Machine Learning II
Final Project
by: Mina Hanna and Mojahid Osman
Date: Dec-06-2021

1. Overview

"An image is worth a thousand words" is the famous statement by Henrik Ibsen which implies the value and the impact of an image and how a complex message can be delivered with one visual rather than a lengthy verbal description. On the other hand, a fake image can have a severe negative impact on the overall public respect and confidence in news and other social communications.

The other threatening fact about fake media (images, videos, and news) is the availability and sophistication of tools and platforms that helps to easily fabricate such fake media and distribute it. This project aims to utilize deep learning models to classify images and be able to detect forged images which was manipulated using different techniques.

This project focuses on Image tampering detection using Convolutional Neural Network (CNN) classification based on popular filters and analysis used in today's forensic image analysis like Error Level Analysis, Noise Analysis and Luminance Gradient Analysis.

The report is divided into the following main sections:

- Introduction to digital image forensic
- The dataset and the exploratory data analysis conducted in the project
- Deep neural networks used
- Results and Conclusions

2. Introduction to digital image forensic

Image tampering can be defined as "adding or removing important features from an image without leaving any obvious traces of tampering" [1]. Image tampering or forgery techniques can have different forms which include the following [2]:

Image retouching: reduce or improve certain features of the image

Source: digitallybeautiful.blogspot.com

Copy-move: copy part of the image to hide certain areas of shows areas several times

source: conspiracy-cafe.com

Image splicing: compose a new image using fragments from same or different images

Source: https://www.mdpi.com/1099-4300/21/4/371

Processing through filters and color changes

Source: https://www.lightstalking.com/boost-golden-hour-images-lightroom/

Detecting image forgery follows Locard's exchange principle "whenever two objects come in contact, there will always be an exchange" [3]. Each image tampering technique requires certain tools and analysis to reveal such exchange and detect the tampered image. There are different techniques known in the digital image forensic, but the report here will focus on the following listed techniques which will be used throughout this report:

Error Level Analysis (ELA)	Based on the lossy nature of the JPG algorithm, where each save introduce new compression error, ELA algorithm is designed to detect different error rates in the image
Noise Analysis	Each image has its own natural noise based on the lens, lighting condition or other digitalization processes and this technique depends on detecting a change of the natural noise due to tampering the image.
Gradient Analysis	Detecting changes in the brightness level across the image and detect any anomalies

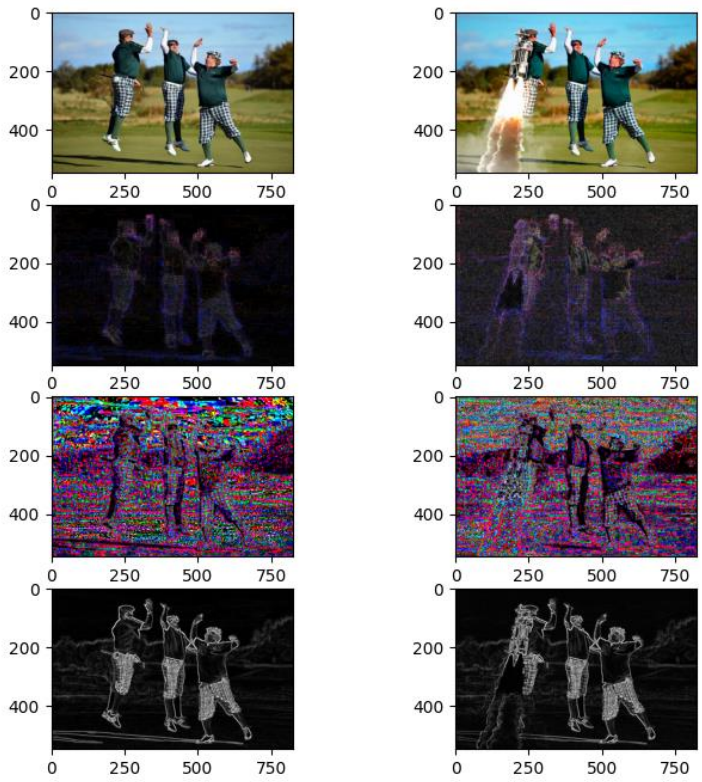


Figure 1: shows the original and photoshopped images (top row) and each is followed by the ELA, Noise ad Gradient filters.

