Severstal: Steel Defect Detection

Steel is one of the most important building materials of modern times. The physical properties of steel include high strength, low weight, durability, ductility and resistance to corrosion. Due to these properties of steel, buildings are resistant to natural and man-made wear which has made the material ubiquitous around the world.

Severstal is among the top 50 producers of steel in the world and produced 12.04 and 11.8 Million tonnes of steel in 2018 and 2019 respectively. It is one among Russia's biggest players in efficient steel mining and production. The company recently created a hybrid Data Lake as part of its digital strategy to secure the Company's competitive advantages in the long-term. The infrastructure is designed to store Company functional data files for subsequent processing and use in Severstal's data analysis, machine learning and artificial intelligence projects. Severstal is now looking to machine learning to improve automation, increase efficiency, and maintain high quality in their production.

1. Business Problem

One of the key products of Severstal is steel sheets. The production process of flat sheet steel is delicate. From heating and rolling, to drying and cutting, several machines touch flat steel by the time it's ready to ship. To ensure quality in the production of steel sheets, today, Severstal uses images from high frequency cameras to power a defect detection algorithm.

Through this competition, Severstal expects the AI community to improve the algorithm by **localizing and classifying surface defects on a steel sheet**.

1.1. Business objectives and constraints

1. A defective sheet most be predicted as defective, since there would be serious concerns about quality if we misclassify a defective sheet as non-defective. i.e. high recall value for each of the classes is needed.
2. No strict latency concerns.

1.2. Sources / References

Kaggle competition page : https://www.kaggle.com/c/severstal-steel-defect-detection/overview

References :

- https://arxiv.org/pdf/1505.04597.pdf
- https://www.kaggle.com/cdeotte/keras-unet-with-eda
- https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient
- https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly

2. Machine Learning Probelm

2.1. Mapping the business problem to an ML problem

Our task is to

1. Detect/localize the defects in a steel sheet using image segmentation and
2. Classify the detected defects into one or more classes from [1, 2, 3, 4]

Therefore, it is a combination of image segementation and multiclass classification.

2.2. Performance metric

Evaluation metric used is the mean Dice coefficient. The Dice coefficient can be used to compare the pixel-wise agreement between a predicted segmentation and its corresponding ground truth. The formula is given by:

$$\frac{2 * |X \cap Y|}{|X| + |Y|}$$

where X is the predicted set of pixels and Y is the ground truth.

[Read more about Dice Coefficient]

2.3. Data Overview

We have been given a zip folder of size 2GB which contains the following:

- train_images/ - folder contailning 12,568 training images (.jpg files)
- test_images/ - folder containing 5506 test images (.jpg files). We need to detect and localize defect in these images.
- train.csv - training annotations which provide segments for defects belonging to ClassId = [1, 2, 3, 4]
- sample_submission.csv - a sample submission file in the correct format, with each ImageId repeated 4 times, one for each of the 4 defect classes

Refer to section 3: EDA for more details about data.

In [6]:

```
% cd /content/sample_data/SSD
```

```
/content/sample_data/SSD
```

Downloading and extracting data

In [5]:

```
!wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36" --header="A
ccept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/s
d-exchange;v=b3;q=0.9" --header="Accept-Language: en-IN,en-GB;q=0.9,en-US;q=0.8,en;q=0.7" --header
="Referer: https://www.kaggle.com/" "https://storage.googleapis.com/kaggle-competitions-
data/kaggle-v2/14241/862020/bundle/archive.zip?GoogleAccessId=web-data@kaggle-
161607.iam.gserviceaccount.com&Expires=1585378453&Signature=h2F1NAm4R%2BE47ciKVCqoApTJuHaq0irbA7Yk1
PhA3Ap6%2Fam2%2BK%2BBB5BNRCsz7oRumb16ZVH%2BHGu3TYeXLhHbvrsD%2BdbpV85mf4YzaEnqmhD3Bk3RGj3%2BHC%2B9%2
xFA1h60vxEiBCWs0YriI5UwblPZ0GFVAxdd3L2yNJGR5Jl36%2FVP7MkdSeXOdP4SjFysy1kldd4OujZbiCu%2FspOxCZo3UCRS
QVVJaG7PPNXABkp%2FzIzzE7yEI9SqnByvPQO3Kzt965r3e38Qo9XcXHya32v4KGKKN3fN3KiQis%2B4qJf55azjryGTYo8MOcE
Iw0wZbCXvRxntmEw%3D%3D&response-content-disposition=attachment%3B+filename%3Dseverstal-steel-defec
t-detection.zip" -O "severstal-steel-defect-detection.zip" -c
```

```
--2020-03-26 11:52:32--  https://storage.googleapis.com/kaggle-competitions-data/kaggle-
v2/14241/862020/bundle/archive.zip?GoogleAccessId=web-data@kaggle-
161607.iam.gserviceaccount.com&Expires=1585378453&Signature=h2F1NAm4R%2BE47ciKVCqoApTJuHaq0irbA7Yk1
PhA3Ap6%2Fam2%2BK%2BBB5BNRCsz7oRumb16ZVH%2BHGu3TYeXLhHbvrsD%2BdbpV85mf4YzaEnqmhD3Bk3RGj3%2BHC%2B9%2
xFA1h60vxEiBCWs0YriI5UwblPZ0GFVAxdd3L2yNJGR5Jl36%2FVP7MkdSeXOdP4SjFysy1kldd4OujZbiCu%2FspOxCZo3UCRS
QVVJaG7PPNXABkp%2FzIzzE7yEI9SqnByvPQO3Kzt965r3e38Qo9XcXHya32v4KGKKN3fN3KiQis%2B4qJf55azjryGTYo8MOcE
Iw0wZbCXvRxntmEw%3D%3D&response-content-disposition=attachment%3B+filename%3Dseverstal-steel-
defect-detection.zip
Resolving storage.googleapis.com (storage.googleapis.com)... 108.177.127.128,
2a00:1450:4013:c03::80
Connecting to storage.googleapis.com (storage.googleapis.com)|108.177.127.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1684204253 (1.6G) [application/zip]
Saving to: 'severstal-steel-defect-detection.zip'

severstal-steel-def 100%[===================>]   1.57G   165MB/s    in 8.3s

2020-03-26 11:52:40 (193 MB/s) - 'severstal-steel-defect-detection.zip' saved
[1684204253/1684204253]
```

In [7]:

In [7]:

```
from zipfile import ZipFile
file_name="severstal-steel-defect-detection.zip"

with ZipFile(file_name,'r') as zip:
  zip.extractall()
  print('Done')
```

```
Done
```

In [1]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

Importing libraries

In [9]:

```
## Importing required packages
import warnings
warnings.filterwarnings("ignore")
from datetime import datetime
import os
import gc
import pickle

from tqdm import tqdm_notebook as tqdm

import pandas as pd
import numpy as np
import math
from numpy import asarray
import cv2
from os import listdir
import random

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

from PIL import Image

from sklearn import metrics

from collections import Counter
from collections import defaultdict
from sklearn.model_selection import train_test_split

import tensorflow as tf
from keras import backend as K
from keras import Model
from keras.layers import UpSampling2D, Conv2D, Activation, LeakyReLU,
BatchNormalization,Input,Conv2DTranspose,Dropout
from keras.layers.pooling import MaxPooling2D, GlobalMaxPool2D
from keras.layers.merge import concatenate,add

from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
from keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint,
LearningRateScheduler,Callback

from keras.optimizers import Adam
from keras.losses import binary_crossentropy
```

The default version of TensorFlow in Colab will switch to TensorFlow 2.x on the 27th of March, 2020.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

```
Using TensorFlow backend.
```

3. Exploratory Data Analysis

3.1. Loading train.csv file

In [11]:

```
# loading the train.csv file containing pixels indicating defects
train_df= pd.read_csv("train.csv")
train_df.head()
```

Out[11]:

	ImageId	ClassId	EncodedPixels
0	0002cc93b.jpg	1	29102 12 29346 24 29602 24 29858 24 30114 24 3...
1	0007a71bf.jpg	3	18661 28 18863 82 19091 110 19347 110 19603 11...
2	000a4bcdd.jpg	1	37607 3 37858 8 38108 14 38359 20 38610 25 388...
3	000f6bf48.jpg	4	131973 1 132228 4 132483 6 132738 8 132993 11 ...
4	0014fce06.jpg	3	229501 11 229741 33 229981 55 230221 77 230468...

- **ImageID**: image file name
- **ClassID**: type/class of the defect [1, 2, 3, 4]
- **EncodedPixels**: represent the range of defective pixels in an image in the form of run-length encoded pixels(pixel number where defect starts <'space'> pixel length of defect). E.g. '**29102 12**' implies the defect is starting at pixel 29102 and running a total of 12 pixels, i.e. pixels 29102, 29103,........., 29113 are defective. The pixels are numbered from top to bottom, then left to right: 1 is pixel (1,1), 2 is pixel (2,1), etc.

The competition requires the submission file to contain predicted *ClassID* and *EncodedPixels* for each test_image, in the same format as given in train.csv.

In [12]:

```
train_df.ImageId.describe()
```

Out[12]:

```
count              7095
unique             6666
top       ef24da2ba.jpg
freq                  3
Name: ImageId, dtype: object
```

- There are 7095 datapoints correspondingto 6666 steel sheet images containing defects

Check for null values

In [13]:

```
train_df[train_df.isnull().any(axis=1)]
```

Out[13]:

	ImageId	ClassId	EncodedPixels

There are no null values in train.csv

3.2. Analysing train & test images folders

3.2.1. Number of train & test images

In [14]:

```
train_count= 0
test_count= 0

for filename in os.listdir('train_images'):
    train_count+=1
for filename in os.listdir('test_images'):
    test_count+=1

print("Number of train images : ",train_count)
print("Number of test images : ",test_count)
```

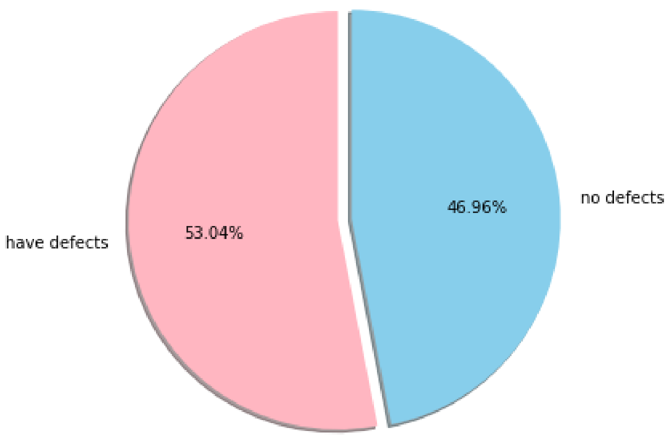
```
Number of train images :  12568
Number of test images :  5506
```

There are more images in train_images folder than unique image Ids in train.csv. This means, not all the images in train_folder have at least one of the defects 1, 2, 3, 4.

In [15]:

```
print("Number of non-defective images in the train_images folder:", train_count-train_df.ImageId.n
unique())

#Pie-chart https://pythonspot.com/matplotlib-pie-chart/
# Data to plot
labels = 'have defects', 'no defects'
sizes = [train_df.ImageId.nunique(), train_count-train_df.ImageId.nunique()]
explode = (0.1, 0)  # explode 1st slice

# Plot
plt.pie(sizes, explode=explode, labels=labels, colors=['lightpink', 'skyblue'], autopct='%1.2f%%',
shadow=True, startangle=90, radius=1.5)

# plt.axis('equal')
plt.show()
```

```
Number of non-defective images in the train_images folder: 5902
```

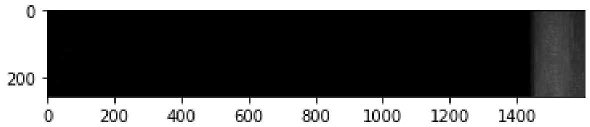


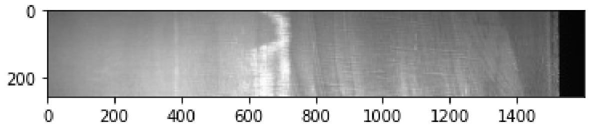
Let's see some images that we are categorizing as non-defective.

