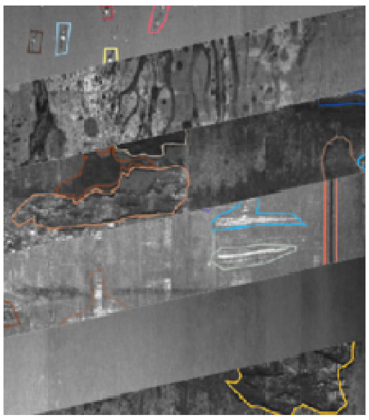
Steel Defect Detection: Image Segmentation using Keras

Karthik Billa
Feb 12 · 12 min read

Author: Karthik [LinkedIn]

- Detect and classify defects in steel

Keywords: *Steel, Defect, Identification, Localization, Dice coefficient, segmentation models, Tensorflow, Run Length Encoding*



1. Business Problem

1.1 Introduction:

Steel is one of the most important building materials of modern times. Steel buildings are resistant to natural and man-made wear which has made the material ubiquitous around the world. Identifying defects will help make production of steel more efficient. Severstal is leading the charge in efficient steel mining and production.

Credits: https://www.kaggle.com/c/severstal-steel-defect-detection/overview

1.2 Problem description:

Severstal is now looking to machine learning to improve automation, increase efficiency, and maintain high quality in their production.

The production process of flat sheet steel is especially delicate. From heating and rolling, to drying and cutting, several machines touch flat steel by the time it's ready to ship. Today, Severstal uses images from high frequency cameras to power a defect detection algorithm.

This notebook will help engineers improve the algorithm by localizing and classifying surface defects on a steel sheet.

1.3 Source/Useful Links:

- Data Source
- Competition hosting company
- For Classification: Xception
- For Segmentation: Unet — EfficientNetB1
- Training and predictions: Google Colab
- Installing segmentation_models packages in Kaggle Kernel (useful for making an Inference kernel on Kaggle Platform)
- Submitting *.csv on kaggle

1.4 Business objectives and constraints:

- Maximize dice score
- Multi-label probability estimates
- Defect identification and localization should not take much time. In an ideal situation it is desirable to match with the frequency of cameras. It should finish in a few seconds. Inference kernel should take <= 1 hours run-time.
- Save model weights to make inference possible anytime.

2. Deep Learning Problem

2.1 Data Description

```
Folder/

    sample_submission.csv    3 columns

    train.csv                3 columns

    test_images/             5506 .jpg images

    train_images/            12568 .jpg images
```

Each image is of **256x1600** resolution. “train.csv” contains defect present image details. Its columns are:

```
ImageId, Class, EncodedPixels
```

Test data ImageIds can be found in sample_submission.csv or can be directly accessed from Image file names. Corresponding images can be accessed from train and test folders with the help of ImageIds.

- Number of Defect Classes: 4

2.2 Translating to Deep Learning Problem

2.2.1 Type of Deep Learning Problem

There are 4 different classes of steel surface defects and we need to locate the defect => Multi-label Image Segmentation

2.2.2 Performance Metric:

Dice coefficient:

This metric is used to gauge similarity of two samples. 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. The Dice coefficient is defined to be 1 when both X and Y are empty. The leaderboard score is the **mean of the Dice coefficients** for each [ImageId, ClassId] pair in the test set.

2.2.3 Deep Learning Objectives

Objective:

- Maximize Dice coefficient
- Identify and locate the type of defect present in the image. Masks generated after predictions should be converted into EncodedPixels.

