Design Lab

# Classification of Twitter image having infrastructural damage

Rohan Rajak
14CS30029

Instructor: Dr.Soumya Kanti Ghosh

Professor , Computer Science and Engineering

## Introduction and overview

- During natural and man-made disasters, people use social media platforms such as Twitter to post textual and multimedia content to report updates about injured or dead people, infrastructure damage, and missing or found people among other information types.

- Studies have revealed that this online information, if processed timely and effectively, is extremely useful for humanitarian organizations to gain situational awareness and plan relief operations.

- In addition to the analysis of textual content, recent studies have shown that imagery content on social media can boost disaster response significantly.

- This work demonstrate the ability of deep learning techniques to classify a image having any infrastructural damage (trained on twitter image).

# Description of the dataset

The CrisisMMD multimodal Twitter dataset consists of several thousands of manually annotated tweets and images collected during seven major natural disasters including earthquakes, hurricanes, wildfires, and floods that happened in the year 2017 across different parts of the World. The provided datasets include three types of annotations. I worked on task 2 to classify if there is any **Infrastructure and utility damage** in a given twitter image.

** Informative vs Not informative

* Informative
* Not informative
* Don't know or can't judge

** Task 2: Humanitarian categories

* Affected individuals
* Infrastructure and utility damage
* Injured or dead people
* Missing or found people
* Rescue, volunteering or donation effort
* Vehicle damage
* Other relevant information
* Not relevant or can't judge

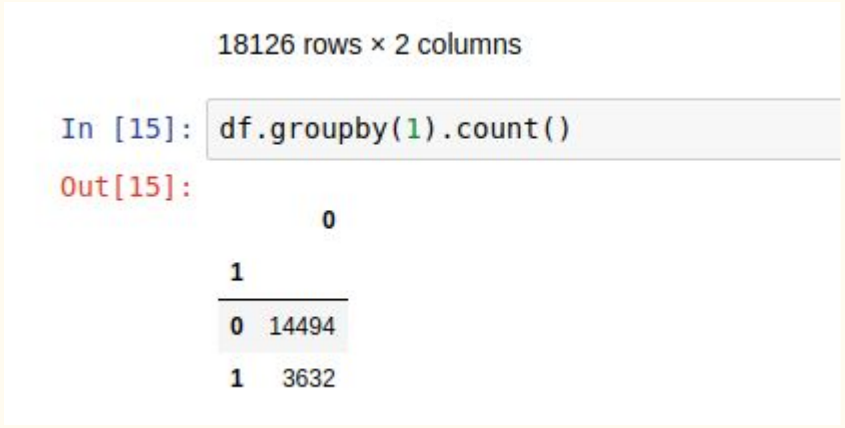
** Task 3: Damage severity assessment

* Severe damage
* Mild damage
* Little or no damage
* Don't know or can't judge

## The Experiment

# Problems while processing image data

- Each image were of different dimensions, some were very low dimension icons and others were HD image with large pixel matrix.
- There were a total of 18126 images of both positive and negative classes but the positive class were very less.



```
18126 rows × 2 columns

In [15]: df.groupby(1).count()
Out[15]:
       0
1
0  14494
1   3632
```

## Comparison of various approaches used and concluding the best.

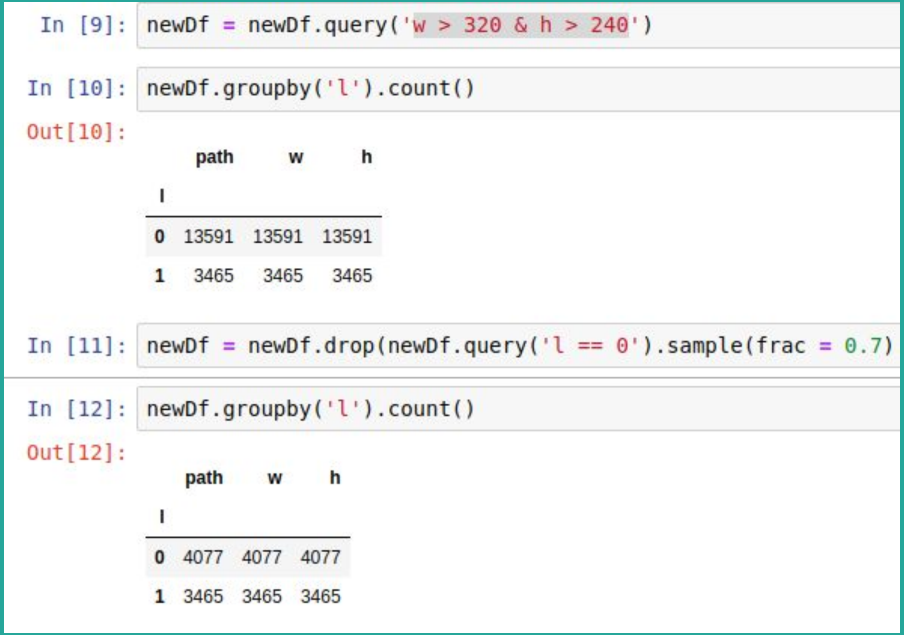
# Approach 1:

1. **Taking all the images of dimension**

   **w > 320 & h > 240**

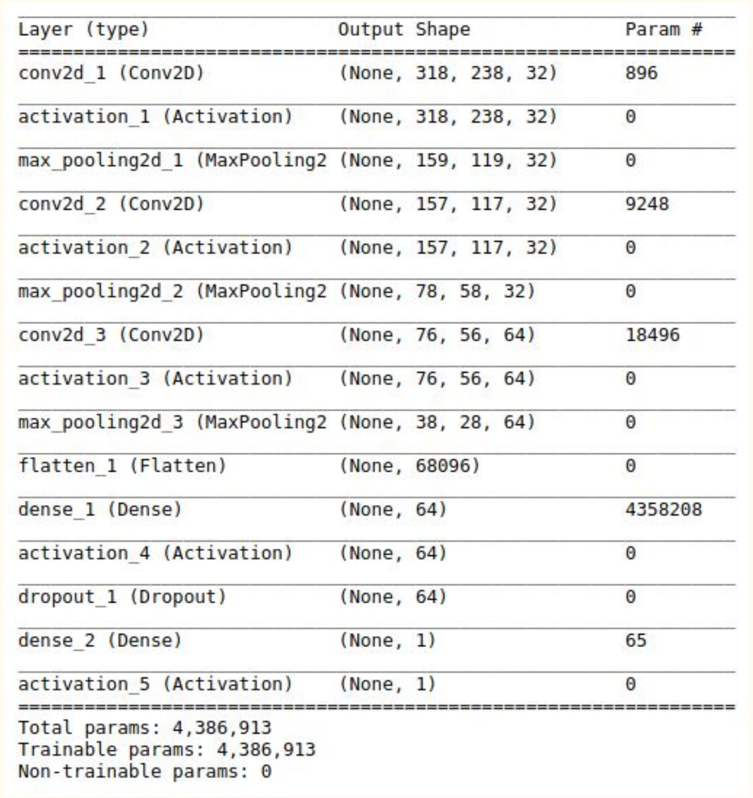
   **works as this is just the size that would eliminate all small thumbnail images which won't provide much info for classification.**

2. **For solving the less data ratio problem, I just dropped the negative class which was large in number by a fraction of 0.7, to make both the classes almost similar.**

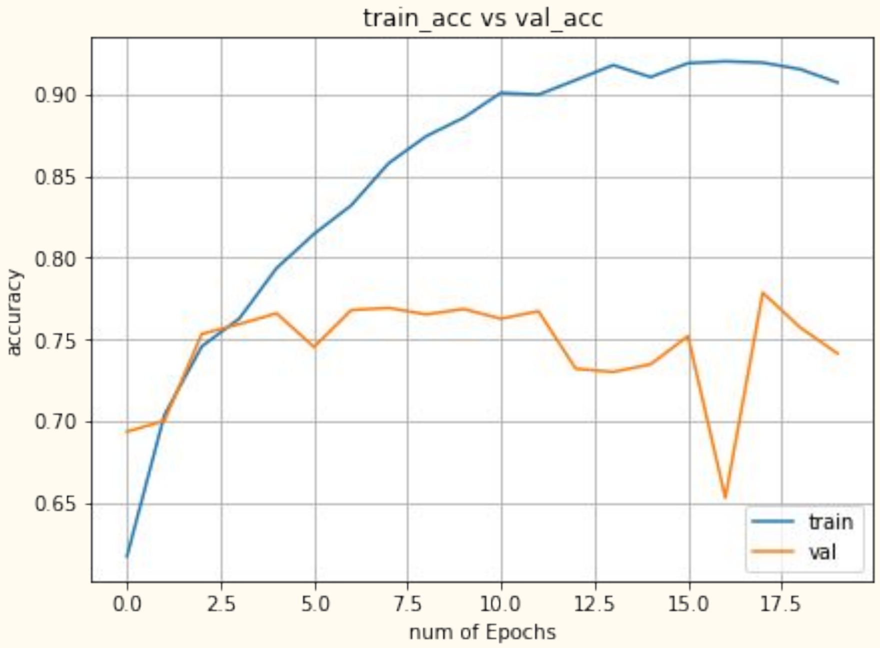


```
In [9]:  newDf = newDf.query('w > 320 & h > 240')

In [10]: newDf.groupby('l').count()
Out[10]:
       path      w      h
   l
   0  13591  13591  13591
   1   3465   3465   3465

In [11]: newDf = newDf.drop(newDf.query('l == 0').sample(frac = 0.7)

In [12]: newDf.groupby('l').count()
Out[12]:
      path     w     h
   l
   0  4077  4077  4077
   1  3465  3465  3465
```