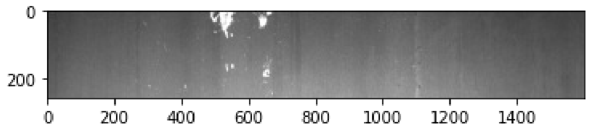
In [0]:

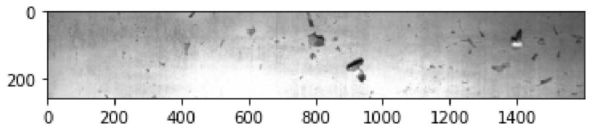
```
random.seed(42)

defective = set(train_df.ImageId.values)
non_defective = set(listdir('train_images')) - defective
for filename in random.sample(non_defective, 5):
```

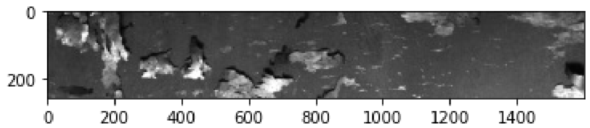
```
    # load image
    img = cv2.imread('train_images/' + filename)
    plt.imshow(img)
    plt.grid(False)
    plt.show()
```











Pictures suggest that there are defects in some of these images too. May be these defects do not belong to one of the four categories [1,2,3,4] *(let's assume this for simplicity)*

3.2.2. Check if all images in train and test are of the same size

Train images

In [0]:

```
image_size = [] #list of tuples containing shape(height, width, channel) of images
for image_id in tqdm_notebook(listdir('train_images')):
    img = cv2.imread("train_images/"+image_id)
    h, w, c = img.shape

    image_size.append((h, w, c))
```

In [0]:

```
# print unique values in image_size list
set(image_size)
```

Out[0]:

```
{(256, 1600, 3)}
```

All train images have same size: 256 x 1600 x 3

Test images

In [0]:

```
test_image_size = [] #list of tuples containing shape(height, width, channel) of images
for image_id in tqdm_notebook(listdir('test_images')):
    img = cv2.imread("test_images/"+image_id)
    h, w, c = img.shape

    test_image_size.append((h, w, c))
```

In [0]:

```
# print unique values in test_image_size list
set(test_image_size)
```

Out[0]:

```
{(256, 1600, 3)}
```

All images in train and test folders have the same size (256 x 1600 x 3)

3.3. Analysis of labels: ClassId

3.3.1. Checking for class count

In [0]:

```
counts = train_df.ClassId.value_counts()
print(counts)
```

```
3    5150
1     897
4     801
2     247
Name: ClassId, dtype: int64
```

In [0]:

```
my_colors = list('rgbkymc')
counts.plot(kind='bar', color=my_colors, rot=0)
plt.xlabel('Class')
plt.ylabel('Count of images')
plt.title('Class-wise count of images')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_counts = np.argsort(-counts.values)
for i in sorted_counts:
    print('Number of images in class', i+1, ':',counts.values[i], '(', np.round((counts.values[i]/t
rain_df.ImageId.nunique()*100), 3), '%)')
```

```
Number of images in class 1 : 5150 ( 77.258 %)
Number of images in class 2 : 897 ( 13.456 %)
Number of images in class 3 : 801 ( 12.016 %)
Number of images in class 4 : 247 ( 3.705 %)
```

- The dataset looks imbalanced.
- Number of images with class 3 defect is very high compared to that of other classses. 77% of the defective images have class 3 defect.
- Class 2 is the least occuring class, only 3.7 % images in train.csv belong to class 2.

The Sum of percentage values in the above analysis is more than 100, which means some images have defects belonging to more than one class.

3.3.2. Checking number of labels tagged per image

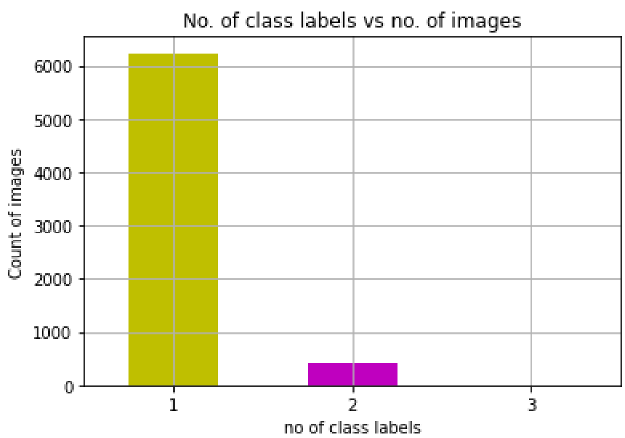
In [0]:

```
labels_per_image = train_df.groupby('ImageId')['ClassId'].count().value_counts()
print(labels_per_image)
```

```
1    6239
2     425
3       2
Name: ClassId, dtype: int64
```

In [0]:

```
my_colors = list('ymcrgbk')
labels_per_image.plot(kind='bar', color=my_colors, rot=0)
plt.xlabel('no of class labels')
plt.ylabel('Count of images')
plt.title('No. of class labels vs no. of images')
plt.grid()
plt.show()

sorted_i = np.argsort(-labels_per_image.values)
for i in sorted_i:
    print(f'Number of images having {i+1} class label(s): {labels_per_image.values[i]}
({np.round((labels_per_image.values[i]/train_df.ImageId.nunique()*100), 3)}%)')
```




```
Number of images having 1 class label(s): 6239 (93.594%)
Number of images having 2 class label(s): 425 (6.376%)
Number of images having 3 class label(s): 2 (0.03%)
```

Observations:

- Majority of the images (93.6%) have only one class of defects.
- Only 2 images (0.03%) have a combination of 3 classes of defects.
- Rest of the images (6.37%) have a combination of 2 classes of defects.

- Rest of the images (0.[illegible]7 %) have a combination of 2 classes of defects.
- No images have all 4 classes of defects.

3.3.3. Let's take look at the images having defects of each of the 4 classes

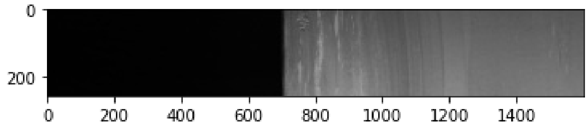
In [0]:

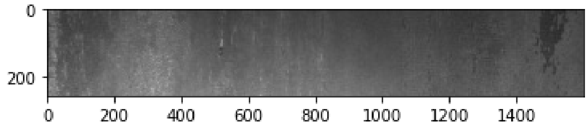
```
d1= set(train_df[train_df.ClassId==1].ImageId.values)
d2= set(train_df[train_df.ClassId==2].ImageId.values)
d3= set(train_df[train_df.ClassId==3].ImageId.values)
d4= set(train_df[train_df.ClassId==4].ImageId.values)
```

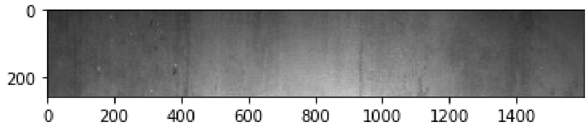
Class 1

In [0]:

```
random.seed(42)
for filename in random.sample(d1, 3):
    # load image
    img = cv2.imread('train_images/' + filename)
    plt.imshow(img)
    plt.grid(False)
    plt.show()
```



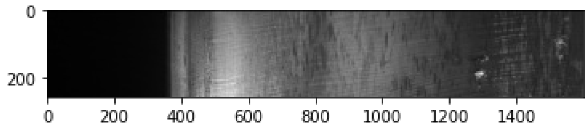


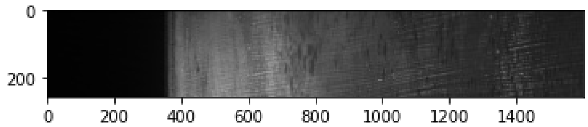


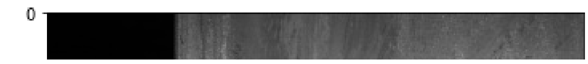
Class 2

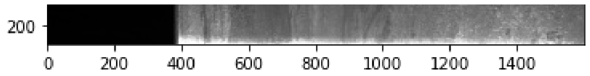
In [0]:

```
random.seed(42)
for filename in random.sample(d2, 3):
    # load image
    img = cv2.imread('train_images/' + filename)
    plt.imshow(img)
    plt.grid(False)
    plt.show()
```





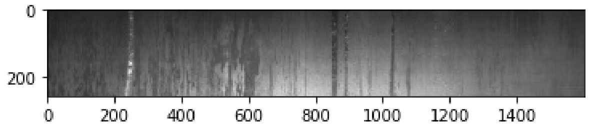


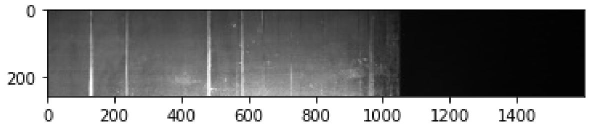


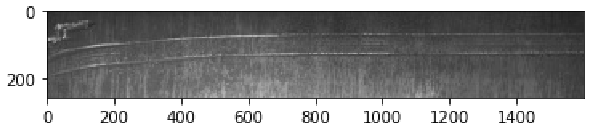
Class 3

In [0]:

```
random.seed(121)
for filename in random.sample(d3, 3):
    # load image
    img = cv2.imread('train_images/' + filename)
    plt.imshow(img)
    plt.grid(False)
    plt.show()
```



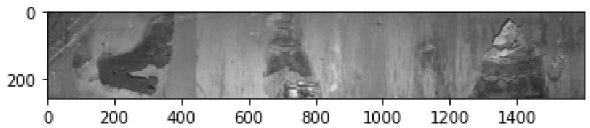


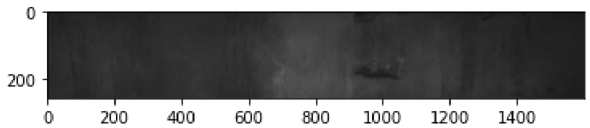


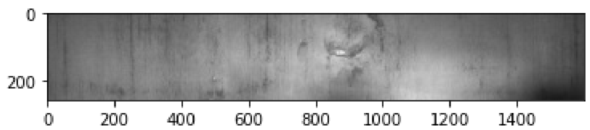
Class 4

In [0]:

```
random.seed(42)
for filename in random.sample(d4, 3):
    # load image
    img = cv2.imread('train_images/' + filename)
    plt.imshow(img)
    plt.grid(False)
    plt.show()
```







Images belonging to 2 classes

In [0]:

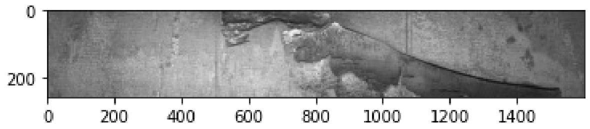
```
labels_per_image = train_df.groupby('ImageId')['ClassId'].count().reset_index(name='count')
```

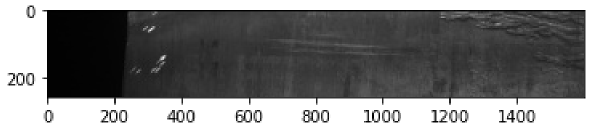
In [0]:

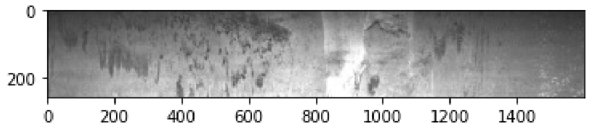
```
d5 = set(labels_per_image[labels_per_image['count']==2].ImageId.values)
```

In [0]:

```
random.seed(42)
for filename in random.sample(d5, 3):
    # load image
    img = cv2.imread('train_images/' + filename)
    plt.imshow(img)
    plt.grid(False)
    plt.show()
```







Images belonging to 3 classes

In [0]:

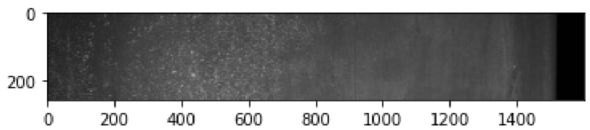
```
d6 = labels_per_image[labels_per_image['count']==3]
d6
```

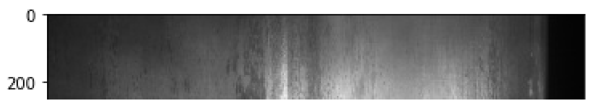
Out[0]:

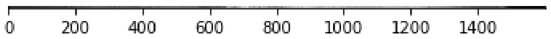
	ImageId	count
5740	db4867ee8.jpg	3
6253	ef24da2ba.jpg	3

In [0]:

```
for filename in set(d5.ImageId.values):
    # load image
    img = cv2.imread('train_images/' + filename)
    plt.imshow(img)
    plt.grid(False)
    plt.show()
```







4. Data preparation

In [16]:

```
images= []
class_id= []
for img in listdir('train_images'):
    images.append(img)
    class_id.append(1)
    images.append(img)
    class_id.append(2)
    images.append(img)
    class_id.append(3)
    images.append(img)
    class_id.append(4)
train_images= pd.DataFrame(images,columns=['ImageId'])
train_images['ClassId'] = class_id
print('train_images shape:', train_images.shape)
train_images.head()
```

```
train_images shape: (50272, 2)
```

Out[16]:

	ImageId	ClassId
0	ba4008245.jpg	1
1	ba4008245.jpg	2
2	ba4008245.jpg	3
3	ba4008245.jpg	4
4	7b0b85b1d.jpg	1

In [17]:

```
train_df.head(3)
```

Out[17]:

	ImageId	ClassId	EncodedPixels
0	0002cc93b.jpg	1	29102 12 29346 24 29602 24 29858 24 30114 24 3...
1	0007a71bf.jpg	3	18661 28 18863 82 19091 110 19347 110 19603 11...
2	000a4bcdd.jpg	1	37607 3 37858 8 38108 14 38359 20 38610 25 388...