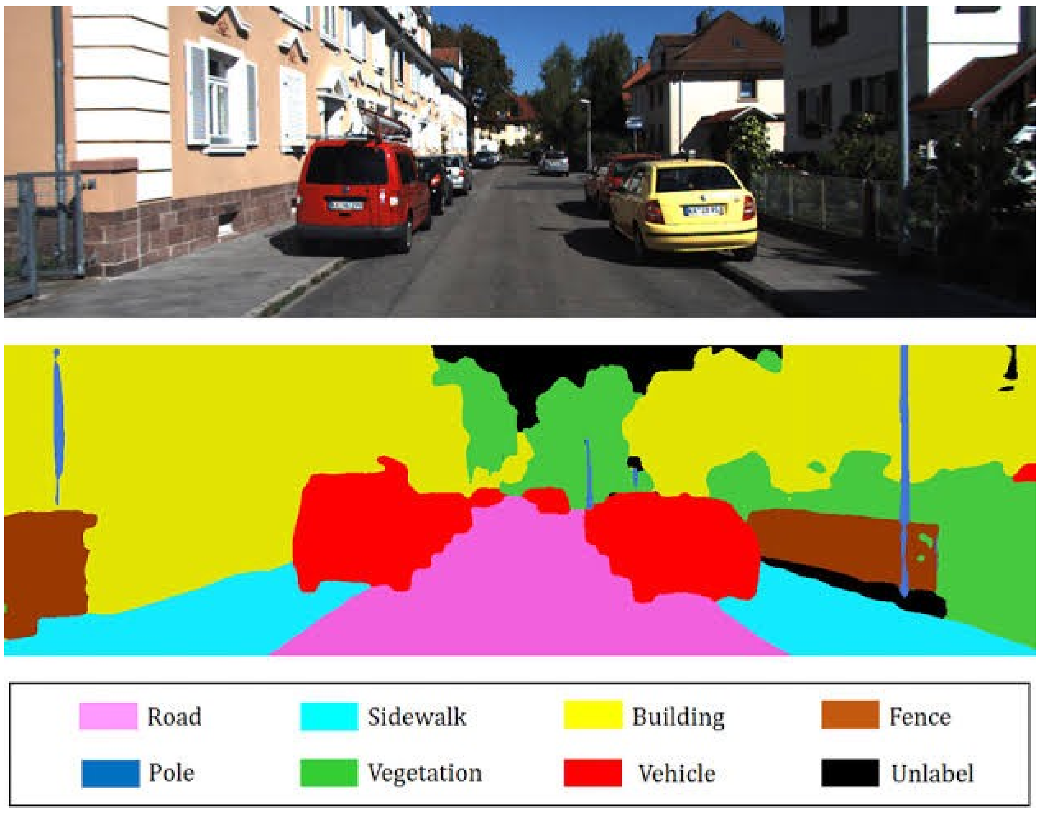
EncodedPixels:

> *In order to reduce the submission file size, our metric uses run-length encoding on the pixel values. Instead of submitting an exhaustive list of indices for your segmentation, you will submit pairs of values that contain a start position and a run length. E.g. ‘1 3’ implies starting at pixel 1 and running a total of 3 pixels (1,2,3).*
>
> *The competition format requires a space delimited list of pairs. For example, ‘1 3 10 5’ implies pixels 1,2,3,10,11,12,13,14 are to be included in the mask. The metric checks that*

> *the pairs are sorted, positive, and the decoded pixel values are not duplicated. The pixels are numbered from top to bottom, then left to right: 1 is pixel (1,1), 2 is pixel (2,1), etc.*

Image Segmentation:

> *In computer vision, image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels, also known as image objects). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images.*



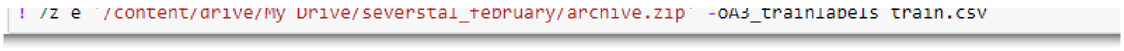
Google Images

> *More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.*
>
> *In the adjacent image, the original is hard to analyze with the help of computer vision models. With the help of image segmentation we can partition the image into multiple segments. This will make it easy for the computer to learn from patterns in these multiple segments. For example, each pixel belonging to cars is colored red.*

3. Data Preparation:

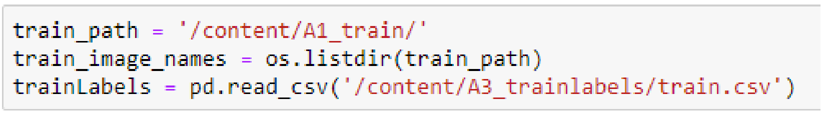
Available data is not in the X_train, Y_train format, we need to generate these with the help of getting image names from train_images folder and merging these with train.csv as:

```
# https://stackoverflow.com/questions/31984387/command-line-for-7z-to-extract-specific-files-from-spec
# extracting raw data
! 7z e '/content/drive/My Drive/severstal_february/archive.zip' -oA1_train       train_images/*.jpg
```



