In [1]:
```
### Satellite Images of Hurricane Damage
```

Overview

The data are satellite images from Texas after Hurricane Harvey divided into two groups (damage and no_damage). The goal is to make a model which can automatically identify if a given region is likely to contain flooding damage.

Source

Data originally taken from: https://ieee-dataport.org/open-access/detecting-damaged-buildings-post-hurricane-satellite-imagery-based-customized and can be cited with http://dx.doi.org/10.21227/sdad-1e56 and the original paper is here: https://arxiv.org/abs/1807.01688

In [1]:
```
##!mkdir ~/.kaggle
```

In [2]:
```
 ##!cp /kaggle.json ~/.kaggle/
```

In [3]:
```
 ####!chmod 600 ~/.kaggle/kaggle.json
```

In [ ]:
```
 ####! pip install kaggle
```

In [ ]:
```
 ####!pip install keras-tuner
```

In [ ]:
```
###!kaggle datasets download -d kmader/satellite-images-of-hurricane-damage
```

In [ ]:
```
####! unzip /content/satellite-images-of-hurricane-damage.zip
```

In [8]:
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
```

In [9]:
```
print(tf.__version__)
```

```
2.7.0
```

In [10]:
```
# import the libraries as shown below

from tensorflow.keras.layers import Input, Lambda, Dense, Flatten,Conv2D
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img
from tensorflow.keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt
```

In [11]:
```
# re-size all the images to this
IMAGE_SIZE = [224, 224]

train_path = '/content/train_another'
valid_path = '/content/test_another'
valid_path2 = '/content/validation_another'
valid_path3 = '/content/test'
```

In [12]:
```
# Import the Vgg 16 library as shown below and add preprocessing layer to the front of VGG
# Here we will be using imagenet weights

vggnet = VGG19(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 1s 0us/step
80150528/80134624 [==============================] - 1s 0us/step
```

In [13]:
```
# don't train existing weights
for layer in vggnet.layers:
    layer.trainable = False
```

In [14]:
```
  # useful for getting number of output classes
folders = glob('/content/train_another/*')
```

In [15]:
```
folders
```

Out[15]:
```
['/content/train_another/no_damage', '/content/train_another/damage']
```

In [16]:
```
# our layers - you can add more if you want
x = Flatten()(vggnet.output)
```

In [17]:
```
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=vggnet.input, outputs=prediction)
```

In [18]:
```
# view the structure of the model
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv4 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv4 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv4 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 2)                 50178

=================================================================
```

```
Total params: 20,074,562
Trainable params: 50,178
Non-trainable params: 20,024,384
```

In [19]:

```
# tell the model what cost and optimization method to use
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

In [20]:

```
# Use the Image Data Generator to import the images from the dataset
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

In [21]:

```
# Make sure you provide the same target size as initialied for the image size
training_set = train_datagen.flow_from_directory('/content/train_another',
                                                 target_size = (224, 224),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')
```

```
Found 10000 images belonging to 2 classes.
```

In [22]:

```
training_set
```

Out[22]:

```
<keras.preprocessing.image.DirectoryIterator at 0x7f9f110a3350>
```

In [23]:

```
test_set = test_datagen.flow_from_directory('/content/test_another',
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')
```

```
Found 9000 images belonging to 2 classes.
```

In [24]:

```
# fit the model
# Run the cell. It will take some time to execute
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=30,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
```

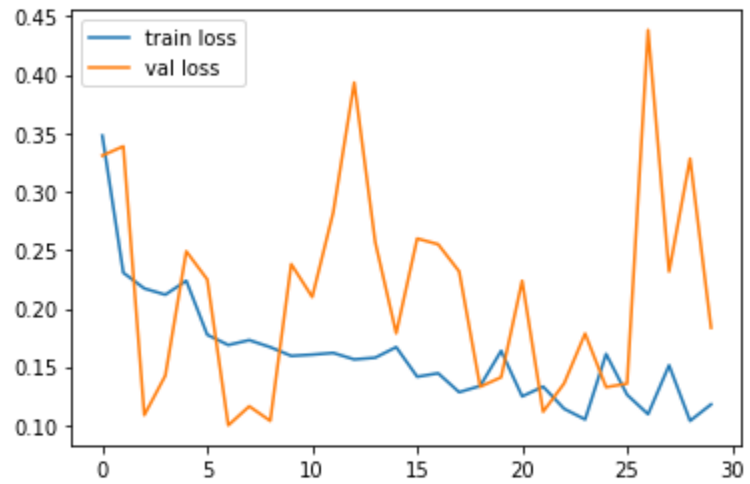
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, whic
```