In [18]:

```
# Create a dataframe that conatains all the images(defective and non_defective)
all_df = pd.merge(train_images, train_df, how='outer',on=['ImageId','ClassId'])
all_df = all_df.fillna('')
print(all_df.shape)
all_df[50:55]
```

```
(50272, 3)
```

Out[18]:

	ImageId	ClassId	EncodedPixels
50	13b66e5de.jpg	3	
51	13b66e5de.jpg	4	187146 6 187402 12 187658 13 187914 14 188170 ...

	ImageId	ClassId	EncodedPixels
52	972837f87.jpg	1	
53	972837f87.jpg	2	
54	972837f87.jpg	3	

In [19]:

```
#https://www.geeksforgeeks.org/python-pandas-pivot_table/
all_df = pd.pivot_table(all_df, values='EncodedPixels', index='ImageId',columns='ClassId', aggfunc=
np.sum).astype(str)
all_df = all_df.reset_index()
all_df.columns = ['ImageId','Defect_1','Defect_2','Defect_3','Defect_4']
all_df.head()
```

Out[19]:

	ImageId	Defect_1	Defect_2	Defect_3	Defect_4
0	0002cc93b.jpg	29102 12 29346 24 29602 24 29858 24 30114 24 3...			
1	00031f466.jpg				
2	000418bfc.jpg				
3	000789191.jpg				
4	0007a71bf.jpg			18661 28 18863 82 19091 110 19347 110 19603 11...	

In [0]:

```
all_df.to_csv("prep_data.csv", index=False)
```

In [21]:

```
data = pd.read_csv("prep_data.csv")
data.head()
```

Out[21]:

	ImageId	Defect_1	Defect_2	Defect_3	Defect_4
0	0002cc93b.jpg	29102 12 29346 24 29602 24 29858 24 30114 24 3...	NaN	NaN	NaN
1	00031f466.jpg	NaN	NaN	NaN	NaN
2	000418bfc.jpg	NaN	NaN	NaN	NaN
3	000789191.jpg	NaN	NaN	NaN	NaN
4	0007a71bf.jpg	NaN	NaN	18661 28 18863 82 19091 110 19347 110 19603 11...	NaN

In [22]:

```
# Replace NAs with blank spaces
data.fillna('', inplace=True)
data.head()
```

Out[22]:

	ImageId	Defect_1	Defect_2	Defect_3	Defect_4
0	0002cc93b.jpg	29102 12 29346 24 29602 24 29858 24 30114 24 3...			
1	00031f466.jpg				

	ImageId	Defect_1	Defect_2	Defect_3	Defect_4
2	000418bfc.jpg				
3	000789191.jpg				
4	0007a71bf.jpg			18661 28 18863 82 19091 110 19347 110 19603 11...	

4.2 Train, CV split 85:15

In [23]:

```
#splitting the data into train & cv
from sklearn.model_selection import train_test_split
train_data, cv_data = train_test_split(data, test_size=0.15, random_state=42)
print(train_data.shape)
print(cv_data.shape)
```

```
(10682, 5)
(1886, 5)
```

4.3. Generating data for Keras model

Train generator

In [0]:

```
# https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly
import keras
from keras.preprocessing.image import ImageDataGenerator
class Train_DataGenerator(keras.utils.Sequence):
    def __init__(self, df, batch_size = 32,shuffle=False,
                 preprocess=None, info={}):
        super().__init__()
        self.df = df
        self.shuffle = shuffle
        self.batch_size = batch_size
        self.preprocess = preprocess
        self.info = info
        self.data_path = 'train_images/'
        self.on_epoch_end()

    def __len__(self):
        return int(np.floor(len(self.df) / self.batch_size))

    def on_epoch_end(self):
        self.indexes = np.arange(len(self.df))
        if self.shuffle == True:
            np.random.shuffle(self.indexes)
    #fliping the images horizontally and normalization of samples
    def __getitem__(self, index):
        train_datagen = ImageDataGenerator()
        param = {'flip_horizontal':True, 'samplewise_std_normalization' : True}

        X = np.empty((self.batch_size,128,800,3),dtype=np.float32) #images
        y = np.empty((self.batch_size,128,800,4),dtype=np.int8)     #masks
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]
        for i,f in enumerate(self.df['ImageId'].iloc[indexes]):
            self.info[index*self.batch_size+i]=f
            img = Image.open(self.data_path + f).resize((800,128))
            X[i,] = train_datagen.apply_transform(x = img, transform_parameters = param)
                #run-length encoding on the pixel values
            for j in range(4):
                mask = rle2mask(self.df['Defect_'+str(j+1)].iloc[indexes[i]])
                y[i,:,:,j] = train_datagen.apply_transform(x = mask, transform_parameters = param)
        if self.preprocess!=None: X = self.preprocess(X)
        return X, y
```

Validation generator

In [0]:

```
# https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly
import keras
from keras.preprocessing.image import ImageDataGenerator
class Val_DataGenerator(keras.utils.Sequence):
    def __init__(self, df, batch_size = 32,shuffle=False,
                 preprocess=None, info={}):
        super().__init__()
        self.df = df
        self.shuffle = shuffle
        self.batch_size = batch_size
        self.preprocess = preprocess
        self.info = info
        self.data_path = 'train_images/'
        self.on_epoch_end()

    def __len__(self):
        return int(np.floor(len(self.df) / self.batch_size))

    def on_epoch_end(self):
        self.indexes = np.arange(len(self.df))
        if self.shuffle == True:
            np.random.shuffle(self.indexes)
    #fliping the images horizontally and normalization of samples
    def __getitem__(self, index):
        train_datagen = ImageDataGenerator()
        param = {'flip_horizontal':False, 'samplewise_std_normalization' : True}

        X = np.empty((self.batch_size,128,800,3),dtype=np.float32) #images
        y = np.empty((self.batch_size,128,800,4),dtype=np.int8)    #masks
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]
        for i,f in enumerate(self.df['ImageId'].iloc[indexes]):
            self.info[index*self.batch_size+i]=f
            img = Image.open(self.data_path + f).resize((800,128))
            X[i,] = train_datagen.apply_transform(x = img, transform_parameters = param)
                #run-length encoding on the pixel values
            for j in range(4):
                mask = rle2mask(self.df['Defect_'+str(j+1)].iloc[indexes[i]])
                y[i,:,:,j] = train_datagen.apply_transform(x = mask, transform_parameters = param)
        if self.preprocess!=None: X = self.preprocess(X)
        return X, y
```

4.4. Functions for converting RLE encoded pixels to masks and vice-versa

In [0]:

```
# https://www.kaggle.com/titericz/building-and-visualizing-masks

def rle2mask(rle):
    # CONVERT RLE TO MASK
    if (pd.isnull(rle))|(rle==''):
        return np.zeros((128,800) ,dtype=np.uint8)

    height= 256
    width = 1600
    mask= np.zeros( width*height ,dtype=np.uint8)

    array = np.asarray([int(x) for x in rle.split()])
    starts = array[0::2]-1
    lengths = array[1::2]
    for index, start in enumerate(starts):
        mask[int(start):int(start+lengths[index])] = 1

    return mask.reshape( (height,width), order='F' )[::2,::2]


# to convert masks to run length encoded values
def mask2rle(img):
    '''
    img: numpy array containing ones and zeros as pixel values, 1 - mask, 0 - background
    Returns String run length ecoded pixels
    '''
```

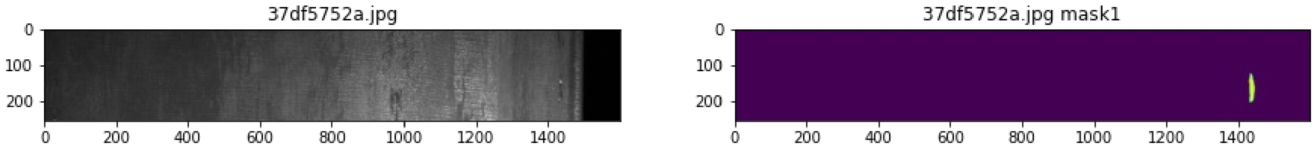
```
    pixels= img.T.flatten() # Convert nd-array to 1d-array (numbering of pixels is from top to bottom)
    pixels = np.concatenate([[0], pixels, [0]]) # Adding zeros at the start and end so that if there's mask at the first/last pixel, it gets detected.
    runs = np.where(pixels[1:] != pixels[:-1])[0] + 1 # Detect all changing pixels (where pixel values changes, either 0 -> 1 or 1 -> 0)

    # To get RLE, we need start pixels and run lengths
    # Start pixels are the pixels where change 0 -> 1 occurs, i.e. pixels at even indices
    # Run length is the pixel distance between two consecutive changing pixels. So, run lengths = odd indices - even indices
    runs[1::2] -= runs[::2]

    return ' '.join(str(x) for x in runs)
```

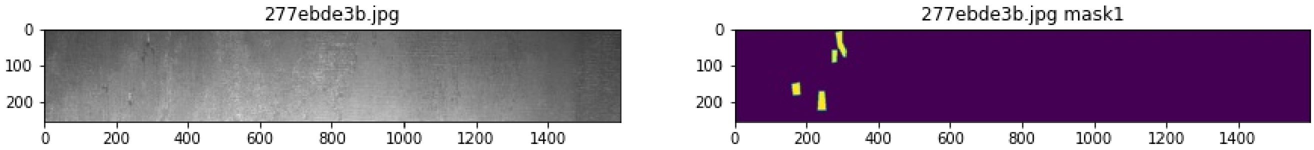
Check if the above functions are working fine

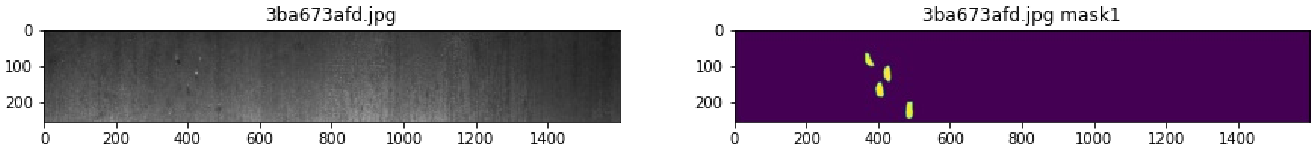
In [0]:

```
# Visualizing some images and their masks

for k in [1,2,3,4]:
    cnt=0
    print("Sample images with Class {} defect:".format(k))
    for i in train_data[train_data[f'Defect_{k}']!=''][['ImageId',f'Defect_{k}']].values:
        if cnt<3:
            fig, (ax1,ax2) = plt.subplots(nrows = 1,ncols = 2,figsize=(15, 7))
            img = cv2.imread('train_images/' + str(i[0]))
            ax1.imshow(img)
            ax1.set_title(i[0])
            cnt+=1
            ax2.imshow(rle2mask(i[1]))
            ax2.set_title(i[0]+' mask'+str(k))
            plt.show()
    print('='*100,'\n')
```

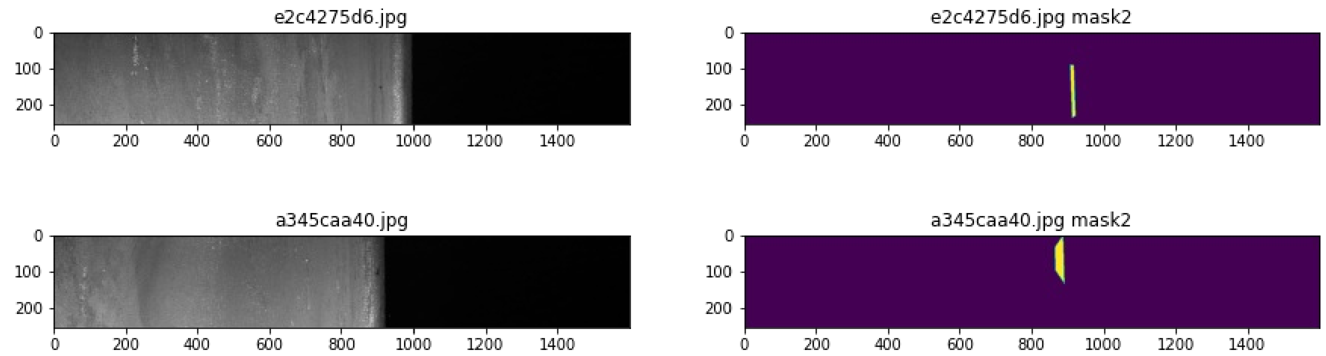
```
Sample images with Class 1 defect:
```