# Model Overview and learning graph of approach 1:



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 318, 238, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 318, 238, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 159, 119, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 157, 117, 32)      9248
_________________________________________________________________
activation_2 (Activation)    (None, 157, 117, 32)      0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 78, 58, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 76, 56, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 76, 56, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 38, 28, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 68096)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4358208
_________________________________________________________________
activation_4 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_5 (Activation)    (None, 1)                 0
=================================================================
Total params: 4,386,913
Trainable params: 4,386,913
Non-trainable params: 0
```



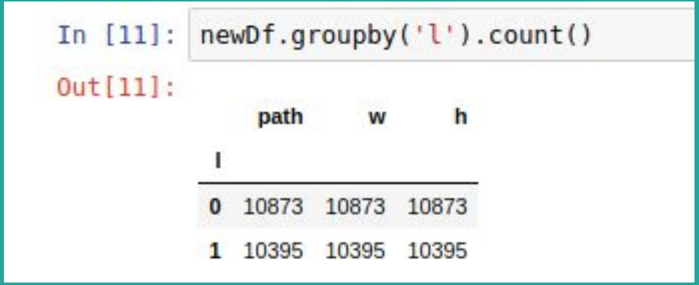
# Approach 2:

1. **This part remains same.**

   3 channels RGB, pixel levels in range [0–255]

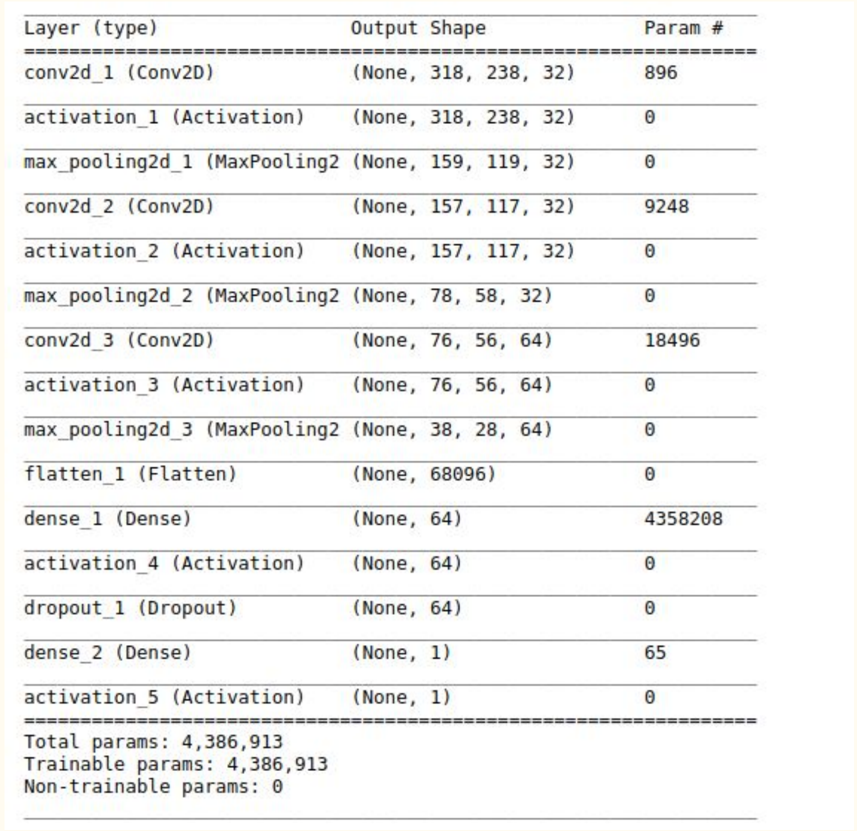
   Dimension: **320 x 240 x 3**

2. **But here rather randomly dropping the data which could have valuable info, to match the number of data point for both the classes I rotated the positive classed image by angle angle of -45 and +45 degrees. (And this is intuitive as the image clicked can be tilted sometimes)**

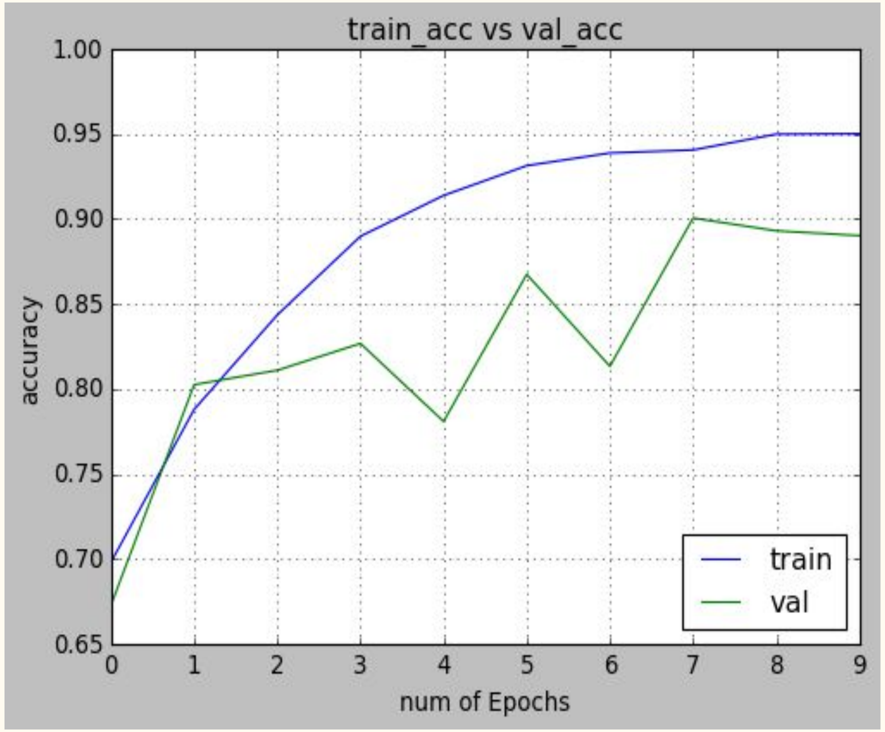