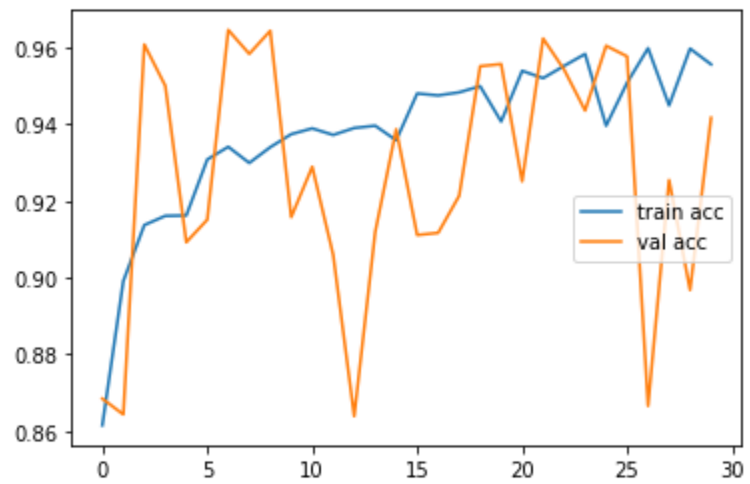
```
h supports generators.
```

```
Epoch 1/30
313/313 [==============================] - 253s 747ms/step - loss: 0.3480 - accuracy: 0.8615 - val_loss: 0.3309 - val_accuracy: 0.8684
Epoch 2/30
313/313 [==============================] - 223s 712ms/step - loss: 0.2306 - accuracy: 0.8991 - val_loss: 0.3388 - val_accuracy: 0.8643
Epoch 3/30
313/313 [==============================] - 223s 714ms/step - loss: 0.2172 - accuracy: 0.9137 - val_loss: 0.1088 - val_accuracy: 0.9608
Epoch 4/30
313/313 [==============================] - 218s 698ms/step - loss: 0.2118 - accuracy: 0.9161 - val_loss: 0.1425 - val_accuracy: 0.9500
Epoch 5/30
313/313 [==============================] - 221s 705ms/step - loss: 0.2237 - accuracy: 0.9162 - val_loss: 0.2490 - val_accuracy: 0.9092
Epoch 6/30
313/313 [==============================] - 221s 707ms/step - loss: 0.1775 - accuracy: 0.9308 - val_loss: 0.2248 - val_accuracy: 0.9151
Epoch 7/30
313/313 [==============================] - 220s 702ms/step - loss: 0.1688 - accuracy: 0.9341 - val_loss: 0.1001 - val_accuracy: 0.9646
Epoch 8/30
313/313 [==============================] - 219s 699ms/step - loss: 0.1730 - accuracy: 0.9299 - val_loss: 0.1163 - val_accuracy: 0.9583
Epoch 9/30
313/313 [==============================] - 220s 703ms/step - loss: 0.1670 - accuracy: 0.9340 - val_loss: 0.1039 - val_accuracy: 0.9643
Epoch 10/30
313/313 [==============================] - 219s 698ms/step - loss: 0.1595 - accuracy: 0.9374 - val_loss: 0.2380 - val_accuracy: 0.9158
Epoch 11/30
313/313 [==============================] - 219s 700ms/step - loss: 0.1606 - accuracy: 0.9389 - val_loss: 0.2099 - val_accuracy: 0.9289
Epoch 12/30
313/313 [==============================] - 219s 700ms/step - loss: 0.1620 - accuracy: 0.9372 - val_loss: 0.2822 - val_accuracy: 0.9059
Epoch 13/30
313/313 [==============================] - 216s 690ms/step - loss: 0.1565 - accuracy: 0.9390 - val_loss: 0.3934 - val_accuracy: 0.8639
Epoch 14/30
313/313 [==============================] - 223s 712ms/step - loss: 0.1579 - accuracy: 0.9396 - val_loss: 0.2569 - val_accuracy: 0.9119
Epoch 15/30
313/313 [==============================] - 222s 709ms/step - loss: 0.1670 - accuracy: 0.9358 - val_loss: 0.1788 - val_accuracy: 0.9387
Epoch 16/30
313/313 [==============================] - 224s 715ms/step - loss: 0.1417 - accuracy: 0.9480 - val_loss: 0.2599 - val_accuracy: 0.9111
Epoch 17/30
313/313 [==============================] - 221s 707ms/step - loss: 0.1446 - accuracy: 0.9475 - val_loss: 0.2550 - val_accuracy: 0.9117
Epoch 18/30
313/313 [==============================] - 221s 707ms/step - loss: 0.1284 - accuracy: 0.9483 - val_loss: 0.2317 - val_accuracy: 0.9213
Epoch 19/30
313/313 [==============================] - 223s 714ms/step - loss: 0.1337 - accuracy: 0.9499 - val_loss: 0.1331 - val_accuracy: 0.9551
Epoch 20/30
313/313 [==============================] - 221s 707ms/step - loss: 0.1638 - accuracy: 0.9407 - val_loss: 0.1412 - val_accuracy: 0.9557
Epoch 21/30
313/313 [==============================] - 215s 687ms/step - loss: 0.1248 - accuracy: 0.9539 - val_loss: 0.2237 - val_accuracy: 0.9251
Epoch 22/30
313/313 [==============================] - 213s 679ms/step - loss: 0.1332 - accuracy: 0.9520 - val_loss: 0.1117 - val_accuracy: 0.9623
Epoch 23/30
313/313 [==============================] - 216s 690ms/step - loss: 0.1142 - accuracy: 0.9552 - val_loss: 0.1359 - val_accuracy: 0.9542
Epoch 24/30
313/313 [==============================] - 218s 696ms/step - loss: 0.1051 - accuracy: 0.9583 - val_loss: 0.1786 - val_accuracy: 0.9436
Epoch 25/30
313/313 [==============================] - 220s 702ms/step - loss: 0.1611 - accuracy: 0.9396 - val_loss: 0.1327 - val_accuracy: 0.9604
Epoch 26/30
313/313 [==============================] - 219s 699ms/step - loss: 0.1262 - accuracy: 0.9508 - val_loss: 0.1358 - val_accuracy: 0.9577
Epoch 27/30
313/313 [==============================] - 219s 699ms/step - loss: 0.1096 - accuracy: 0.9598 - val_loss: 0.4384 - val_accuracy: 0.8666
Epoch 28/30
313/313 [==============================] - 218s 698ms/step - loss: 0.1514 - accuracy: 0.9449 - val_loss: 0.2316 - val_accuracy: 0.9254
Epoch 29/30
313/313 [==============================] - 218s 696ms/step - loss: 0.1039 - accuracy: 0.9597 - val_loss: 0.3283 - val_accuracy: 0.8968
Epoch 30/30
313/313 [==============================] - 219s 699ms/step - loss: 0.1179 - accuracy: 0.9556 - val_loss: 0.1836 - val_accuracy: 0.9417
```

In [25]:
```
# plot the loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# plot the accuracy
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```