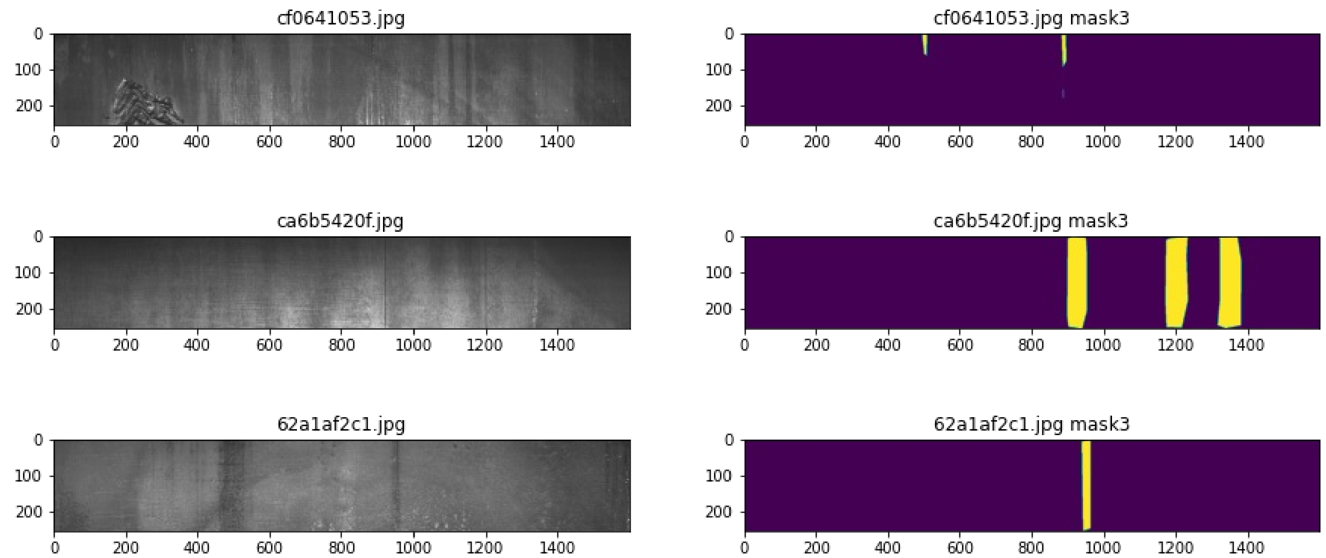



```
====================================================================================================

Sample images with Class 2 defect:
```



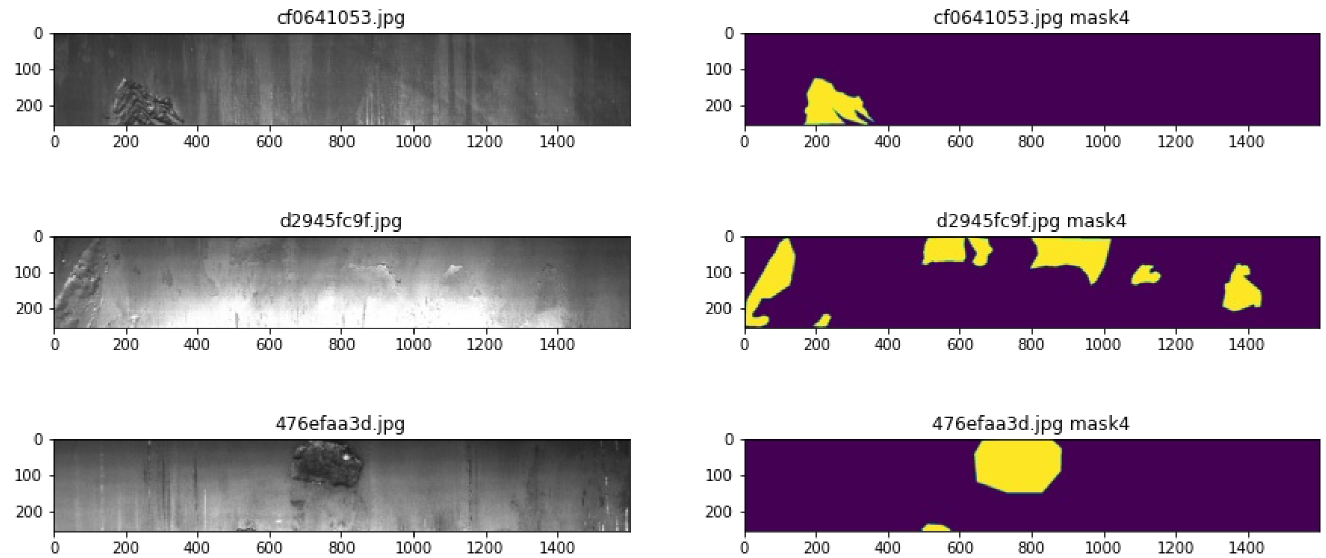

```
================================================================================
```

```
Sample images with Class 3 defect:
```




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

```
Sample images with Class 4 defect:
```




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

In [0]:

In [0]:

```
mask2rle(rle2mask(i[1]))
```

Out[0]:

```
'67466 2 67720 4 67975 7 68231 9 68486 11 68742 12 68997 13 69253 14 69509 14 69764 16 70020 19 701
47 3 70275 21 70403 9 70531 24 70659 15 70786 32 70914 21 71042 32 71170 27 71297 34 71426 30 71553
34 71682 32 71808 36 71938 34 72064 36 72194 36 72320 37 72450 38 72577 36 72706 39 72833 37 72962
39 73090 36 73218 40 73347 7 73359 22 73474 40 73615 22 73730 40 73871 21 73986 40 74127 22 74242 4
0 74383 24 74498 40 74639 25 74754 40 74895 26 75010 40 75151 28 75266 30 75297 9 75407 31 75522 29
75555 7 75663 33 75778 29 75813 5 75919 37 76034 28 76175 38 76290 28 76431 39 76546 28 76687 40 76
802 29 76943 41 77058 30 77199 42 77314 30 77454 44 77570 31 77710 45 77826 32 77966 46 78082 32 78
221 48 78338 33 78477 49 78594 33 78728 57 78850 34 78982 64 79106 34 79237 68 79362 33 79493 69 79
618 33 79748 71 79874 33 80003 73 80130 33 80258 75 80386 32 80514 76 80642 32 80769 78 80898 30 81
024 80 81154 28 81280 81 81410 27 81535 83 81666 26 81790 85 81922 24 82046 87 82178 23 82301 93 82
434 22 82556 98 82690 19 82811 100 82946 17 83067 101 83202 16 83322 102 83458 16 83577 104 83715 1
4 83832 105 83971 14 84087 107 84227 14 84342 108 84483 14 84597 110 84739 14 84853 110 84995 14 85
110 109 85251 14 85366 110 85507 14 85623 109 85763 14 85879 110 86019 15 86136 109 86275 15 86394
76 86472 29 86531 15 86652 75 86729 28 86787 15 86910 73 86986 27 87043 15 87167 71 87244 25 87299
15 87424 71 87501 24 87555 15 87680 71 87758 23 87811 15 87937 71 88016 21 88067 15 88193 71 88273
20 88323 15 88450 71 88531 17 88579 15 88706 71 88788 16 88835 15 88963 71 89048 11 89091 15 89219
71 89347 15 89475 71 89603 16 89732 70 89859 16 89988 71 90115 16 90245 70 90371 16 90501 70 90627
16 90758 70 90883 16 91014 70 91139 16 91270 70 91395 18 91527 70 91651 19 91783 70 91907 21 92040
69 92163 22 92296 70 92419 22 92553 69 92675 23 92809 68 92931 23 93066 66 93187 23 93322 66 93443
24 93578 65 93700 23 93834 65 93956 22 94089 65 94212 21 94345 64 94468 21 94601 62 94724 20 94857
61 94980 20 95112 61 95236 20 95368 59 95492 20 95624 57 95748 20 95880 54 96004 20 96136 52 96260
20 96392 52 96516 20 96648 52 96772 20 96904 52 97028 20 97160 52 97284 20 97416 53 97540 22 97672
53 97796 24 97928 53 98052 25 98184 54 98308 25 98440 54 98564 24 98696 54 98820 24 98953 53 99076
24 99209 52 99332 24 99466 51 99588 23 99722 50 99844 23 99978 25 100005 23 100100 22 100234 25 100
261 22 100356 22 100490 25 100517 21 100612 22 100746 25 100773 21 100868 21 101002 25 101029 20 1
01124 21 101259 24 101286 19 101380 21 101515 24 101542 18 101636 21 101771 24 101799 17 101892 21
102027 24 102055 17 102148 20 102284 23 102312 16 102404 20 102540 22 102568 15 102660 20 102798 2
0 102825 14 102916 20 103054 20 103081 13 103172 20 103311 19 103338 11 103429 20 103567 19 103594
9 103685 20 103825 15 103851 6 103941 20 104085 9 104197 1 104199 19 104456 17 104714 15 104971 13
105228 11 105486 8 105743 5'
```

In [0]:

```
i[1]
```

Out[0]:

```
'67466 2 67720 4 67975 7 68231 9 68486 11 68742 12 68997 13 69253 14 69509 14 69764 16 70020 19 701
47 3 70275 21 70403 9 70531 24 70659 15 70786 32 70914 21 71042 32 71170 27 71297 34 71426 30 71553
34 71682 32 71808 36 71938 34 72064 36 72194 36 72320 37 72450 38 72577 36 72706 39 72833 37 72962
39 73090 36 73218 40 73347 7 73359 22 73474 40 73615 22 73730 40 73871 21 73986 40 74127 22 74242 4
0 74383 24 74498 40 74639 25 74754 40 74895 26 75010 40 75151 28 75266 30 75297 9 75407 31 75522 29
75555 7 75663 33 75778 29 75813 5 75919 37 76034 28 76175 38 76290 28 76431 39 76546 28 76687 40 76
802 29 76943 41 77058 30 77199 42 77314 30 77454 44 77570 31 77710 45 77826 32 77966 46 78082 32 78
221 48 78338 33 78477 49 78594 33 78728 57 78850 34 78982 64 79106 34 79237 68 79362 33 79493 69 79
618 33 79748 71 79874 33 80003 73 80130 33 80258 75 80386 32 80514 76 80642 32 80769 78 80898 30 81
024 80 81154 28 81280 81 81410 27 81535 83 81666 26 81790 85 81922 24 82046 87 82178 23 82301 93 82
434 22 82556 98 82690 19 82811 100 82946 17 83067 101 83202 16 83322 102 83458 16 83577 104 83715 1
4 83832 105 83971 14 84087 107 84227 14 84342 108 84483 14 84597 110 84739 14 84853 110 84995 14 85
110 109 85251 14 85366 110 85507 14 85623 109 85763 14 85879 110 86019 15 86136 109 86275 15 86394
76 86472 29 86531 15 86652 75 86729 28 86787 15 86910 73 86986 27 87043 15 87167 71 87244 25 87299
15 87424 71 87501 24 87555 15 87680 71 87758 23 87811 15 87937 71 88016 21 88067 15 88193 71 88273
20 88323 15 88450 71 88531 17 88579 15 88706 71 88788 16 88835 15 88963 71 89048 11 89091 15 89219
71 89347 15 89475 71 89603 16 89732 70 89859 16 89988 71 90115 16 90245 70 90371 16 90501 70 90627
16 90758 70 90883 16 91014 70 91139 16 91270 70 91395 18 91527 70 91651 19 91783 70 91907 21 92040
69 92163 22 92296 70 92419 22 92553 69 92675 23 92809 68 92931 23 93066 66 93187 23 93322 66 93443
24 93578 65 93700 23 93834 65 93956 22 94089 65 94212 21 94345 64 94468 21 94601 62 94724 20 94857
61 94980 20 95112 61 95236 20 95368 59 95492 20 95624 57 95748 20 95880 54 96004 20 96136 52 96260
20 96392 52 96516 20 96648 52 96772 20 96904 52 97028 20 97160 52 97284 20 97416 53 97540 22 97672
53 97796 24 97928 53 98052 25 98184 54 98308 25 98440 54 98564 24 98696 54 98820 24 98953 53 99076
24 99209 52 99332 24 99466 51 99588 23 99722 50 99844 23 99978 25 100005 23 100100 22 100234 25 100
261 22 100356 22 100490 25 100517 21 100612 22 100746 25 100773 21 100868 21 101002 25 101029 20 1
01124 21 101259 24 101286 19 101380 21 101515 24 101542 18 101636 21 101771 24 101799 17 101892 21
102027 24 102055 17 102148 20 102284 23 102312 16 102404 20 102540 22 102568 15 102660 20 102798 2
0 102825 14 102916 20 103054 20 103081 13 103172 20 103311 19 103338 11 103429 20 103567 19 103594
9 103685 20 103825 15 103851 6 103941 20 104085 9 104197 1 104199 19 104456 17 104714 15 104971 13
105228 11 105486 8 105743 5'
```

As can be seen above, the original values of EncodedPixels match exactly with RLE pixels encoded using mask2rle()

function over the mask created using rle2mask(). Therefore, the two functions are working as desired.

