that was not considered in the previous work. The idea behind the no score region is to ignore the regions on the boundary of the 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, which could incurr a high penalty to the algorithm and give it a score that does not accurately reflect its performance. To get a more fair score, the score is calculated by considering the pixels in the image excluding the pixels that are present in the no score region. In order to produce the no score region, two of the core morphologic operations; erosion and dilasion, are applied to the image. A difference operation is performed on the images produced from erosion and dilation, which is followed by a binary inverse thresholding to produce the set of pixels that constitute the no score region.

4.3 Dilation

Applying dilation $\oplus$ to an image expands the shape of the image. Let X be a gray scale shape and B be a symmetric structuring element. The dilation operation on these two elements, $X \oplus B$ can be defined as

$$X \oplus B = X + b = \{x + b : x \in X \& b \in B\} \tag{2}$$

The output of dilation is the set of translated points such that translate of the reflected structuring element has a non-empty intersection with X. Equation 2 is based on obtaining the refelction of B about its origin and shifting this reflection by b. The dilation of X by B is the set of all the displacements, b, such that x and b overlap by atleast one element [10].

4.4 Erosion

Erosion Θ is generally applied to images for eliminating irrelevent details in terms of size.

$$X \Theta B = X - b = \{z : (B + z) \in X\} \tag{3}$$

The output of erosion is the set of translation points such that the translated structuring element is contained in the input set X. Equation 3 indicates that the erosion of X by B is the set of all points b such that B, translated by b, contain X [10]. The general output of erosion is that it shrinks the shape of the image.

4.5 No Score Region

The boundary no score region is produce by performing two operations sequentially, which are defined in Equation 4 and 5 Equation 5 is basically a binaray inverse thresholding.

$$\psi = (X \Theta B) - (X \oplus B) \tag{4}$$

$$\text{NoScore}(x, y) = \begin{cases} 0, & \text{if } \psi(x, y) > 0 \\ 1, & \text{otherwise} \end{cases} \tag{5}$$

4.6 Thresholded MCC

The output of the algorithms is a grayscale image where pixel values are in the range $[0 - 255]$ and the ground truth is an image which has pixel values of 0 or 1, which represent manipulated and non-manipulated pixels respectively. The output image is converted from a 2 dimensional array into a 1 dimensional array and the pixel indices that are in the no score region are filtered out to produce a list of pixels that are to be used in scoring. The indices in this list are called the scoring indices which will be used to calculate the MCC score.

The pixel values in the scoring indices list need to be converted to have a value of either 0 or 1. This conversion is done using a binary thresholding on the scoring indices list, where the threshold values are in the range $[0 - 255]$. Thresholding is applied using every value in the range and the output produced is used with the ground truth to produce an MCC score for that particular threholded value. The maximum MCC value obtained is selected as the MCC score for the image.

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

The central mathematical concept that underlies logistic regression is the logit —the natural logarithm of an odds ratio. The simplest example of a logit derives from a 2×2 contingency table [8]. For simplicity, lets assume that we

have designated one of the outcome levels the event of interest, which we can call the event. The odds of the event is the ratio of the probability of the event happening divided by the probability of the event not happening. In a fair die, the odds of rolling a number < 5 would be 2 as rolling a number < 5 is twice as likely as rolling a 5 or 6. Symmetry in the odds can be calculated by taking the reciprocal, and so the odds of rolling at least a 5 would be 0.5 [6].

A simple logistic regression has the form defined in Equation 6 [6]

$$ln[odds(Y = 1)] = \beta_0 + \beta_1 X \tag{6}$$

where:

$$\begin{aligned} \text{Y} &: \text{outcome} \\ \text{Y} = 1 &: \text{when the event happens} \\ \text{Y=0} &: \text{when the event does not happen} \\ \beta_0 &: \text{intercept term} \\ \beta_1 &: \text{regression coefficient} \end{aligned}$$

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 in the environment we are running the experiments. 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

Add literature about 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

Add literature about multi layer nn

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 64×64, and follow it with 3 dense layers, then the dimensions of these 3 layers would be 32×32, 16×16 and 8×8. 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

Add literature on UNET

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

Add literature about 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

6.1 Logistic Regression

List scores of LR that shows baseline to compare against

6.2 Neural Networks

List results of the following experiments

- Number of layers
- Regularization
- Learning Rate
- Activation Function
- Total Data size
- Batch Size
- Epochs

6.3 UNET

- Number of layers
- Regularization
- Patch shape
- Total Data size
- Batch Size
- Epochs

6.4 Ensemble

Vary weights and see performance of two algorithms in ensemble

- LR with NN,
- LR with UNET

- NN with UNET

Pick best results from each architecture and create comparison graph for MCC score

7 Discussion and Future Work

Talk about why UNET is failing

Future work: Find more appropriate architecture that has complexity in between a LR model and UNET model

A Project Code Structure

The program has been written in python 3.6 and the list of dependencies can be found in the requirement.txt file. It is important to note that having a different version of python, keras, or tensorflow could result in runtime errors. Installing the tensorflow-gpu package greatly reduces the machine learning training time when compared to tensorflow-cpu.