3.The dataset, Exploratory Data Analysis and Pre-Processing

The dataset used in this project is the PS-Battles dataset which is gathered from a large community of image manipulation enthusiasts and provides a basis for media derivation and manipulation detection in the visual domain. The dataset consists of 102'028 images grouped into 11'142 subsets, each containing the original image as well as a varying number of manipulated derivatives. [4] which means that original photos are 11,142 and the photoshopped photos are 90,886.

Each original image has a corresponding sub directory (having the same image name) under the photoshopped directory that contains the manipulated version of this image.

Figure 2: The original image (top row) and followed by the photoshopped versions

The following sequence took place to prepare the date:

1. Cleaning up: deleting any unwanted extensions and images that failed to be loaded by openCV
2. Exploration functions: count and analyze the data and run different functions to pull original picture and its corresponding photoshopped images for comparisons
3. Organization: remove sub-directories in photoshopped images and end up having two directories “original” and “photoshopped” each contains corresponding images
4. Use only subset of the ~10,000 from each class due to size and performance limitations
5. Apply filters: prepare filter extracted images (ELA, noise, and gradient) under the same folder structure

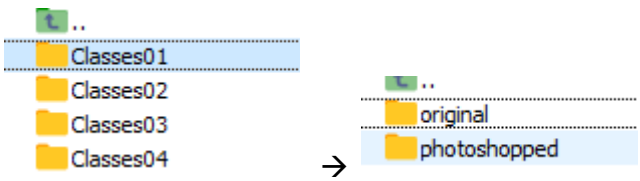


Figure 3: folder strcuture

4. Convolutional Neural Network

Using an ANN for the purpose of image classification would end up being very costly in terms of computation since the trainable parameters become extremely large. For example, if we have a 50 X 50 image of a cat, and we want to train our traditional ANN on that image to classify it into a dog or a cat the trainable parameters become – (50*50) * 100 image pixels multiplied by hidden layer + 100 bias + 2 * 100 output neurons + 2 bias = 2,50,302.

Convolutional Neural Networks come under the subdomain of Machine Learning which is Deep Learning. It was proposed by computer scientist ***Yann LeCun*** in the late 90s, when he was inspired from the human visual perception of recognizing things.[5]

Constructing CNN convolution layer follows a hierarchical model which works on building a network, like a funnel, and finally gives out a fully connected layer where all the neurons are connected to each other, and the output is processed.

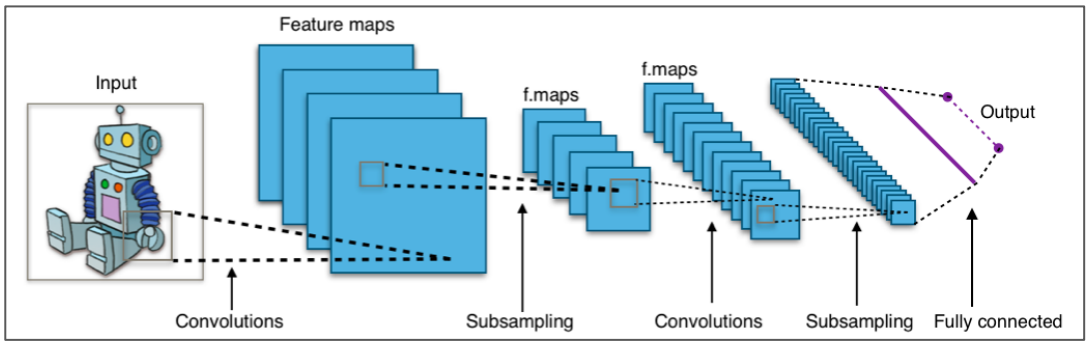


Figure 4: Convolution Concept
(image source: https://medium.com/@himadrisankarchatterjee/a-basic-introduction-to-convolutional-neural-network-8e39019b27c4)

A convolution network is a multilayer feedforward network that has two- or three-dimensional inputs. It has weight functions that are not generally viewed as matrix multiplication (or inner product) operations, The principal layer type for convolution networks is the convolution layer that convolute a kernel over the image and create feature maps, Pooling layer that is used to subsample the output of the convolution layer which reduce the size while maintaining the features for next layers.