- **Note: The above illustration is done using masks of size 256 x 1600 whereas the code in rle2mask() has now been changed so as to get masks of size 128 x 800, as we will use images of halved size for training.**

4.5. Defining metric and loss function

In [0]:

```
from keras import backend as K
from keras.losses import binary_crossentropy

def dice_coef(y_true, y_pred, smooth=1):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) + smooth)

def bce_dice_loss(y_true, y_predict):
    return binary_crossentropy(y_true, y_predict) + (1-dice_coef(y_true, y_predict))

def  dice_loss(y_true, y_predict):
    return (1-dice_coef(y_true, y_predict))
```

5. Model

5.1. U-net architecture

In [0]:

```
def conv2d_block(input_tensor, n_filters, kernel_size = 3, batchnorm = True):
    '''returns a block of two 3x3 convolutions, each  followed by a rectified linear unit
(ReLU)'''
    # first layer
    x = Conv2D(filters = n_filters, kernel_size = (kernel_size, kernel_size),\
              kernel_initializer = 'he_normal', padding = 'same')(input_tensor)
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)

    # second layer
    x = Conv2D(filters = n_filters, kernel_size = (kernel_size, kernel_size),\
              kernel_initializer = 'he_normal', padding = 'same')(x)
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)

    return x
```

In [0]:

```
def get_unet(input_img, n_filters, dropout, batchnorm):
    """Function to define the UNET Model"""
    # Contracting Path
    c1 = conv2d_block(input_img, n_filters * 1, kernel_size = 3, batchnorm = batchnorm)
    p1 = MaxPooling2D((2, 2))(c1)
    p1 = Dropout(dropout)(p1)

    c2 = conv2d_block(p1, n_filters * 2, kernel_size = 3, batchnorm = batchnorm)
    p2 = MaxPooling2D((2, 2))(c2)
    p2 = Dropout(dropout)(p2)

    c3 = conv2d_block(p2, n_filters * 4, kernel_size = 3, batchnorm = batchnorm)
    p3 = MaxPooling2D((2, 2))(c3)
    p3 = Dropout(dropout)(p3)

    c4 = conv2d_block(p3, n_filters * 8, kernel_size = 3, batchnorm = batchnorm)
    p4 = MaxPooling2D((2, 2))(c4)
    p4 = Dropout(dropout)(p4)
```

```
    c5 = conv2d_block(p4, n_filters = n_filters * 16, kernel_size = 3, batchnorm = batchnorm)

    # Expansive Path
    u6 = UpSampling2D()(c5)
    u6 = Conv2D(filters = n_filters *8, kernel_size = (2, 2), kernel_initializer = 'he_normal',
padding = 'same')(u6)
    u6 = concatenate([u6, c4])
    u6 = Dropout(dropout)(u6)
    c6 = conv2d_block(u6, n_filters * 8, kernel_size = 3, batchnorm = batchnorm)

    u7 = UpSampling2D()(c6)
    u7 = Conv2D(filters = n_filters *4, kernel_size = (2, 2), kernel_initializer = 'he_normal',
padding = 'same')(u7)
    u7 = concatenate([u7, c3])
    u7 = Dropout(dropout)(u7)
    c7 = conv2d_block(u7, n_filters * 4, kernel_size = 3, batchnorm = batchnorm)

    u8 = UpSampling2D()(c7)
    u8 = Conv2D(filters = n_filters *2, kernel_size = (2, 2), kernel_initializer = 'he_normal',
padding = 'same')(u8)
    u8 = concatenate([u8, c2])
    u8 = Dropout(dropout)(u8)
    c8 = conv2d_block(u8, n_filters * 2, kernel_size = 3, batchnorm = batchnorm)

    u9 = UpSampling2D()(c8)
    u9 = Conv2D(filters = n_filters *1, kernel_size = (2, 2), kernel_initializer = 'he_normal',
padding = 'same')(u9)
    u9 = concatenate([u9, c1])
    u9 = Dropout(dropout)(u9)
    c9 = conv2d_block(u9, n_filters * 1, kernel_size = 3, batchnorm = batchnorm)

    outputs = Conv2D(4, (1, 1), activation='sigmoid')(c9)
    model = Model(inputs=[input_img], outputs=[outputs])
    return model
```

In [31]:

```
input_img = Input((128, 800, 3), name='img')
model = get_unet(input_img, n_filters=8, dropout=0.2, batchnorm=True)
model.compile(optimizer=Adam(), loss=bce_dice_loss, metrics=[dice_coef])
model.summary()
```

```
Model: "model_2"
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
img (InputLayer)                 (None, 128, 800, 3)   0
____________________________________________________________________________________________________
conv2d_24 (Conv2D)               (None, 128, 800, 8)   224         img[0][0]
____________________________________________________________________________________________________
batch_normalization_19 (BatchNo  (None, 128, 800, 8)   32          conv2d_24[0][0]
____________________________________________________________________________________________________
activation_19 (Activation)       (None, 128, 800, 8)   0           batch_normalization_19[0][0]
____________________________________________________________________________________________________
conv2d_25 (Conv2D)               (None, 128, 800, 8)   584         activation_19[0][0]
____________________________________________________________________________________________________
batch_normalization_20 (BatchNo  (None, 128, 800, 8)   32          conv2d_25[0][0]
____________________________________________________________________________________________________
activation_20 (Activation)       (None, 128, 800, 8)   0           batch_normalization_20[0][0]
____________________________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)   (None, 64, 400, 8)    0           activation_20[0][0]
____________________________________________________________________________________________________
dropout_9 (Dropout)              (None, 64, 400, 8)    0           max_pooling2d_5[0][0]
____________________________________________________________________________________________________
conv2d_26 (Conv2D)               (None, 64, 400, 16)   1168        dropout_9[0][0]
____________________________________________________________________________________________________
batch_normalization_21 (BatchNo  (None, 64, 400, 16)   64          conv2d_26[0][0]
____________________________________________________________________________________________________
activation_21 (Activation)       (None, 64, 400, 16)   0           batch_normalization_21[0][0]
____________________________________________________________________________________________________
conv2d_27 (Conv2D)               (None, 64, 400, 16)   2320        activation_21[0][0]
____________________________________________________________________________________________________
batch_normalization_22 (BatchNo  (None, 64, 400, 16)   64          conv2d_27[0][0]
____________________________________________________________________________________________________
activation_22 (Activation)       (None, 64, 400, 16)   0           batch_normalization_22[0][0]
```

max_pooling2d_6 (MaxPooling2D)	(None, 32, 200, 16)	0	activation_22[0][0]
dropout_10 (Dropout)	(None, 32, 200, 16)	0	max_pooling2d_6[0][0]
conv2d_28 (Conv2D)	(None, 32, 200, 32)	4640	dropout_10[0][0]
batch_normalization_23 (BatchNo	(None, 32, 200, 32)	128	conv2d_28[0][0]
activation_23 (Activation)	(None, 32, 200, 32)	0	batch_normalization_23[0][0]
conv2d_29 (Conv2D)	(None, 32, 200, 32)	9248	activation_23[0][0]
batch_normalization_24 (BatchNo	(None, 32, 200, 32)	128	conv2d_29[0][0]
activation_24 (Activation)	(None, 32, 200, 32)	0	batch_normalization_24[0][0]
max_pooling2d_7 (MaxPooling2D)	(None, 16, 100, 32)	0	activation_24[0][0]
dropout_11 (Dropout)	(None, 16, 100, 32)	0	max_pooling2d_7[0][0]
conv2d_30 (Conv2D)	(None, 16, 100, 64)	18496	dropout_11[0][0]
batch_normalization_25 (BatchNo	(None, 16, 100, 64)	256	conv2d_30[0][0]
activation_25 (Activation)	(None, 16, 100, 64)	0	batch_normalization_25[0][0]
conv2d_31 (Conv2D)	(None, 16, 100, 64)	36928	activation_25[0][0]
batch_normalization_26 (BatchNo	(None, 16, 100, 64)	256	conv2d_31[0][0]
activation_26 (Activation)	(None, 16, 100, 64)	0	batch_normalization_26[0][0]
max_pooling2d_8 (MaxPooling2D)	(None, 8, 50, 64)	0	activation_26[0][0]
dropout_12 (Dropout)	(None, 8, 50, 64)	0	max_pooling2d_8[0][0]
conv2d_32 (Conv2D)	(None, 8, 50, 128)	73856	dropout_12[0][0]
batch_normalization_27 (BatchNo	(None, 8, 50, 128)	512	conv2d_32[0][0]
activation_27 (Activation)	(None, 8, 50, 128)	0	batch_normalization_27[0][0]
conv2d_33 (Conv2D)	(None, 8, 50, 128)	147584	activation_27[0][0]
batch_normalization_28 (BatchNo	(None, 8, 50, 128)	512	conv2d_33[0][0]
activation_28 (Activation)	(None, 8, 50, 128)	0	batch_normalization_28[0][0]
up_sampling2d_5 (UpSampling2D)	(None, 16, 100, 128)	0	activation_28[0][0]
conv2d_34 (Conv2D)	(None, 16, 100, 64)	32832	up_sampling2d_5[0][0]
concatenate_5 (Concatenate)	(None, 16, 100, 128)	0	conv2d_34[0][0] activation_26[0][0]
dropout_13 (Dropout)	(None, 16, 100, 128)	0	concatenate_5[0][0]
conv2d_35 (Conv2D)	(None, 16, 100, 64)	73792	dropout_13[0][0]
batch_normalization_29 (BatchNo	(None, 16, 100, 64)	256	conv2d_35[0][0]
activation_29 (Activation)	(None, 16, 100, 64)	0	batch_normalization_29[0][0]
conv2d_36 (Conv2D)	(None, 16, 100, 64)	36928	activation_29[0][0]
batch_normalization_30 (BatchNo	(None, 16, 100, 64)	256	conv2d_36[0][0]
activation_30 (Activation)	(None, 16, 100, 64)	0	batch_normalization_30[0][0]
up_sampling2d_6 (UpSampling2D)	(None, 32, 200, 64)	0	activation_30[0][0]
conv2d_37 (Conv2D)	(None, 32, 200, 32)	8224	up_sampling2d_6[0][0]
concatenate_6 (Concatenate)	(None, 32, 200, 64)	0	conv2d_37[0][0] activation_24[0][0]

```
dropout_14 (Dropout)            (None, 32, 200, 64)  0           concatenate_6[0][0]
__________________________________________________________________________________________________
conv2d_38 (Conv2D)              (None, 32, 200, 32)  18464       dropout_14[0][0]
__________________________________________________________________________________________________
batch_normalization_31 (BatchNo (None, 32, 200, 32)  128         conv2d_38[0][0]
__________________________________________________________________________________________________
activation_31 (Activation)      (None, 32, 200, 32)  0           batch_normalization_31[0][0]
__________________________________________________________________________________________________
conv2d_39 (Conv2D)              (None, 32, 200, 32)  9248        activation_31[0][0]
__________________________________________________________________________________________________
batch_normalization_32 (BatchNo (None, 32, 200, 32)  128         conv2d_39[0][0]
__________________________________________________________________________________________________
activation_32 (Activation)      (None, 32, 200, 32)  0           batch_normalization_32[0][0]
__________________________________________________________________________________________________
up_sampling2d_7 (UpSampling2D)  (None, 64, 400, 32)  0           activation_32[0][0]
__________________________________________________________________________________________________
conv2d_40 (Conv2D)              (None, 64, 400, 16)  2064        up_sampling2d_7[0][0]
__________________________________________________________________________________________________
concatenate_7 (Concatenate)     (None, 64, 400, 32)  0           conv2d_40[0][0]
                                                                 activation_22[0][0]
__________________________________________________________________________________________________
dropout_15 (Dropout)            (None, 64, 400, 32)  0           concatenate_7[0][0]
__________________________________________________________________________________________________
conv2d_41 (Conv2D)              (None, 64, 400, 16)  4624        dropout_15[0][0]
__________________________________________________________________________________________________
batch_normalization_33 (BatchNo (None, 64, 400, 16)  64          conv2d_41[0][0]
__________________________________________________________________________________________________
activation_33 (Activation)      (None, 64, 400, 16)  0           batch_normalization_33[0][0]
__________________________________________________________________________________________________
conv2d_42 (Conv2D)              (None, 64, 400, 16)  2320        activation_33[0][0]
__________________________________________________________________________________________________
batch_normalization_34 (BatchNo (None, 64, 400, 16)  64          conv2d_42[0][0]
__________________________________________________________________________________________________
activation_34 (Activation)      (None, 64, 400, 16)  0           batch_normalization_34[0][0]
__________________________________________________________________________________________________
up_sampling2d_8 (UpSampling2D)  (None, 128, 800, 16) 0           activation_34[0][0]
__________________________________________________________________________________________________
conv2d_43 (Conv2D)              (None, 128, 800, 8)  520         up_sampling2d_8[0][0]
__________________________________________________________________________________________________
concatenate_8 (Concatenate)     (None, 128, 800, 16) 0           conv2d_43[0][0]
                                                                 activation_20[0][0]
__________________________________________________________________________________________________
dropout_16 (Dropout)            (None, 128, 800, 16) 0           concatenate_8[0][0]
__________________________________________________________________________________________________
conv2d_44 (Conv2D)              (None, 128, 800, 8)  1160        dropout_16[0][0]
__________________________________________________________________________________________________
batch_normalization_35 (BatchNo (None, 128, 800, 8)  32          conv2d_44[0][0]
__________________________________________________________________________________________________
activation_35 (Activation)      (None, 128, 800, 8)  0           batch_normalization_35[0][0]
__________________________________________________________________________________________________
conv2d_45 (Conv2D)              (None, 128, 800, 8)  584         activation_35[0][0]
__________________________________________________________________________________________________
batch_normalization_36 (BatchNo (None, 128, 800, 8)  32          conv2d_45[0][0]
__________________________________________________________________________________________________
activation_36 (Activation)      (None, 128, 800, 8)  0           batch_normalization_36[0][0]
__________________________________________________________________________________________________
conv2d_46 (Conv2D)              (None, 128, 800, 4)  36          activation_36[0][0]
==================================================================================================
Total params: 488,788
Trainable params: 487,316
Non-trainable params: 1,472
__________________________________________________________________________________________________
```

