THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Machine Learning II
Final Project Individual Report
by: Mina Hanna
Date: Dec-06-2021

1. Overview

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

In real life there is no one filter or tool that can cover all image manipulation techniques available today especially with the presence of very sophisticated tool that help to fabricate such image.

My first contribution to the project started with proposing the idea which fulfilled different criteria:

- Meeting all final project requirement where CNN will be utilized to classify a massive dataset of 100K images
- The topic has a big room for contribution given the evolving nature of manipulation techniques and the powerful computation power available to train very complex deep neural networks
- The dataset was not heavily used on online implementations
- The topic is overall interesting and has a potential enhancement and further use cases to be covered.

2. Individual work

Core focus of my contribution was the EDA process, support with the model training and combining all results in model stacking and the following sections shows what I contributed to in more detail:

- **Basic image processing**: refreshing knowledge on image processing and cv2 to be able to extract necessary features from image to train our CNN model:
  - **ELA**: save the jpeg image with known quality and obtain the difference between the original and the newly save

```
def create_ela(img):
    cv2.imwrite("temp.jpg", img, [cv2.IMWRITE_JPEG_QUALITY, 95])
    img2 = cv2.imread("temp.jpg")
    cv2.imwrite("temp.jpg", img2, [cv2.IMWRITE_JPEG_QUALITY, 90])
    img2 = cv2.imread("temp.jpg")
    diff = 15 * cv2.absdiff(img, img2)
    return diff
```

  - **Noise**: One if the simplest forms to capture the noise of the image is to apply a denoising filter and subtract the output of this filter from the original image.

```
def create_noise_analysis(img):
    #img2 = cv2.medianBlur(img, 3)
    img2 = cv2.fastNlMeansDenoisingColored(img, h=1)
    return img - img2
```

  - Gradient: the following steps are used to calculate the gradient magnitude of the image:
    - Calculate the gradient in both x and y direction by convoluting kernels Gx and Gy over image I

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

- Calculate gradient at each point of the image:
  G=|Gx|+|Gy|

- **Data exploration:** Coding simple functions to help explore and tune up the data.
- **Data preprocessing:** creating os functions to copy data and perform a solid pipeline for the models
- **Contribution on selecting Convolutional Neural Network**
- **Multi-run training and model stacking:**
  Built three python files to achieve the following:
  1. Measure performance of each model
  2. Apply hard voting techniques for top 5 models
  3. Train MLP to predict the best output based on the different predictions from the best models

3. Results and observations (on the top what shared by the team final report)**:**

- With low learning rate model is not reaching an overfitting state and may be with more epochs it could achieve better performance
- Pre-trained model was not performing well especially on filtered data which might be because the filtered images have no meaning and is not anywhere close to any image such networks used to train
- The oscillations in the validation loss and accuracy compared to the training. This might be due to regularization or due to wrong prediction are getting worse and increase the loss function
- The hard voting requires three (or more) models to share the same decision:

FileName	Photoshopped	size -mb	Shape	Model-P3	Model-04	Model-07	Model-P1	Model-09	SUM	Final
10092l.jpg	0	0.24442	(2048, 1363, 3)	1	1	0	0	0	2	0
100c1k.jpg	0	0.471768	(1200, 1600, 3)	1	0	0	0	0	1	0
100d24.jpg	0	0.363097	(807, 1200, 3)	0	1	0	0	1	2	0
100jh1.jpg	0	0.154267	(1080, 1620, 3)	0	1	1	0	0	2	0
100qo2.jpg	0	0.197824	(750, 1200, 3)	0	0	1	0	1	2	0
100yc6.jpg	0	0.087696	(643, 960, 3)	0	1	1	0	0	2	0
101453.jpg	0	0.109316	(706, 500, 3)	1	0	1	0	1	3	1
101j5w.jpg	0	0.078157	(720, 1280, 3)	1	0	0	1	1	3	1

- Size of the image might be a factor in the best model selection (i.e., low resolution and low size images will not be detected by ELA filter) so in the MLP training we did on the various model, size was added as a parameter:

	size-mb	Model-P3	Model-P1	Model-04	Model-07	Model-09	Model-11	Photoshopped
0	0.19226933	0	0	0	0	0	0	0
1	0.72533894	0	0	1	0	1	0	0
2	0.11198235	0	0	1	0	0	0	0
3	0.08490181	0	0	1	0	0	0	0
4	0.07178402	0	0	1	0	0	0	0
5	0.03704929	0	0	1	1	0	0	0
6	0.04709148	0	0	1	0	0	0	0

4. Key Learning Outcomes

- Model ensembling
- Basic openCV2 functions
- Pre-trained model is not the solution for everything. Some problems (like our filtered images) will not work well with pre-trained models

5. Code contributions

- Data preprocessing, model ensembling, hard voting and load predict and measure files
- Around 25-30% of the code was taken from the internet and exam 1 pipeline, code copies were mainly:
    - Basic cv2 functions from documentation or stackoverflow
    - TensorFlow examples for MLP, plotting accuracy and loss and loading data
    - Exam 1 predict function and
    - Sk-learn (i.e. plot confusion matrix)