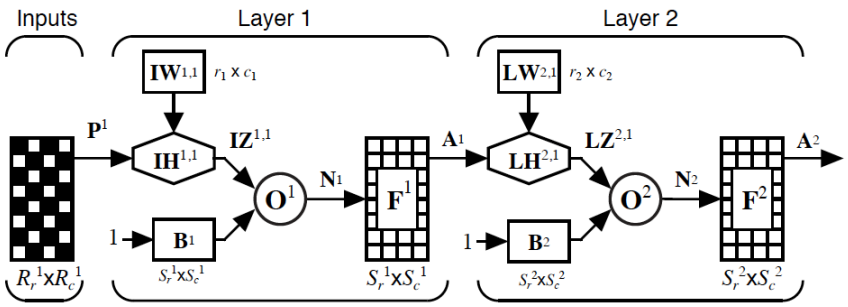


Figure 5: Convolution Neural Network Architecture

For an image v with dimension Rr x Rc and kernel W with size of r x c, the weight function is calculated as follows:

$$z_{i,j} = \sum_{k=1}^{r} \sum_{l=1}^{c} w_{k,l} v_{i+k-1,j+l-1}$$

TensorFlow:

When it comes to picking a framework for your deep learning project, people are always debating between TensorFlow and PyTorch. Both frameworks are very useful abstractions and reduce a decent amount of code and speed up model development. In this project, Tensorflow was the framework of choice due to the experimental nature of the project and the need to build fast model to assess the further implementation approaches.

Building the model:

As described in the previous section, four image datasets was created with the following properties:

- Normal image as downloaded from the original dataset
- Error Level Analysis version of the image
- Noise extracted version of the image
- Gradient magnitude version of the image

The decision to pre-build these datasets and not to convert the images on the fly was due to the time complexity of the conversion process which will not be able to leverage the GPU capabilities and will slow down the training process. After the Data Preprocessing phase, the following was performed to complete the model architecture:

Step 1: Prepare Dataset for Training

Leverage TensorFlow **image_dataset_from_directory()** function to automatically load images and their corresponding target class from the directory structure. The function is used also to Create Training & Validation Data, where data was divided into two groups with a ratio of 80/20.

```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  label_mode="binary",
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

Step 2: Images Augmentation and rescaling:

Sperate layers for image augmentation and rescaling are defined and was added later to CNN Architectures utilizing the GPU and to minimize the overfitting