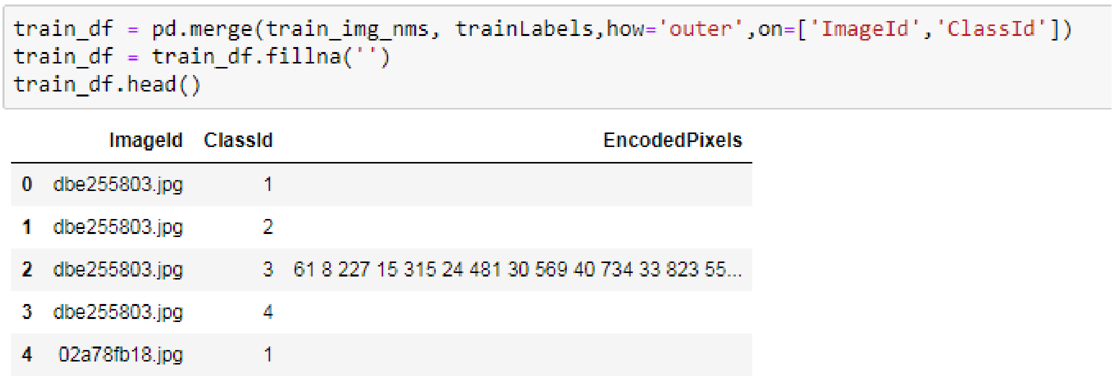
```
! 7z e '/content/drive/My Drive/severstal_february/archive.zip' -oA3_trainlabels train.csv
```

Data Extraction (archive.zip is downloaded from Kaggle to Google Drive)



```
train_path = '/content/A1_train/'
train_image_names = os.listdir(train_path)
trainLabels = pd.read_csv('/content/A3_trainlabels/train.csv')
```

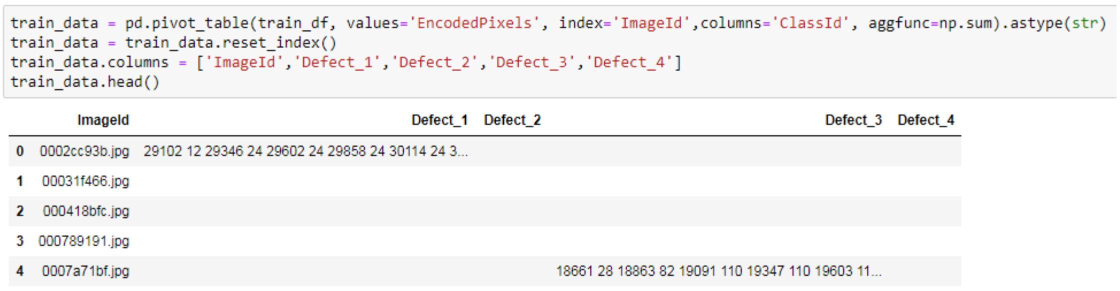
Data Loading



```
train_df = pd.merge(train_img_nms, trainLabels,how='outer',on=['ImageId','ClassId'])
train_df = train_df.fillna('')
train_df.head()
```

	ImageId	ClassId	EncodedPixels
0	dbe255803.jpg	1	
1	dbe255803.jpg	2	
2	dbe255803.jpg	3	61 8 227 15 315 24 481 30 569 40 734 33 823 55...
3	dbe255803.jpg	4	
4	02a78fb18.jpg	1	

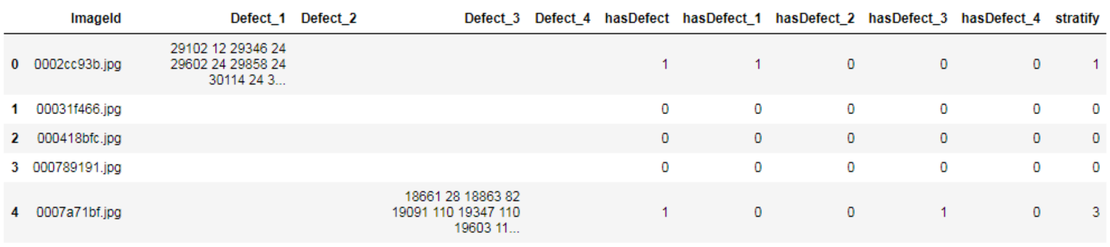
Full train dataset



```
train_data = pd.pivot_table(train_df, values='EncodedPixels', index='ImageId',columns='ClassId', aggfunc=np.sum).astype(str)
train_data = train_data.reset_index()
train_data.columns = ['ImageId','Defect_1','Defect_2','Defect_3','Defect_4']
train_data.head()
```