```
In [11]: newDf.groupby('l').count()
Out[11]:
       path      w      h
 l
 0    10873  10873  10873
 1    10395  10395  10395
```

# Model Overview and learning graph of approach 2:



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 318, 238, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 318, 238, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 159, 119, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 157, 117, 32)      9248
_________________________________________________________________
activation_2 (Activation)    (None, 157, 117, 32)      0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 78, 58, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 76, 56, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 76, 56, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 38, 28, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 68096)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4358208
_________________________________________________________________
activation_4 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_5 (Activation)    (None, 1)                 0
=================================================================
Total params: 4,386,913
Trainable params: 4,386,913
Non-trainable params: 0
_________________________________________________________________
```



## Result and conclusion

The second approach does a much better job hence I save the model and any image can be classified as having a infrastructure damage with 89% correctness.The same method can be used as a baseline for the classification other tasks such as

* Affected individuals
  * Injured or dead people
  * Missing or found people
  * Vehicle damage

But the parameters and the structure of processed data feeding into the model should be taken care accordingly.

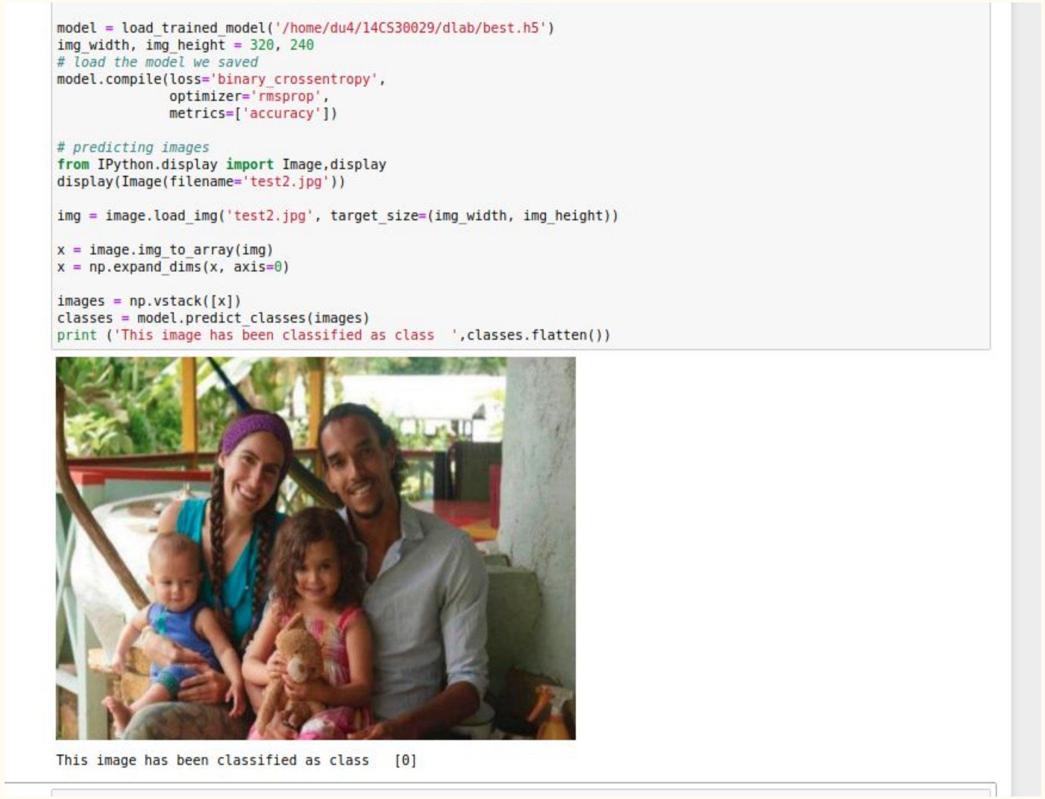
# Practical trails

1

0

1

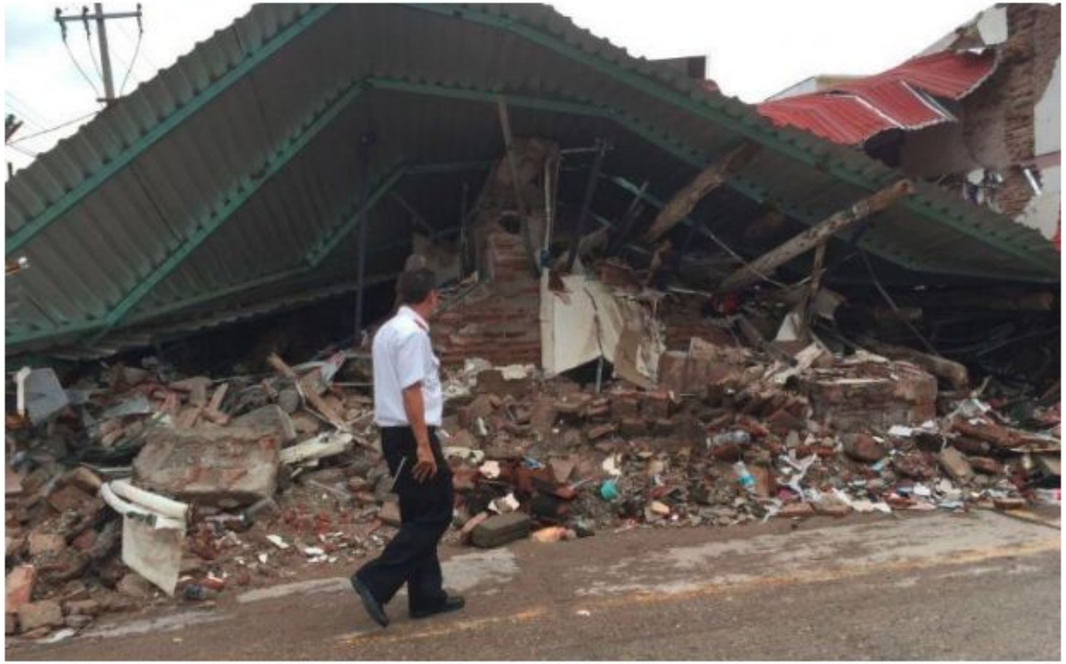
## Live results with other images found online



```
model = load_trained_model('/home/du4/14CS30029/dlab/best.h5')
img_width, img_height = 320, 240
# load the model we saved
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

# predicting images
from IPython.display import Image,display
display(Image(filename='test2.jpg'))

img = image.load_img('test2.jpg', target_size=(img_width, img_height))

x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)

images = np.vstack([x])
classes = model.predict_classes(images)
print ('This image has been classified as class  ',classes.flatten())
```

```
This image has been classified as class   [0]
```

```
# predicting images
from IPython.display import Image,display
display(Image(filename='test3.jpg'))

img = image.load_img('test3.jpg', target_size=(img_width, img_height))

x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)

images = np.vstack([x])
classes = model.predict_classes(images)
print ('This image has been classified as class  ',classes.flatten())
```



```
This image has been classified as class   [1]
```