```
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",
                      input_shape=(img_height,
                                   img_width,
                                   3)),
    #layers.RandomRotation(0.01),
    layers.RandomZoom(0.1),
  ])
```

Step 3: Convolutional, Pooling and Batch Normalization Layers:

The following is the architecture of the CNN used throughout the project.

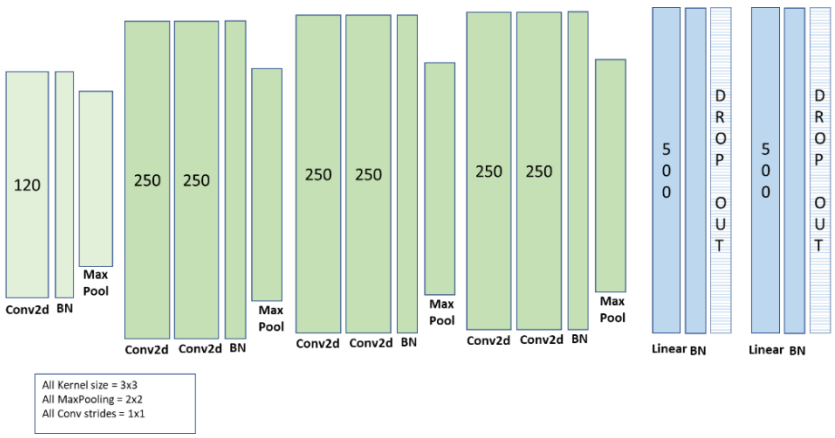


Figure 6: CNN designed for the project

In this step we define the convolutional neural network where the convolution, **Pooling**, and **Fattening** layers will be applied. We also added **Dropout layer** to reduce overfitting. Since it's a binary class problem, the **Sigmoid**

activation function is applied in the last layer. **Batch normalization** is used to keep output close to mean of 0 and variance of 1 which prevents the saturations of the activation function

Step 4: Model Compilation, optimizer, and loss function:
The next step is to compile the model with the following parameters:

- **Adam** as an optimizer with small learning rate of 0.0001 (which is adapted by the algorithm) was the best performance of the model and known for its computational efficiencies with images
- **Binary Cross-entropy** loss is used because the labels are binary.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Y is the label and p(y) is the probability of y to belong to certain class and N is number of points.
In the event that you want to encode the labels, then you will have to use the Categorical Cross-Entropy loss function.

Step 5: Training the model:
The model is set to run for 50 epochs, but the process will be stopped by the callback when the loss doesn't improve after 5 epochs as we defined in the Early Stopping callback to avoid overfitting. A batch size is set to 48 to minimize any memory allocation problems with large images and maintain a good rate of updating the weight based on avg batch gradient.

Once the training is completed, historical losses and accuracies obtained throughout the different epochs will be displayed for model assessment.

Learning Transfer:

As part of the experimental setup in the project, pre-trained **VGG19** model using Keras API was explored. This model and others like (Xception, ResNet50 & MobileNet) have been pre-trained on the ImageNet dataset which has over million images. This provides our model with a very relevant weight initialization which provide a better generalization and convergence likelihood.

The approach used in the transfer learning process is to freeze the base layer and add a set of linear layers as a head model to be trained and classify the images.

Model performance measurement

For model performance assessment, SK-learn package was used to obtain the confusion matrix, recall, precision, and accuracy.
Accuracy: ratio of all correct classification against all data
Precision: percentage of classification that are relevant
Recall: percentage of relevant classification

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \qquad Precision = \frac{TP}{TP+FP} \qquad Recall = \frac{TP}{TP+FN}$$

Model Stacking and Ensembling

Due to the usage of multiple filters and different learning approaches (i.e., transfer learning), there was a need to utilize the best performance from all the models created throughout the training phase. To leverage the different models, the following two techniques were examined:

1. Voting Committee: where best performing models will be used for prediction and then the final prediction will be determined by majority vote by the models.

```
# Create a new column to hold the sum of all models
df['SUM'] = df['Model-P3'] + df['Model-04'] + df['Model-07'] + df['Model-P1'] + df['Model-09']

# Create a new column that checks if the SUM of all models >= 3
# which means (3 or more models predicted the image as photoshopped)
df['Final'] = 0
df.loc[df.SUM >= 3, "Final"] = 1
```

2. MLP Classifier: Collect all predictions from top performing models and train a Multi-Layer Perceptron to predict the final classification

```
def train_stacked_model(df_all):
    """ This function process the dataframe that contains all models predictions to train a MLP
    Keyword arguments:
    df_all -- dataframe with all model predictions
    """
    train_features = df_all.copy()
    train_labels = train_features.pop('Photoshopped')
    train_features = train_features.iloc[:, 1:]