```
<Figure size 432x288 with 0 Axes>
```

In [26]:
```
# save it as a h5 file

from tensorflow.keras.models import load_model

model.save('model_vgg19_new.h5')
```

In [27]:
```
y_pred = model.predict(test_set)
```

In [28]:
```
y_pred
```

Out[28]:
```
array([[9.6935892e-01, 3.0641133e-02],
       [9.9955755e-01, 4.4239406e-04],
       [9.9619859e-01, 3.8013996e-03],
       ...,
       [9.8282832e-01, 1.7171687e-02],
       [8.3030403e-01, 1.6969596e-01],
       [5.7293600e-01, 4.2706403e-01]], dtype=float32)
```

In [29]:
```
import numpy as np
y_pred = np.argmax(y_pred, axis=1)
```

In [30]:
```
# Evaluating model on validation data
evaluate = model.evaluate(test_set)
print(evaluate)
```

```
282/282 [==============================] - 80s 285ms/step - loss: 0.1836 - accuracy: 0.9417
[0.18358109891414642, 0.9416666626930237]
```

In [34]:
```
from sklearn.metrics import classification_report, confusion_matrix
def give_accuracy():
    p=model.predict(test_set)
    cm=confusion_matrix(y_true=test_set.classes,y_pred=np.argmax(p,axis=-1))
    acc=cm.trace()/cm.sum()
    print('The Classification Report \n', cm)
    print(f'Accuracy: {acc*100}')
```

In [36]:
```
give_accuracy()
```

```
The Classification Report
 [[6768 1232]
 [ 837  163]]
Accuracy: 77.0111111111111
```

In [46]:
```
import numpy as np
from tensorflow.keras.preprocessing import image
test_image = image.load_img('/content/test_another/damage/-93.560702_30.766426.jpeg', target_size = (224,224))
test_image = image.img_to_array(test_image)
test_image=test_image/255
test_image = np.expand_dims(test_image, axis = 0)
result = model.predict(test_image)
```

In [47]:
```
test = np.array(test_image)
```

In [48]:
```
# making predictions
#prediction = np.argmax(cnn.predict(test_image), axis=-1)
prediction = np.argmax(model.predict(test_image))
```

In [49]:
```
prediction
```

Out[49]: 0

In [50]:
```
OUTPUT = {0:'DAMAGE',1:'NON DAMAGE'}
```

In [51]:
```
print("The prediction Of the Image is : ", OUTPUT[prediction])
```

```
The prediction Of the Image is :  DAMAGE
```

In [52]:
```
# show the image
import matplotlib.pyplot as plt
test_image = image.load_img('/content/test_another/damage/-93.560702_30.766426.jpeg', target_size = (224,224))
plt.axis('off')
plt.imshow(test_image)
plt.show()
```

In [53]:
```
##### Similarly
```

In [54]:
```
import numpy as np
from tensorflow.keras.preprocessing import image
test_image = image.load_img('/content/validation_another/no_damage/-95.062123_30.056714000000003.jpeg', target_size = (224,224))
test_image = image.img_to_array(test_image)
test_image=test_image/255
test_image = np.expand_dims(test_image, axis = 0)
result = model.predict(test_image)
```

In [55]:
```
test = np.array(test_image)
```

In [56]:
```
# making predictions
#prediction = np.argmax(cnn.predict(test_image), axis=-1)
prediction = np.argmax(model.predict(test_image))
```

In [57]:
```
prediction
```

Out[57]: 1

In [58]:
```
OUTPUT = {0:'DAMAGE',1:'NON DAMAGE'}
```

In [59]:
```
print("The prediction Of the Image is : ", OUTPUT[prediction])
```

```
The prediction Of the Image is :  NON DAMAGE
```

In [60]:
```
# show the image
import matplotlib.pyplot as plt
test_image = image.load_img('/content/validation_another/no_damage/-95.062123_30.056714000000003.jpeg', target_size = (224,224))
plt.axis('off')
plt.imshow(test_image)
plt.show()
```