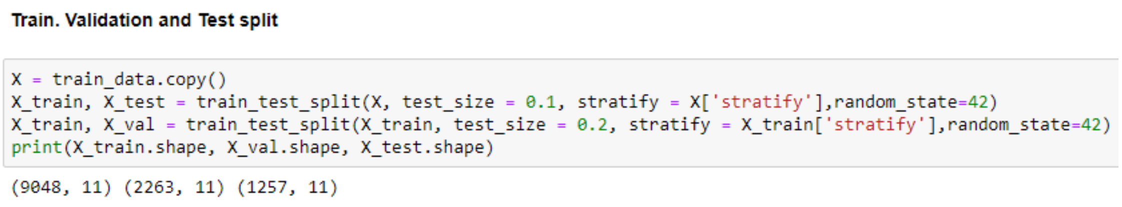
	ImageId	Defect_1	Defect_2	Defect_3	Defect_4
0	0002cc93b.jpg	29102 12 29346 24 29602 24 29858 24 30114 24 3...			
1	00031f466.jpg				
2	000418bfc.jpg				
3	000789191.jpg				
4	0007a71bf.jpg			18661 28 18863 82 19091 110 19347 110 19603 11...	

Simple DataFrame modifications



	ImageId	Defect_1	Defect_2	Defect_3	Defect_4	hasDefect	hasDefect_1	hasDefect_2	hasDefect_3	hasDefect_4	stratify
0	0002cc93b.jpg	29102 12 29346 24 29602 24 29858 24 30114 24 3...				1	1	0	0	0	1
1	00031f466.jpg					0	0	0	0	0	0
2	000418bfc.jpg					0	0	0	0	0	0
3	000789191.jpg					0	0	0	0	0	0
4	0007a71bf.jpg			18661 28 18863 82 19091 110 19347 110 19603 11...		1	0	0	1	0	3

Final result from DataFrame modifications



Train. Validation and Test split

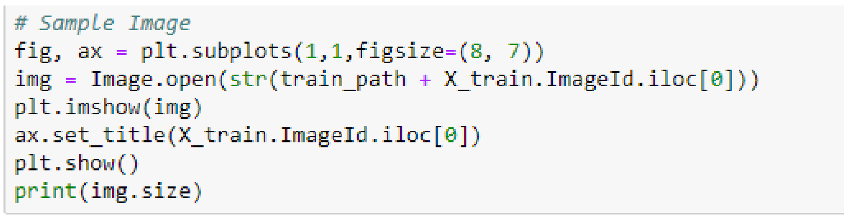
```
X = train_data.copy()
X_train, X_test = train_test_split(X, test_size = 0.1, stratify = X['stratify'],random_state=42)
X_train, X_val = train_test_split(X_train, test_size = 0.2, stratify = X_train['stratify'],random_state=42)
print(X_train.shape, X_val.shape, X_test.shape)
```

```
(9048, 11) (2263, 11) (1257, 11)
```

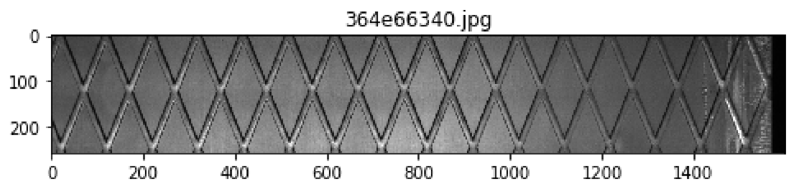
The final stage for Data Preparation

Using train_test_split before Exploratory Data Analysis we will avoid any kind of data leakage.