    model = Sequential()
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(500, activation='relu'))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    check_point = tf.keras.callbacks.ModelCheckpoint('model_{}.h5'.format('stacked_01'),
```

Implementation Approach Summary

The following figure summarizes the full implementation approach of the project:

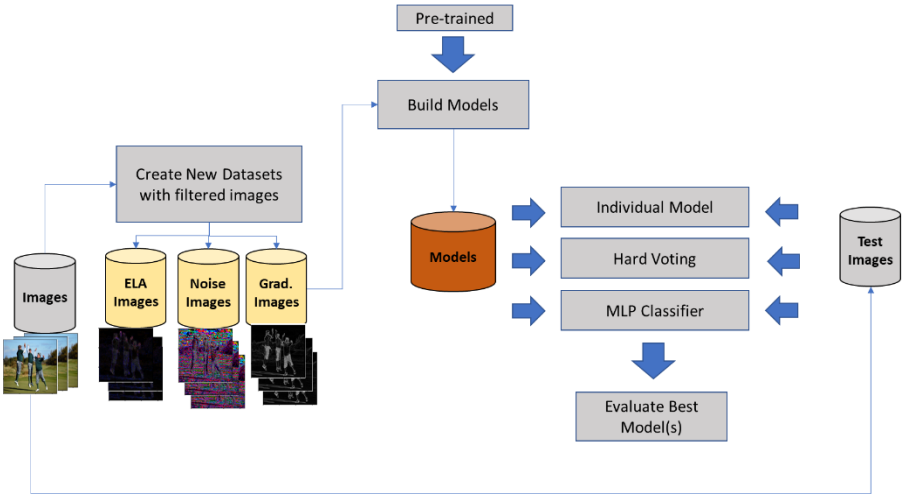
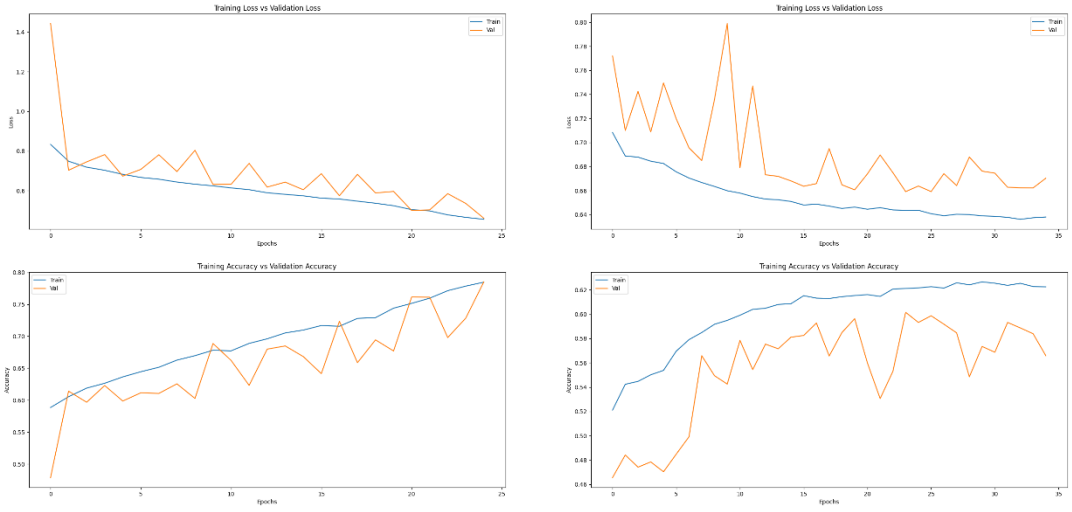


Figure 7: Implementation approach summary

5. Results and Observations

With early experimental models, it was obvious that ELA image is providing better performance compared to image datasets (normal image, noise analysis, gradient analysis other).



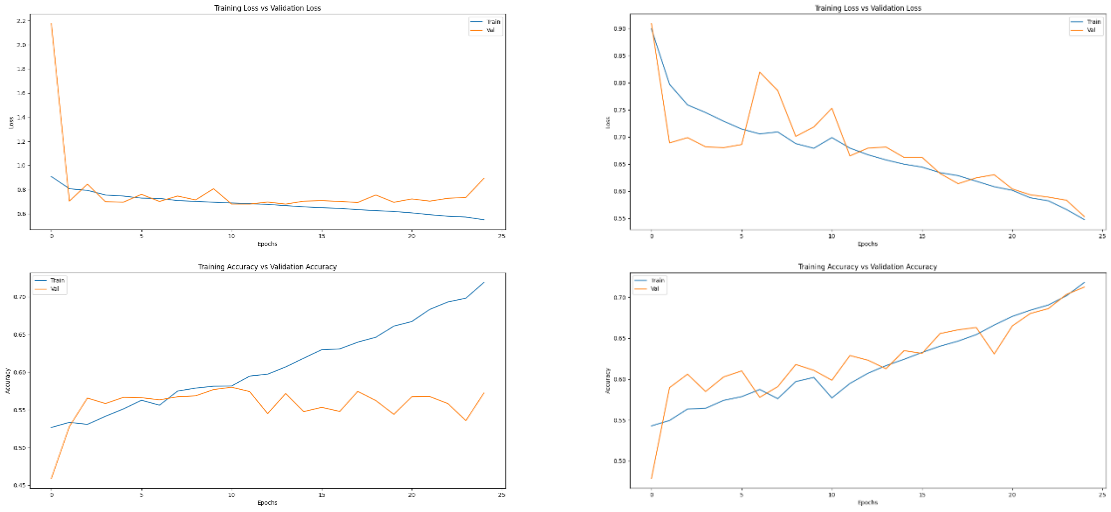


Figure 8: Model performance: ELA images top left, Noise images top right, Gradient images bottom left and normal images bottom right

Other filters like noise analysis or gradient analysis was not performing well. The ELA using different configuration and pre-trained weight was never reaching above 70% accuracy for the validation dataset. By analyzing the the different images that were misclassified, the following was observed:

- Noise and gradient filters output might be able to spot some distortions when tuned up based on the image (i.e. outdoor bright image might have less noise and filter will need to be more sensitive) so using fixed filter settings will not work with all images
- ELA on the other side is focused on the JPEG losses with every compression (when being saved) so it has less sensetivity to the overall image noise or lighting conditions
- ELA might fail with images with a lot of JPEG saving or low resolution images
- Noise analysis will not detect copy and move manipulation technique because the manilpulation is taking place by copying and moving areas from the same image with same noise distribution
- Images with focus and different depth might be misclassified as photoshopped
- Pre-trained network with small learning rate end up with overfitting and a higher validation loss. The high validation loss could be due to some outlier images in the dataset that keeps on getting predicted more badly and affect the loss function.