The program has multiple independent modules that perform tasks:training, testing and scoring; that can be used to perform experiments for image localization. It also has the ability to automatically detect the environment:local, talapas and openstack; in which the code is running. Each module has a default set of parameters in a configuration file, which can be overridden by command line arguments when running a module from the command line.

The file structure of the code can be seen in Figure 7. The main folders to focus on at the root level of the project are src and outputs. The source code is located inside the src folder and the output folder contains the results produced after running a module.

The src folder has 6 subdirectories : architectures, *csv_to_image*, *patches*, *predictions*, *scoring* and *shared*. The independently runnable modules are

csv_to_image, *patches*, *predictions*, and *scoring*, whereas the other two folders, shared and architectures, are consumed by the other modules.

Each runnable module follows a convention for naming files and folders. The config folder contains json files- one per environment- that have default values numerous variables that are used in the code. A $config\_loader.py$ folder is also present in this folder, which parses command line arguments that can be passed as inputs to the module. These input values override the default values that are present in the json file.

To run a module, the *<module_name>_runner.py* file needs to be executed. The *<module_name>_runner.srun* files store the configurations for submitting a job in talapas. Command line inputs need to be provided in the .srun file when the module is executed in talapas.

The shared and architectures folders do not contain any files that can be executed independently. The architectures folder contains files that must implement a method called get_model, which should return a machine learning model object. Currently, only model objects from the Keras and Sklearn libraries are returned. The shared folder contains files with static python classes that provide utility functions and the name of the file indicates the type of utility functions it has.

B Prediction Module

The prediction module in the Medifor Code base is analogous to a rib-eye steak, the prime module of the system. This module loads image data, loads and trains a machine learning model, and makes predictions on test data. This document will outline how the prediction module works by initially starting at a high level and slowly delving into the intricate particulars.

The point of entry of the prediction module is the *prediction_runner.py* file. The module contains an abstract Prediction class, and three concrete classes —

