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

1. Overview

Working with Detecting Fake Images classification is one of the most challenging problems in and time consuming in the field of Deep learning and ML because of the advances and different techniques that been used recently to manipulated the Images. With said there is no specific route our model you can follow to get a good result.

Our focus in this project is to split the work in a way that keep the harmony of the group and teamwork and not to redo each of other work.

I was focused on budling the CNN model and to fit the model over the different dataset we created using different filters:

1- Error Level Analysis (ELA)
2- Noise Analysis
3- Gradient Analysis

2. The dataset and the Exploratory Data Analysis

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

3. Model Development and Evolution Approach

The approach followed in this project is to identify the different data pre-processing, model design and parameters that will need to be tuned and progressively update them and monitor the performance of the model.

1- Build CNN Model Architecture
2- Use Learning Transfer (Pretrained Model)

4. Model definition and building network:

First we will start with the CNN model we designed using TensorFlow, and then will talk about the Pertained model.

Step 1: Data Preprocessing:

As many of you already know working with Images forensic fakes classification there is no straight way to use a model that fit with all types of photoshopped or fakes images so we start to apply different filters to the images and apply model to see which model work best with which type of filter to give a better results.

So we end up with 4 different class of data:

1- Class01 – Original Images with no filter
2- Class02 – ELA (Error Level Analysis)
3- Class03 – Noise Analysis
4- Class04 – Grading Analysis

Step 2: Prepare Dataset for Training

Create Training & Validation Data:
We divided the data into two groups: one for training and the other for model validation in 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)
```

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

Step 3: Images Augmentation:

We define a sperate layer of Image augmentation and 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 4: Scheduler:

Create a Scheduler function that help to dynamically assign Learning Rates while the model optimizing the weights, although some optimizer like “Adam” has built-in features to adjust the “LR” while the training process.

```
def scheduler(epoch, lr):
  lr_0 = lr
  lr_max = 0.001
  lr_0_steps = 5
  lr_max_steps = 5
  lr_min = 0.0005
  lr_decay = 0.9
  if epoch < lr_0_steps:
    lr = lr_0 + ((lr_max - lr_min) / lr_0_steps) * (epoch - 1)
  elif epoch < lr_0_steps + lr_max_steps:
    lr = lr_max
  else:
    lr = max(lr_max * lr_decay ** (epoch - lr_0_steps - lr_max_steps), lr_min)
  return lr
```

Step 5: Build the convolutional neural network:

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.

```
model = Sequential([
  data_augmentation,
  layers.Conv2D(60, 3, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.Conv2D(120, 3, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Conv2D(200, 3, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Conv2D(200, 3, padding='same', activation='relu'),
  layers.Conv2D(100, 3, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dropout(0.2),
  layers.Dense(128, activation='relu'),
  layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adagrad',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

Step 6 : Compiling the model:

The next step is to compile the model. The Binary Crossentropy loss is used because the labels are binary. In the event that you want to encode the labels, then you will have to use the Categorical Cross-Entropy loss function.

```
model.compile(optimizer='adagrad',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

Step 7 : Early Stopping, Model Check Point :

Early stopping: To avoid that is to stop the training process when the model stops improving

Model Check Point : automatically saving the best model or model weights during training

```
early_stop = tf.keras.callbacks.EarlyStopping(monitor='accuracy', patience=5)
check_point = tf.keras.callbacks.ModelCheckpoint('model_{}.h5'.format('07'),
                                                 monitor='accuracy',
                                                 save_best_only=True)
```

Step 8: Training the model:

Now we are ready to fit the data to the training set. 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.

```
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs,
  callbacks=[early_stop, check_point, callback]
)
```

Pretrained Model:

In this project we used transfer learning choose to use **VGG16** model using Keras API. This model and others like (Xception, ResNet50 & MobileNet) have been pre-trained on the ImageNet dataset which has over million images.

```
base_model = tf.keras.applications.VGG16(weights = 'imagenet', include_top = False,
                                         input_shape = (img_height, img_width, 3))
```

Show the deep learning model results:

```
fig, axs = plt.subplots(2, 1, figsize=(15, 15))
axs[0].plot(history.history['loss'])
axs[0].plot(history.history['val_loss'])
axs[0].title.set_text('Training Loss vs Validation Loss')
axs[0].set_xlabel('Epochs')
axs[0].set_ylabel('Loss')
axs[0].legend(['Train','Val'])
axs[1].plot(history.history['accuracy'])
axs[1].plot(history.history['val_accuracy'])
axs[1].title.set_text('Training Accuracy vs Validation Accuracy')
axs[1].set_xlabel('Epochs')
axs[1].set_ylabel('Accuracy')
axs[1].legend(['Train', 'Val'])

plt.show()
```

	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