Training records

The following table highlights different model performance through the various experiments conducted for this project. Only VGG16 and VGG19 were used (the table below only shows the VGG19 records).

	Optimizer	lr	epochs	batch	T. Loss	T. Acc	V. Loss	V. Acc	Dataset
New CNN	Adamax	0.0005	25	48	0.6491	0.615	0.8001	0.6008	Normal Image
	adagrad	0.0005	50	48	0.4993	0.752	0.6172	0.6741	ELA image
	rmsprop	0.0005	50	48	0.4434	0.7949	0.8953	0.5777	Normal Image
	Adamax	0.0005	50	48	0.19	0.9209	0.9818	0.681	ELA image
	Adam	0.0001	50	48	0.19	0.832	0.22	**0.7141**	ELA Image
	Adam	0.0001	25	48	0.32	0.74	0.85	0.57	Grad
	Adam	0.0002	50	48	0.1217	0.9502	2.0401	0.5413	Noise image
	Adam	0.0001	25	48	0.5475	0.7184	0.5530	0.7127	Normal image
	Adam	0.0001	25	48	0.4555	0.7845	0.4608	**0.7851**	ELA image
VGG19	Adam	0.0005	50	48	0.2685	0.8349	1.8642	0.6379	ELA image
	Adam	0.0005	50	48	0.5351	0.6724	1.3972	0.5494	Noise image
	Adam	0.0005	50	48	0.3755	0.7717	1.6125	0.5837	Normal Image
	Adam	0.0001	50	48	0.1291	0.9498	1.6001	0.6165	ELA image
	Adam	0.0001	25	48	0.0818	0.9691	1.7308	0.6251	ELA image

It was observed that multiple training runs of the CNN with ELA image keeps on improving when we change the seed (new random initialization) and help get out of the local minimum reached in previous runs.

Aggregate Models Performance

As highlighted earlier, there were two techniques followed in the project to combine the different model performances and leverage the best of all. The following shows the performance measures of the best achieved model, hard voting committee and trained MLP model.

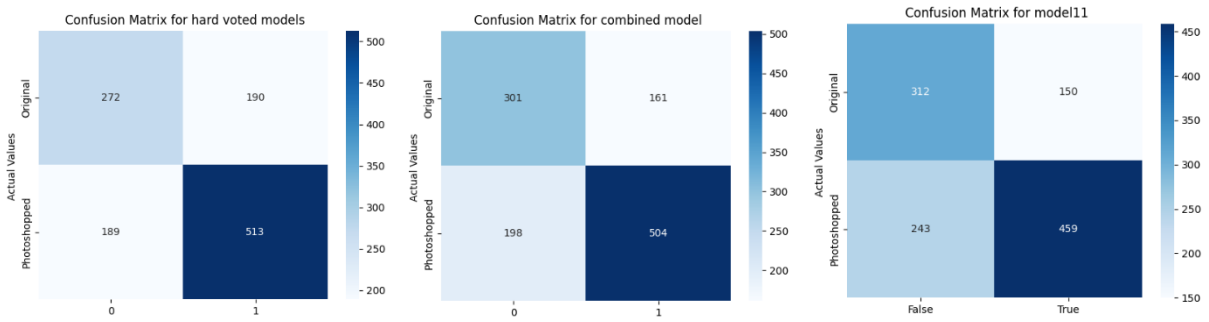


Figure 9: Confusion matrix of hard voting models, MLP stacked model and best standalone model

The following table has the weighted average performance scores

	Precision	Re-call	F1-Score
Best model	0.68	0.66	0.67
Hard Vote	0.67	0.67	0.67
MLP	0.7	0.69	0.69

Benchmarking

Through the project, the following referenced work was very useful to benchmark the work and build on the top of some of the lesson learned.

Publication Source	Methodology	Performance