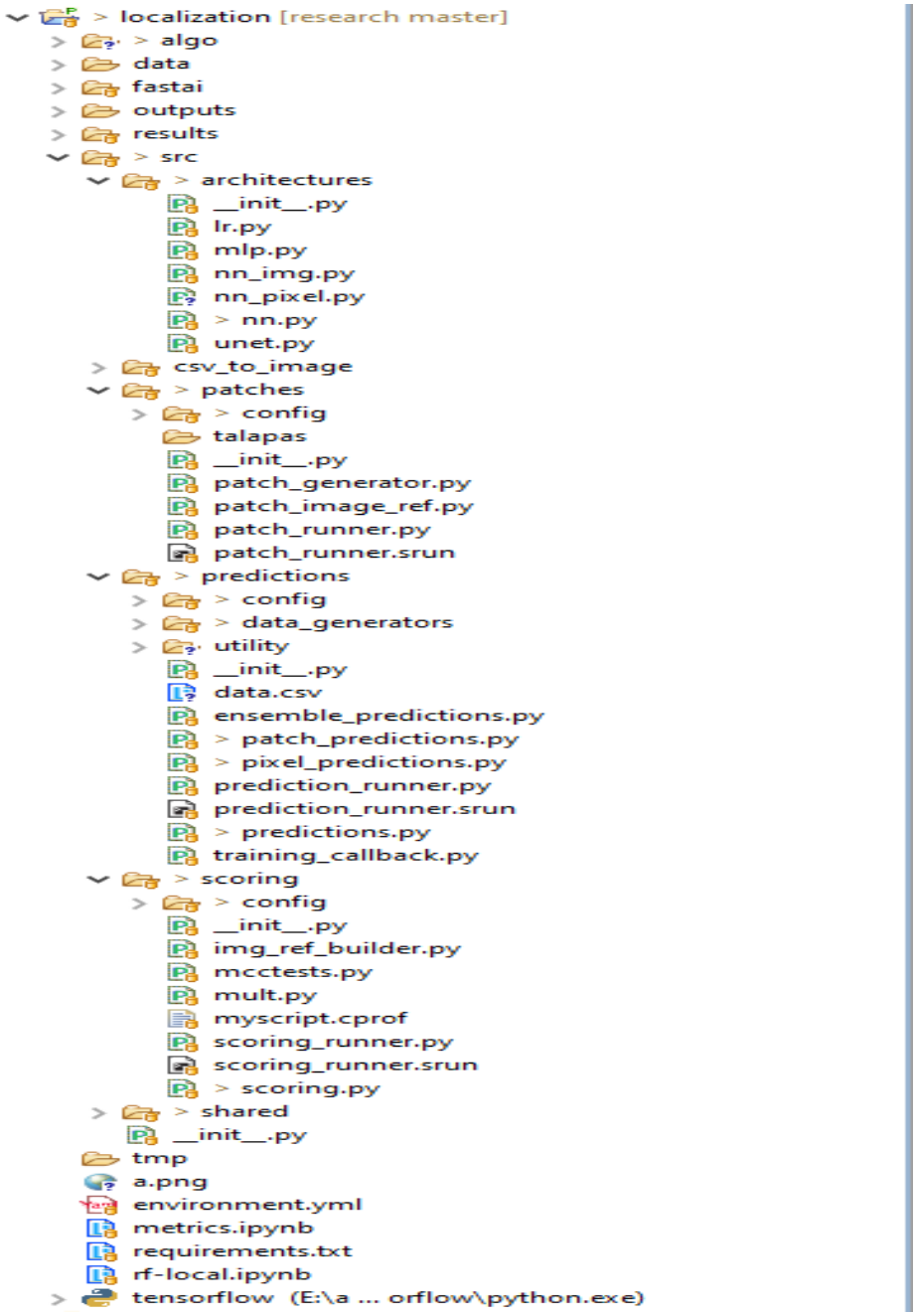


Figure 7: Code File Structure

PixelPredictions, PatchPredictions and EnsemblePredictions. Similar to all the other module runners, the first step is to read in the configurations. The next step is to load a concrete Prediction class based on the configuration rules and call the *train_predict* function.

The *train_predict* function in the Predictions class directs the work being done in this module. It initializes the data generators, trains the model, produces the model output in the predict function, and finally calls the scoring module to get the scores. The child prediction classes implement methods for preparing the image references, initializing data generators, and reconstructing the image from the model output. The *train_model* function in the Predictions class has a call to the *get_architecture* function, which must also be implemented by the child prediction classes.

Two data generators, TrainDataGenerator and TestDataGenerator; are required by a Prediction class. As the name suggests, TrainDataGenerator is used to load data for training the model and TestDataGenerator is used to load the data for testing the model. The data generators are located inside the *data_generators* folder. The project uses different data types: csv, pixel, patch and image. There are test and train data generators for each of the data types. These data generators are fairly complex as they have a lot of parameters in the constructors. The key method in a data generator is the *__getitem__* method, which returns two arrays. The first array is used as the input to the machine learning model and the second array is the ground truth. It is of paramount importance that the dimensions of the two arrays be consistent with what the machine learning model is expecting.

This module follows naming conventions when it comes to naming files and methods, which can be identified upon inspecting the files and methods. When creating new prediction or data generator classes, it is suggested to follow the pattern and place those files in the appropriate directories. If a new data type is introduced, a new Predictions class should be created and it should extend

the abstract Prediction class and implement the abstract methods.

C Architecture Module

The architectures module contains the implementations of the different machine learning models used. Each class in this module defines a machine learning model and must implement a *get_model* method, which returns a model object. This module cannot be run independently; it exists to serve the predictions module. Based on the model name provided configuration of the predictions module, a model class will be instantiated and the *get_model* method called during the training phase.

A model class serves only one data format, therefore when creating or investigating an existing model, the data format must also be explored. Neural network models have been implemented for pixel and image data formats. Logistic regression models have been implemented for the pixel data formats and the UNet model has been implemented for the patched image data format.

The Keras library has been used to create UNet and Neural Network models, whereas SkLearn library has been used to create Logistic regression models. The model objects returned from these two libraries are not consistent, thus the prediction module has to account for the differences. Keras provides the ability to perform training on small batches, whereas Sklearn expects the whole data to be loaded at once.

The UNet and Neural Network models have been parameterized to be customizable. The full list of parameters and details about specific parts of the model can be found in the respective classes. The logistic regression model just returns a SkLearn model object without any such customizations.

References

[1] Baldi, P., Brunak, S., Chauvin, Y., Andersen, C. A., and Nielsen, H. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics 16*, 5 (2000), 412–424.

[2] Chicco, D., and Jurman, G. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics 21*, 1 (2020), 1–13.

[3] Gorodkin, J. Comparing two k-category assignments by a k-category correlation coefficient. *Computational biology and chemistry 28*, 5-6 (2004), 367–374.

[4] Guan, H., Kozak, M., Robertson, E., Lee, Y., Yates, A. N., Delgado, A., Zhou, D., Kheyrkhah, T., Smith, J., and Fiscus, J. Mfc datasets: Large-scale benchmark datasets for media forensic challenge evaluation. In *2019 IEEE Winter Applications of Computer Vision Workshops (WACVW)* (2019), IEEE, pp. 63–72.

[5] Guilford, J. P. Psychometric methods.

[6] LaValley, M. P. Logistic regression. *Circulation 117*, 18 (2008), 2395–2399.

[7] Matthews, B. W. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure 405*, 2 (1975), 442–451.

[8] Peng, C.-Y. J., Lee, K. L., and Ingersoll, G. M. An introduction to logistic regression analysis and reporting. *The journal of educational research 96*, 1 (2002), 3–14.

[9] Powers, D. M. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061* (2020).

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

[11] Wang, L., Chu, F., and Xie, W. Accurate cancer classification using expressions of very few genes. *IEEE/ACM Transactions on computational biology and bioinformatics 4*, 1 (2007), 40–53.