5.2. Checkpointing the model and creating callback list

In [34]:

```python
from keras.callbacks import ModelCheckpoint
from tensorflow.python.keras.callbacks import TensorBoard
from keras.callbacks import TensorBoard
import tensorflow as tf
import keras
from tensorboardcolab import *
```

```
tbc=TensorBoardColab()

filepath="/content/drive/My Drive/Colab Notebooks/Steel Defect Detection/model2.h5"
checkpoints = ModelCheckpoint(filepath, monitor='val_dice_coef', verbose=1, save_best_only=True, mo
de='max')
callbacks_list = [checkpoints, TensorBoardColabCallback(tbc)]
```

```
Wait for 8 seconds...
TensorBoard link:
https://49200814.ngrok.io
```

5.3. Training

In [35]:

```
train_batches = Train_DataGenerator(train_data,shuffle=True)
valid_batches = Val_DataGenerator(cv_data)
history = model.fit_generator(train_batches, validation_data = valid_batches, epochs = 50, verbose=
1,
                              callbacks = callbacks_list)
```

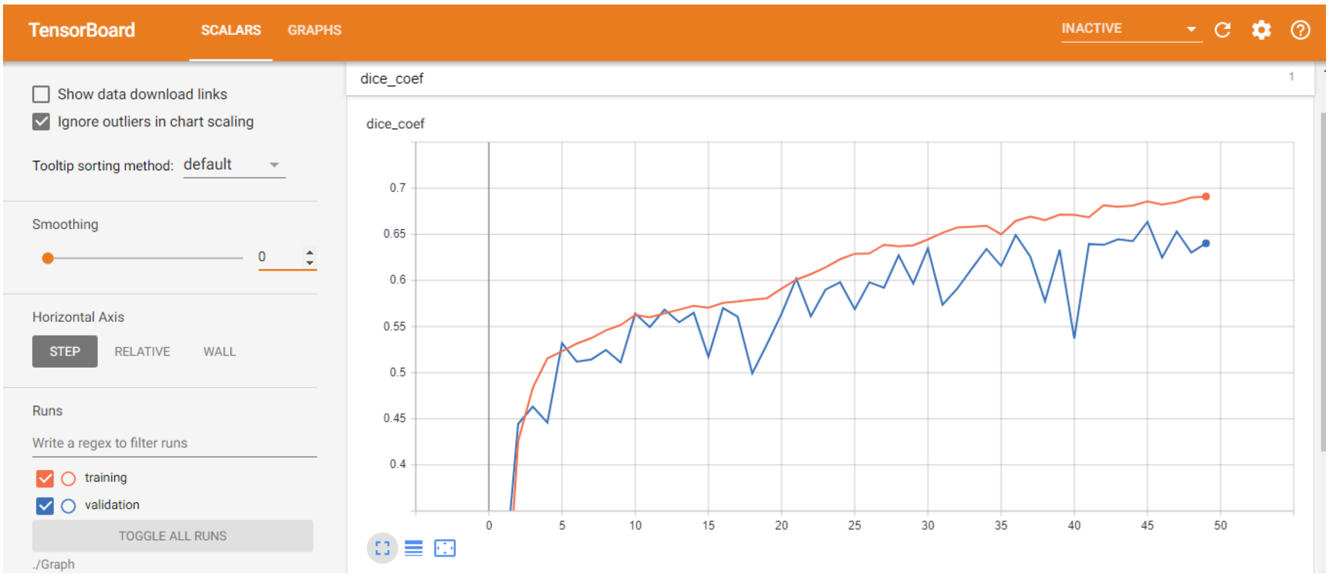
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:1033: The name tf.assign_add is deprecated. Please us
e tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:1020: The name tf.assign is deprecated. Please use tf
.compat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorboardcolab/core.py:49: The na
me tf.summary.FileWriter is deprecated. Please use tf.compat.v1.summary.FileWriter instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/callbacks.py:1122: The name t
f.summary.merge_all is deprecated. Please use tf.compat.v1.summary.merge_all instead.

Epoch 1/50
333/333 [==============================] - 271s 814ms/step - loss: 1.2618 - dice_coef: 0.0370 - va
l_loss: 1.0338 - val_dice_coef: 0.0957

Epoch 00001: val_dice_coef improved from -inf to 0.09568, saving model to /content/drive/My
Drive/Colab Notebooks/Steel Defect Detection/model2.h5
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorboardcolab/callbacks.py:51: T
he name tf.Summary is deprecated. Please use tf.compat.v1.Summary instead.

Epoch 2/50
333/333 [==============================] - 258s 776ms/step - loss: 0.8803 - dice_coef: 0.1842 - va
l_loss: 0.8293 - val_dice_coef: 0.2641

Epoch 00002: val_dice_coef improved from 0.09568 to 0.26414, saving model to /content/drive/My Dri
ve/Colab Notebooks/Steel Defect Detection/model2.h5
Epoch 3/50
333/333 [==============================] - 258s 775ms/step - loss: 0.6117 - dice_coef: 0.4257 - va
l_loss: 0.6062 - val_dice_coef: 0.4444

Epoch 00003: val_dice_coef improved from 0.26414 to 0.44442, saving model to /content/drive/My Dri
ve/Colab Notebooks/Steel Defect Detection/model2.h5
Epoch 4/50
333/333 [==============================] - 260s 780ms/step - loss: 0.5556 - dice_coef: 0.4836 - va
l_loss: 0.5848 - val_dice_coef: 0.4632

Epoch 00004: val_dice_coef improved from 0.44442 to 0.46321, saving model to /content/drive/My Dri
ve/Colab Notebooks/Steel Defect Detection/model2.h5
Epoch 5/50
333/333 [==============================] - 264s 794ms/step - loss: 0.5221 - dice_coef: 0.5156 - va
l_loss: 0.6246 - val_dice_coef: 0.4458

Epoch 00005: val_dice_coef did not improve from 0.46321
Epoch 6/50
333/333 [==============================] - 267s 803ms/step - loss: 0.5142 - dice_coef: 0.5231 - va
l_loss: 0.5110 - val_dice_coef: 0.5320

Epoch 00006: val_dice_coef improved from 0.46321 to 0.53201, saving model to /content/drive/My Dri
ve/Colab Notebooks/Steel Defect Detection/model2.h5
```

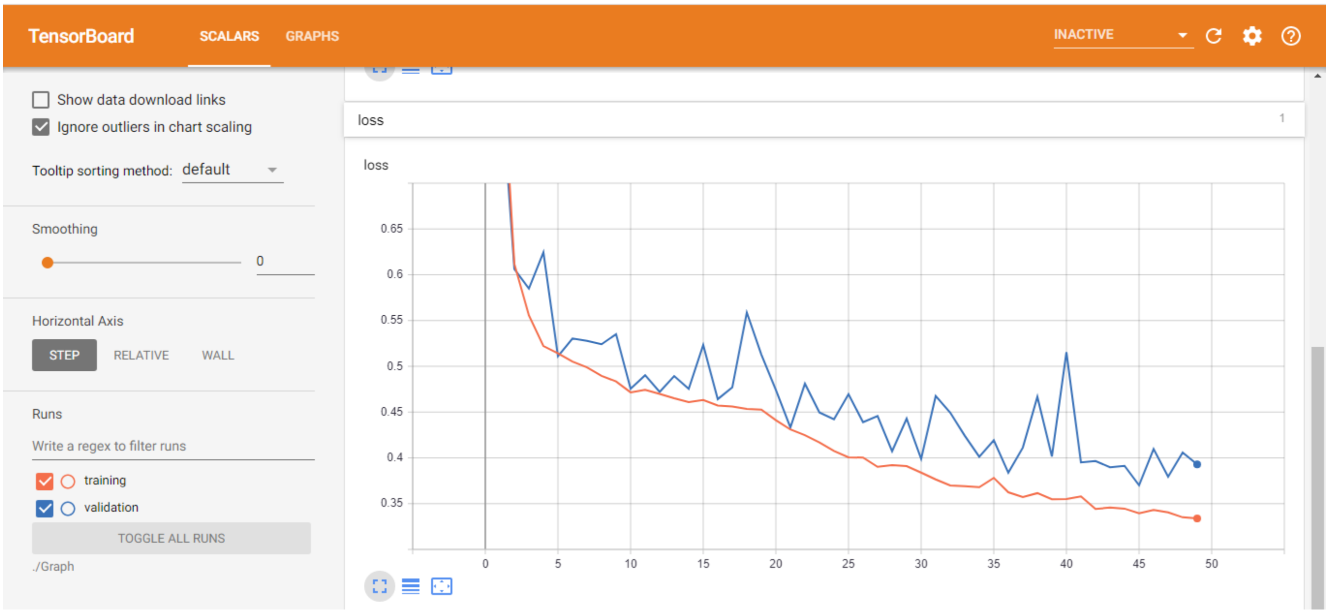
```
Epoch 7/50
333/333 [==============================] - 267s 802ms/step - loss: 0.5052 - dice_coef: 0.5315 - va
l_loss: 0.5304 - val_dice_coef: 0.5119

Epoch 00007: val_dice_coef did not improve from 0.53201
Epoch 8/50
333/333 [==============================] - 263s 789ms/step - loss: 0.4987 - dice_coef: 0.5375 - va
l_loss: 0.5278 - val_dice_coef: 0.5144

Epoch 00008: val_dice_coef did not improve from 0.53201
Epoch 9/50
333/333 [==============================] - 259s 779ms/step - loss: 0.4896 - dice_coef: 0.5459 - va
l_loss: 0.5240 - val_dice_coef: 0.5247

Epoch 00009: val_dice_coef did not improve from 0.53201
Epoch 10/50
333/333 [==============================] - 261s 785ms/step - loss: 0.4835 - dice_coef: 0.5516 - va
l_loss: 0.5351 - val_dice_coef: 0.5112

Epoch 00010: val_dice_coef did not improve from 0.53201
Epoch 11/50
333/333 [==============================] - 265s 794ms/step - loss: 0.4716 - dice_coef: 0.5625 - va
l_loss: 0.4755 - val_dice_coef: 0.5639

Epoch 00011: val_dice_coef improved from 0.53201 to 0.56388, saving model to /content/drive/My Dri
ve/Colab Notebooks/Steel Defect Detection/model2.h5
Epoch 12/50
333/333 [==============================] - 258s 774ms/step - loss: 0.4743 - dice_coef: 0.5601 - va
l_loss: 0.4903 - val_dice_coef: 0.5495

Epoch 00012: val_dice_coef did not improve from 0.56388
Epoch 13/50
333/333 [==============================] - 258s 774ms/step - loss: 0.4699 - dice_coef: 0.5642 - va
l_loss: 0.4719 - val_dice_coef: 0.5684

Epoch 00013: val_dice_coef improved from 0.56388 to 0.56841, saving model to /content/drive/My Dri
ve/Colab Notebooks/Steel Defect Detection/model2.h5
Epoch 14/50
333/333 [==============================] - 259s 777ms/step - loss: 0.4652 - dice_coef: 0.5684 - va
l_loss: 0.4893 - val_dice_coef: 0.5547

Epoch 00014: val_dice_coef did not improve from 0.56841
Epoch 15/50
333/333 [==============================] - 260s 779ms/step - loss: 0.4608 - dice_coef: 0.5725 - va
l_loss: 0.4753 - val_dice_coef: 0.5650

Epoch 00015: val_dice_coef did not improve from 0.56841
Epoch 16/50
333/333 [==============================] - 259s 777ms/step - loss: 0.4632 - dice_coef: 0.5705 - va
l_loss: 0.5233 - val_dice_coef: 0.5174

Epoch 00016: val_dice_coef did not improve from 0.56841
Epoch 17/50
333/333 [==============================] - 259s 778ms/step - loss: 0.4572 - dice_coef: 0.5756 - va
l_loss: 0.4641 - val_dice_coef: 0.5701