IRACST - International Journal of Computer Science and Information Technology & Security (IJCSITS), Vol.7, No. 2, 2017 Fake Image Detection Using Machine Learning	MLP on ELA images from CASIA dataset + Metadata analysis to lookup image software names in the metadata	Reaching an accuracy of **80%**
Department of Computer Science, Cornell Tech , Department of Operations Research & Information Engineering, Cornell Tech Identifying Human Edited Images using a CNN	Focuses on face image editing and boost dataset using GAN network	Reaching accuracy of **61%** using GAN generated images in the training and leveraging ResNeXt as a pre-trained model Without the GAN images, accuracy was 51%
IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 10, No. 12, 2019 170 Detecting Fake Images on Social Media using Machine Learning	Usinga CNN and pre-trained network to train of a dataset of 1400 images	Reaching accuracy of **80.9%** on testing data when using AlexNet pre-trained network and 64.8 with a classic CNN

6. Conclusions and future work

There is no doubt that deep learning can support fake image detection and CNN is very well suited to fulfill the task, but the challenge will be to define and utilize necessary feature extraction filters to spot on different known techniques of image manipulations. The following are some suggestions to be added on the top of what was developed for this project:

- Leverage the metadata analysis to detect any digital fingerprints of photo editing software
- Customize filter tuning based on image brightness and other features to make sure that filters are maximizing spotting the anomalies by CNN
- Custom pipeline for known tampering techniques (i.e., move and copy)

7. References

[1] J. Fridrich, D. Soukal and J. Lukas, “Detection of Copy-move forgery in digital images”, in Digital Forensic Research Workshop, 2003.

[2] COMPREHENSIVE STUDY OF DIFFERENT TYPES IMAGE FORGERIES
http://www.ijstm.com/images/short_pdf/1440486816_229P.pdf

[3] http://aboutforensics.co.uk/edmond-locard/
[4] https://arxiv.org/abs/1804.04866
[5] https://bdtechtalks.com/2020/01/06/convolutional-neural-networks-cnn-convnets/
[6] Neural Network Design 2nd Edition: https://hagan.okstate.edu/nnd.html