Epoch 00017: val_dice_coef improved from 0.56841 to 0.57008, saving model to /content/drive/My Dri
ve/Colab Notebooks/Steel Defect Detection/model2.h5
Epoch 18/50
333/333 [==============================] - 265s 795ms/step - loss: 0.4560 - dice_coef: 0.5772 - va
l_loss: 0.4769 - val_dice_coef: 0.5607

Epoch 00018: val_dice_coef did not improve from 0.57008
Epoch 19/50
333/333 [==============================] - 258s 775ms/step - loss: 0.4534 - dice_coef: 0.5792 - va
l_loss: 0.5587 - val_dice_coef: 0.4992

Epoch 00019: val_dice_coef did not improve from 0.57008
Epoch 20/50
333/333 [==============================] - 258s 776ms/step - loss: 0.4527 - dice_coef: 0.5808 - va
l_loss: 0.5129 - val_dice_coef: 0.5306

Epoch 00020: val_dice_coef did not improve from 0.57008
Epoch 21/50
333/333 [==============================] - 257s 772ms/step - loss: 0.4411 - dice_coef: 0.5911 - va
l_loss: 0.4743 - val_dice_coef: 0.5638
```

```
Epoch 00021: val_dice_coef did not improve from 0.57008
Epoch 22/50
333/333 [==============================] - 257s 773ms/step - loss: 0.4310 - dice_coef: 0.6008 - va
l_loss: 0.4332 - val_dice_coef: 0.6019

Epoch 00022: val_dice_coef improved from 0.57008 to 0.60192, saving model to /content/drive/My Dri
ve/Colab Notebooks/Steel Defect Detection/model2.h5
Epoch 23/50
333/333 [==============================] - 256s 769ms/step - loss: 0.4247 - dice_coef: 0.6069 - va
l_loss: 0.4810 - val_dice_coef: 0.5612

Epoch 00023: val_dice_coef did not improve from 0.60192
Epoch 24/50
333/333 [==============================] - 256s 770ms/step - loss: 0.4167 - dice_coef: 0.6142 - va
l_loss: 0.4496 - val_dice_coef: 0.5900

Epoch 00024: val_dice_coef did not improve from 0.60192
Epoch 25/50
333/333 [==============================] - 257s 772ms/step - loss: 0.4075 - dice_coef: 0.6230 - va
l_loss: 0.4421 - val_dice_coef: 0.5979

Epoch 00025: val_dice_coef did not improve from 0.60192
Epoch 26/50
333/333 [==============================] - 256s 769ms/step - loss: 0.4006 - dice_coef: 0.6289 - va
l_loss: 0.4695 - val_dice_coef: 0.5688

Epoch 00026: val_dice_coef did not improve from 0.60192
Epoch 27/50
333/333 [==============================] - 258s 774ms/step - loss: 0.4003 - dice_coef: 0.6293 - va
l_loss: 0.4390 - val_dice_coef: 0.5980

Epoch 00027: val_dice_coef did not improve from 0.60192
Epoch 28/50
333/333 [==============================] - 257s 772ms/step - loss: 0.3902 - dice_coef: 0.6386 - va
l_loss: 0.4457 - val_dice_coef: 0.5921

Epoch 00028: val_dice_coef did not improve from 0.60192
Epoch 29/50
333/333 [==============================] - 261s 783ms/step - loss: 0.3921 - dice_coef: 0.6370 - va
l_loss: 0.4072 - val_dice_coef: 0.6272

Epoch 00029: val_dice_coef improved from 0.60192 to 0.62720, saving model to /content/drive/My Dri
ve/Colab Notebooks/Steel Defect Detection/model2.h5
Epoch 30/50
333/333 [==============================] - 264s 793ms/step - loss: 0.3910 - dice_coef: 0.6382 - va
l_loss: 0.4428 - val_dice_coef: 0.5966

Epoch 00030: val_dice_coef did not improve from 0.62720
Epoch 31/50
333/333 [==============================] - 260s 780ms/step - loss: 0.3839 - dice_coef: 0.6444 - va
l_loss: 0.3989 - val_dice_coef: 0.6348

Epoch 00031: val_dice_coef improved from 0.62720 to 0.63476, saving model to /content/drive/My Dri
ve/Colab Notebooks/Steel Defect Detection/model2.h5
Epoch 32/50
333/333 [==============================] - 262s 786ms/step - loss: 0.3764 - dice_coef: 0.6517 - va
l_loss: 0.4676 - val_dice_coef: 0.5735

Epoch 00032: val_dice_coef did not improve from 0.63476
Epoch 33/50
333/333 [==============================] - 262s 786ms/step - loss: 0.3699 - dice_coef: 0.6573 - va
l_loss: 0.4494 - val_dice_coef: 0.5910

Epoch 00033: val_dice_coef did not improve from 0.63476
Epoch 34/50
333/333 [==============================] - 260s 782ms/step - loss: 0.3692 - dice_coef: 0.6582 - va
l_loss: 0.4242 - val_dice_coef: 0.6129

Epoch 00034: val_dice_coef did not improve from 0.63476
Epoch 35/50
333/333 [==============================] - 260s 781ms/step - loss: 0.3680 - dice_coef: 0.6593 - va
l_loss: 0.4013 - val_dice_coef: 0.6342

Epoch 00035: val_dice_coef did not improve from 0.63476
Epoch 36/50
333/333 [==============================] - 260s 780ms/step - loss: 0.3781 - dice_coef: 0.6500 - va
l_loss: 0.4192 - val_dice_coef: 0.6158
```

```
Epoch 00036: val_dice_coef did not improve from 0.63476
Epoch 37/50
333/333 [==============================] - 260s 782ms/step - loss: 0.3624 - dice_coef: 0.6646 - va
l_loss: 0.3838 - val_dice_coef: 0.6492

Epoch 00037: val_dice_coef improved from 0.63476 to 0.64923, saving model to /content/drive/My Dri
ve/Colab Notebooks/Steel Defect Detection/model2.h5
Epoch 38/50
333/333 [==============================] - 261s 785ms/step - loss: 0.3573 - dice_coef: 0.6694 - va
l_loss: 0.4112 - val_dice_coef: 0.6257

Epoch 00038: val_dice_coef did not improve from 0.64923
Epoch 39/50
333/333 [==============================] - 257s 772ms/step - loss: 0.3615 - dice_coef: 0.6654 - va
l_loss: 0.4667 - val_dice_coef: 0.5774

Epoch 00039: val_dice_coef did not improve from 0.64923
Epoch 40/50
333/333 [==============================] - 258s 774ms/step - loss: 0.3548 - dice_coef: 0.6714 - va
l_loss: 0.4015 - val_dice_coef: 0.6334

Epoch 00040: val_dice_coef did not improve from 0.64923
Epoch 41/50
333/333 [==============================] - 257s 773ms/step - loss: 0.3551 - dice_coef: 0.6712 - va
l_loss: 0.5154 - val_dice_coef: 0.5370

Epoch 00041: val_dice_coef did not improve from 0.64923
Epoch 42/50
333/333 [==============================] - 257s 772ms/step - loss: 0.3579 - dice_coef: 0.6686 - va
l_loss: 0.3950 - val_dice_coef: 0.6396

Epoch 00042: val_dice_coef did not improve from 0.64923
Epoch 43/50
333/333 [==============================] - 258s 774ms/step - loss: 0.3441 - dice_coef: 0.6814 - va
l_loss: 0.3964 - val_dice_coef: 0.6385

Epoch 00043: val_dice_coef did not improve from 0.64923
Epoch 44/50
333/333 [==============================] - 257s 771ms/step - loss: 0.3457 - dice_coef: 0.6800 - va
l_loss: 0.3898 - val_dice_coef: 0.6448

Epoch 00044: val_dice_coef did not improve from 0.64923
Epoch 45/50
333/333 [==============================] - 258s 775ms/step - loss: 0.3445 - dice_coef: 0.6812 - va
l_loss: 0.3913 - val_dice_coef: 0.6425

Epoch 00045: val_dice_coef did not improve from 0.64923
Epoch 46/50
333/333 [==============================] - 259s 777ms/step - loss: 0.3395 - dice_coef: 0.6856 - va
l_loss: 0.3701 - val_dice_coef: 0.6636

Epoch 00046: val_dice_coef improved from 0.64923 to 0.66362, saving model to /content/drive/My Dri
ve/Colab Notebooks/Steel Defect Detection/model2.h5
Epoch 47/50
333/333 [==============================] - 258s 776ms/step - loss: 0.3431 - dice_coef: 0.6823 - va
l_loss: 0.4097 - val_dice_coef: 0.6249

Epoch 00047: val_dice_coef did not improve from 0.66362
Epoch 48/50
333/333 [==============================] - 257s 771ms/step - loss: 0.3405 - dice_coef: 0.6849 - va
l_loss: 0.3794 - val_dice_coef: 0.6531

Epoch 00048: val_dice_coef did not improve from 0.66362
Epoch 49/50
333/333 [==============================] - 257s 772ms/step - loss: 0.3351 - dice_coef: 0.6900 - va
l_loss: 0.4057 - val_dice_coef: 0.6303

Epoch 00049: val_dice_coef did not improve from 0.66362
Epoch 50/50
333/333 [==============================] - 257s 771ms/step - loss: 0.3338 - dice_coef: 0.6911 - va
l_loss: 0.3929 - val_dice_coef: 0.6403

Epoch 00050: val_dice_coef did not improve from 0.66362
```

5.4. Tensorboard plots of training & validation results





5.5. Loading the saved model and testing

In [0]:

```
from keras.models import load_model

dependencies = {'bce_dice_loss': bce_dice_loss, 'dice_coef': dice_coef}

filepath= "/content/drive/My Drive/Colab Notebooks/Steel Defect Detection/"
model = load_model(filepath+'model2.h5', custom_objects=dependencies)
```

In [40]:

```
evals= model.evaluate(valid_batches,verbose=1)
```

```
58/58 [==============================] - 30s 516ms/step
```

In [41]:

```
print('Validation score:')
print('loss:',evals[0])
print('dice_coeff:',evals[1])
```

```
Validation score:
loss: 0.37009855591017626
dice_coeff: 0.6636190219172116
```

Utility function to visualize ground truth and predicted mask of an image(train/cv)

In [0]:

```
def visualize_prediction(f: str):

    data_path = 'train_images/'
    X = np.empty((1,128,800,3),dtype='uint8')
    img = cv2.imread(data_path + f)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (800,128))
    X[0,] = img
    mask = model.predict(X)

    df1 = data[data.ImageId == f].reset_index()

    if all((df1['Defect_1'].all()=='',df1['Defect_2'].all()=='',df1['Defect_3'].all()=='',df1['Defe
ct_4'].all()=='')):

        fig, (ax1,ax2, ax3) = plt.subplots(nrows = 1,ncols = 3,figsize=(18, 7))
        ax1.imshow(img)
        ax1.set_title(f)

        ax2.imshow(rle2mask(''))
        ax2.set_title('ground truth mask 0')

        ax3.imshow(mask[0,:,:,0].round().astype('int'))
        ax3.set_title('predicted mask 0')
        plt.show()
        print('-'*120,'\n')

    else:
        for k in [1,2,3,4]:
            for i in range(len(df1)):
                if df1[f'Defect_{k}'][i] != '':
                    encoded_pix = df1[f'Defect_{k}'][i]

                    fig, (ax1,ax2, ax3) = plt.subplots(nrows = 1,ncols = 3,figsize=(18, 7))
                    ax1.imshow(img)
                    ax1.set_title(f)

                    ax2.imshow(rle2mask(encoded_pix))
                    ax2.set_title('ground truth mask '+str(k))

                    ax3.imshow(mask[0,:,:,k-1].round().astype('int'))
                    ax3.set_title('predicted mask '+str(k))
                    plt.show()
                    print('-'*120,'\n')
```

In [58]:

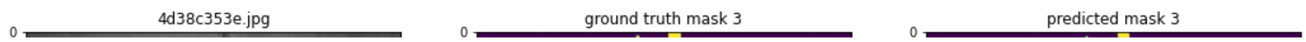
```
cv_data[cv_data.ImageId=='4d38c353e.jpg']
```

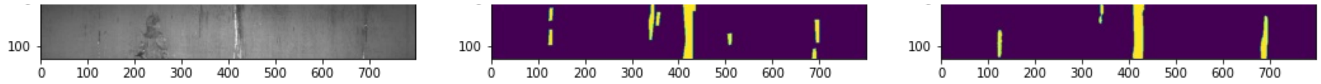
Out[58]:

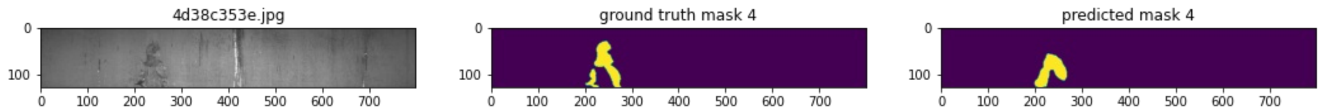
	ImageId	Defect_1	Defect_2	Defect_3	Defect_4
3800	4d38c353e.jpg			62142 9 62382 25 62550 10 62621 42 62788 28 62...	102638 5 102894 9 103150 9 103406 10 103662 10...

In [60]:

```
visualize_predictions('4d38c353e.jpg')
```







5.6. Predicting on raw test images

5.6.1. Predict on half size images(128x800)

In [55]:

```
# Predicting on test data

data_path = 'test_images/'
files = list(os.listdir(data_path))
img_ID= []
classId = []
rle_lst = []
img_classId= []
for f in tqdm(files):
    X = np.empty((1,128,800,3),dtype=np.uint8)
    img = cv2.imread(data_path + f)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (800,128))
    X[0,] = img
    mask = model.predict(X)
    rle_m = np.empty((128,800),dtype=np.uint8)
    for i in range(4):
        rle_m = mask[0,:,:,i].round().astype(int)
        rle = mask2rle(rle_m)
        rle_lst.append(rle)
        img_ID.append(f)
        classId.append(str(i+1))
        img_classId.append(f+'_'+str(i+1))
```

Creating csv file for Kaggle submission

In [0]:

```
output = {'ImageId_ClassId':img_classId, 'EncodedPixels' : rle_lst}

filepath= "/content/drive/My Drive/Colab Notebooks/Steel Defect Detection/"
output_df = pd.DataFrame(output)
output_df.to_csv(filepath+'submission_halfsize2.csv', index=False)
```

5.6.2. Predict on full size images(256x1600)

Modifying the model architecture for new input size 256x1600

In [42]:

```
input_img = Input((256, 1600, 3), name='img')
model1 = get_unet(input_img, n_filters=8, dropout=0.2, batchnorm=True)
```

```
model1 = get_unet(input_img, n_filters=8, dropout=0.2, batchnorm=True)
model1.compile(optimizer=Adam(), loss=bce_dice_loss, metrics=[dice_coef])
model1.summary()
```

```
Model: "model_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
img (InputLayer)                (None, 256, 1600, 3) 0
__________________________________________________________________________________________________
conv2d_47 (Conv2D)              (None, 256, 1600, 8) 224         img[0][0]
__________________________________________________________________________________________________
batch_normalization_37 (BatchNo (None, 256, 1600, 8) 32          conv2d_47[0][0]
__________________________________________________________________________________________________
activation_37 (Activation)      (None, 256, 1600, 8) 0           batch_normalization_37[0][0]
__________________________________________________________________________________________________
conv2d_48 (Conv2D)              (None, 256, 1600, 8) 584         activation_37[0][0]
__________________________________________________________________________________________________
batch_normalization_38 (BatchNo (None, 256, 1600, 8) 32          conv2d_48[0][0]
__________________________________________________________________________________________________
activation_38 (Activation)      (None, 256, 1600, 8) 0           batch_normalization_38[0][0]
__________________________________________________________________________________________________
max_pooling2d_9 (MaxPooling2D)  (None, 128, 800, 8)  0           activation_38[0][0]
__________________________________________________________________________________________________
dropout_17 (Dropout)            (None, 128, 800, 8)  0           max_pooling2d_9[0][0]
__________________________________________________________________________________________________
conv2d_49 (Conv2D)              (None, 128, 800, 16) 1168        dropout_17[0][0]
__________________________________________________________________________________________________
batch_normalization_39 (BatchNo (None, 128, 800, 16) 64          conv2d_49[0][0]
__________________________________________________________________________________________________
activation_39 (Activation)      (None, 128, 800, 16) 0           batch_normalization_39[0][0]
__________________________________________________________________________________________________
conv2d_50 (Conv2D)              (None, 128, 800, 16) 2320        activation_39[0][0]
__________________________________________________________________________________________________
batch_normalization_40 (BatchNo (None, 128, 800, 16) 64          conv2d_50[0][0]
__________________________________________________________________________________________________
activation_40 (Activation)      (None, 128, 800, 16) 0           batch_normalization_40[0][0]
__________________________________________________________________________________________________
max_pooling2d_10 (MaxPooling2D) (None, 64, 400, 16)  0           activation_40[0][0]
__________________________________________________________________________________________________
dropout_18 (Dropout)            (None, 64, 400, 16)  0           max_pooling2d_10[0][0]
__________________________________________________________________________________________________
conv2d_51 (Conv2D)              (None, 64, 400, 32)  4640        dropout_18[0][0]
__________________________________________________________________________________________________
batch_normalization_41 (BatchNo (None, 64, 400, 32)  128         conv2d_51[0][0]
__________________________________________________________________________________________________
activation_41 (Activation)      (None, 64, 400, 32)  0           batch_normalization_41[0][0]
__________________________________________________________________________________________________
conv2d_52 (Conv2D)              (None, 64, 400, 32)  9248        activation_41[0][0]
__________________________________________________________________________________________________
batch_normalization_42 (BatchNo (None, 64, 400, 32)  128         conv2d_52[0][0]
__________________________________________________________________________________________________
activation_42 (Activation)      (None, 64, 400, 32)  0           batch_normalization_42[0][0]
__________________________________________________________________________________________________
max_pooling2d_11 (MaxPooling2D) (None, 32, 200, 32)  0           activation_42[0][0]
__________________________________________________________________________________________________
dropout_19 (Dropout)            (None, 32, 200, 32)  0           max_pooling2d_11[0][0]
__________________________________________________________________________________________________
conv2d_53 (Conv2D)              (None, 32, 200, 64)  18496       dropout_19[0][0]
__________________________________________________________________________________________________
batch_normalization_43 (BatchNo (None, 32, 200, 64)  256         conv2d_53[0][0]
__________________________________________________________________________________________________
activation_43 (Activation)      (None, 32, 200, 64)  0           batch_normalization_43[0][0]
__________________________________________________________________________________________________
conv2d_54 (Conv2D)              (None, 32, 200, 64)  36928       activation_43[0][0]
__________________________________________________________________________________________________
batch_normalization_44 (BatchNo (None, 32, 200, 64)  256         conv2d_54[0][0]
__________________________________________________________________________________________________
activation_44 (Activation)      (None, 32, 200, 64)  0           batch_normalization_44[0][0]
__________________________________________________________________________________________________
max_pooling2d_12 (MaxPooling2D) (None, 16, 100, 64)  0           activation_44[0][0]
__________________________________________________________________________________________________
dropout_20 (Dropout)            (None, 16, 100, 64)  0           max_pooling2d_12[0][0]
__________________________________________________________________________________________________
conv2d_55 (Conv2D)              (None, 16, 100, 128) 73856       dropout_20[0][0]
__________________________________________________________________________________________________
batch_normalization_45 (BatchNo (None, 16, 100, 128) 512         conv2d_55[0][0]
```

activation_45 (Activation)	(None, 16, 100, 128)	0	batch_normalization_45[0][0]
conv2d_56 (Conv2D)	(None, 16, 100, 128)	147584	activation_45[0][0]
batch_normalization_46 (BatchNo	(None, 16, 100, 128)	512	conv2d_56[0][0]
activation_46 (Activation)	(None, 16, 100, 128)	0	batch_normalization_46[0][0]
up_sampling2d_9 (UpSampling2D)	(None, 32, 200, 128)	0	activation_46[0][0]
conv2d_57 (Conv2D)	(None, 32, 200, 64)	32832	up_sampling2d_9[0][0]
concatenate_9 (Concatenate)	(None, 32, 200, 128)	0	conv2d_57[0][0] activation_44[0][0]
dropout_21 (Dropout)	(None, 32, 200, 128)	0	concatenate_9[0][0]
conv2d_58 (Conv2D)	(None, 32, 200, 64)	73792	dropout_21[0][0]
batch_normalization_47 (BatchNo	(None, 32, 200, 64)	256	conv2d_58[0][0]
activation_47 (Activation)	(None, 32, 200, 64)	0	batch_normalization_47[0][0]
conv2d_59 (Conv2D)	(None, 32, 200, 64)	36928	activation_47[0][0]
batch_normalization_48 (BatchNo	(None, 32, 200, 64)	256	conv2d_59[0][0]
activation_48 (Activation)	(None, 32, 200, 64)	0	batch_normalization_48[0][0]
up_sampling2d_10 (UpSampling2D)	(None, 64, 400, 64)	0	activation_48[0][0]
conv2d_60 (Conv2D)	(None, 64, 400, 32)	8224	up_sampling2d_10[0][0]
concatenate_10 (Concatenate)	(None, 64, 400, 64)	0	conv2d_60[0][0] activation_42[0][0]
dropout_22 (Dropout)	(None, 64, 400, 64)	0	concatenate_10[0][0]
conv2d_61 (Conv2D)	(None, 64, 400, 32)	18464	dropout_22[0][0]
batch_normalization_49 (BatchNo	(None, 64, 400, 32)	128	conv2d_61[0][0]
activation_49 (Activation)	(None, 64, 400, 32)	0	batch_normalization_49[0][0]
conv2d_62 (Conv2D)	(None, 64, 400, 32)	9248	activation_49[0][0]
batch_normalization_50 (BatchNo	(None, 64, 400, 32)	128	conv2d_62[0][0]
activation_50 (Activation)	(None, 64, 400, 32)	0	batch_normalization_50[0][0]
up_sampling2d_11 (UpSampling2D)	(None, 128, 800, 32)	0	activation_50[0][0]
conv2d_63 (Conv2D)	(None, 128, 800, 16)	2064	up_sampling2d_11[0][0]
concatenate_11 (Concatenate)	(None, 128, 800, 32)	0	conv2d_63[0][0] activation_40[0][0]
dropout_23 (Dropout)	(None, 128, 800, 32)	0	concatenate_11[0][0]
conv2d_64 (Conv2D)	(None, 128, 800, 16)	4624	dropout_23[0][0]
batch_normalization_51 (BatchNo	(None, 128, 800, 16)	64	conv2d_64[0][0]
activation_51 (Activation)	(None, 128, 800, 16)	0	batch_normalization_51[0][0]
conv2d_65 (Conv2D)	(None, 128, 800, 16)	2320	activation_51[0][0]
batch_normalization_52 (BatchNo	(None, 128, 800, 16)	64	conv2d_65[0][0]
activation_52 (Activation)	(None, 128, 800, 16)	0	batch_normalization_52[0][0]
up_sampling2d_12 (UpSampling2D)	(None, 256, 1600, 16	0	activation_52[0][0]
conv2d_66 (Conv2D)	(None, 256, 1600, 8)	520	up_sampling2d_12[0][0]
concatenate_12 (Concatenate)	(None, 256, 1600, 16	0	conv2d_66[0][0]

```
                                                                  activation_38[0][0]
__________________________________________________________________________________________________
dropout_24 (Dropout)            (None, 256, 1600, 16 0           concatenate_12[0][0]
__________________________________________________________________________________________________
conv2d_67 (Conv2D)              (None, 256, 1600, 8) 1160        dropout_24[0][0]
__________________________________________________________________________________________________
batch_normalization_53 (BatchNo (None, 256, 1600, 8) 32          conv2d_67[0][0]
__________________________________________________________________________________________________
activation_53 (Activation)      (None, 256, 1600, 8) 0           batch_normalization_53[0][0]
__________________________________________________________________________________________________
conv2d_68 (Conv2D)              (None, 256, 1600, 8) 584         activation_53[0][0]
__________________________________________________________________________________________________
batch_normalization_54 (BatchNo (None, 256, 1600, 8) 32          conv2d_68[0][0]
__________________________________________________________________________________________________
activation_54 (Activation)      (None, 256, 1600, 8) 0           batch_normalization_54[0][0]
__________________________________________________________________________________________________
conv2d_69 (Conv2D)              (None, 256, 1600, 4) 36          activation_54[0][0]
==================================================================================================
Total params: 488,788
Trainable params: 487,316
Non-trainable params: 1,472
__________________________________________________________________________________________________
```

In [0]:

```
model1.set_weights(model.get_weights())
```

In [45]:

```
# Predicting on test data

data_path = 'test_images/'
files = list(os.listdir(data_path))
img_ID= []
classId = []
rle_lst = []
img_classId= []
for f in tqdm(files):
    X = np.empty((1,256,1600,3),dtype=np.uint8)
    img = cv2.imread(data_path + f)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # img = cv2.resize(img, (800,128))
    X[0,] = img
    mask = model1.predict(X)
    rle_m = np.empty((256,1600),dtype=np.uint8)
    for i in range(4):
        rle_m = mask[0,:,:,i].round().astype(int)
        rle = mask2rle(rle_m)
        rle_lst.append(rle)
        img_ID.append(f)
        classId.append(str(i+1))
        img_classId.append(f+'_'+str(i+1))
```

Creating csv file for Kaggle submission

In [0]:

```
output = {'ImageId_ClassId':img_classId, 'EncodedPixels' : rle_lst}

filepath= "/content/drive/My Drive/Colab Notebooks/Steel Defect Detection/"
output_df = pd.DataFrame(output)
output_df.to_csv(filepath+'submission_fullsize2.csv', index=False)
```