4. Exploratory Data Analysis:



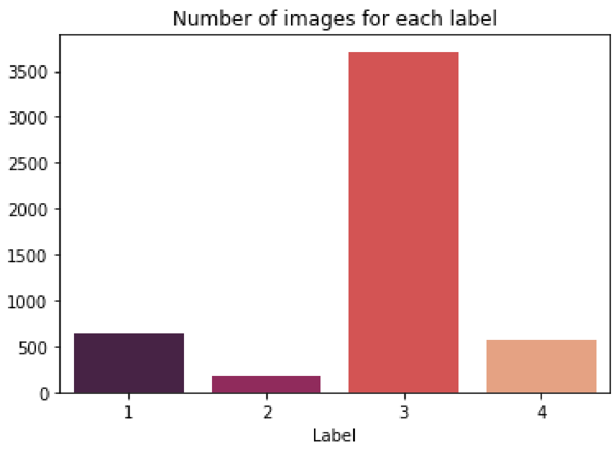
```
# Sample Image
fig, ax = plt.subplots(1,1,figsize=(8, 7))
img = Image.open(str(train_path + X_train.ImageId.iloc[0]))
plt.imshow(img)
ax.set_title(X_train.ImageId.iloc[0])
plt.show()
print(img.size)
```



```
(1600, 256)
```

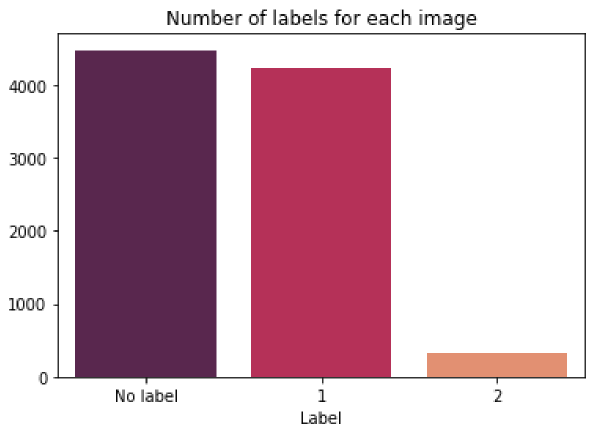
Summary: The images have 1600x256 pixel resolution

```
No. of Images in train set:  9048
 --------------------------------------------------
```



```
No. of Images having: Label 1 = 645, Label 2 = 178, Label 3 = 3712, Label 4 = 576
```

```
 ------------------------------------------------
```



```
No. of Images with no defects: 4249, with only one label: 4487, with two labels: 312
```

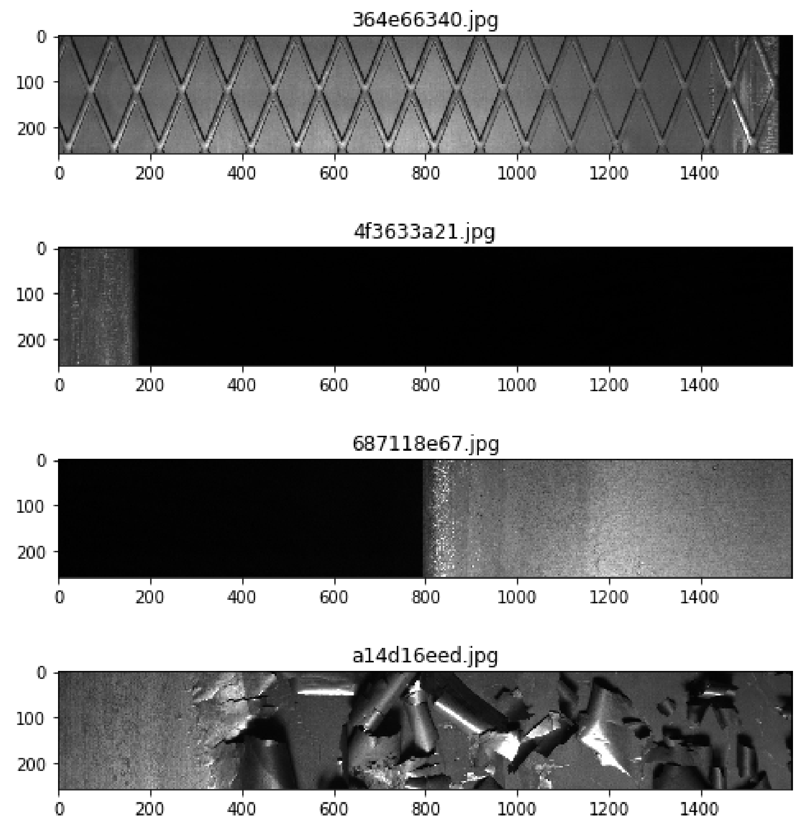
Observation: The dataset is highly imbalanced. This will make predicting minority class (Class 2) difficult

Observation: The dataset is highly imbalanced. This will make predicting minority class (Class 2) difficult.

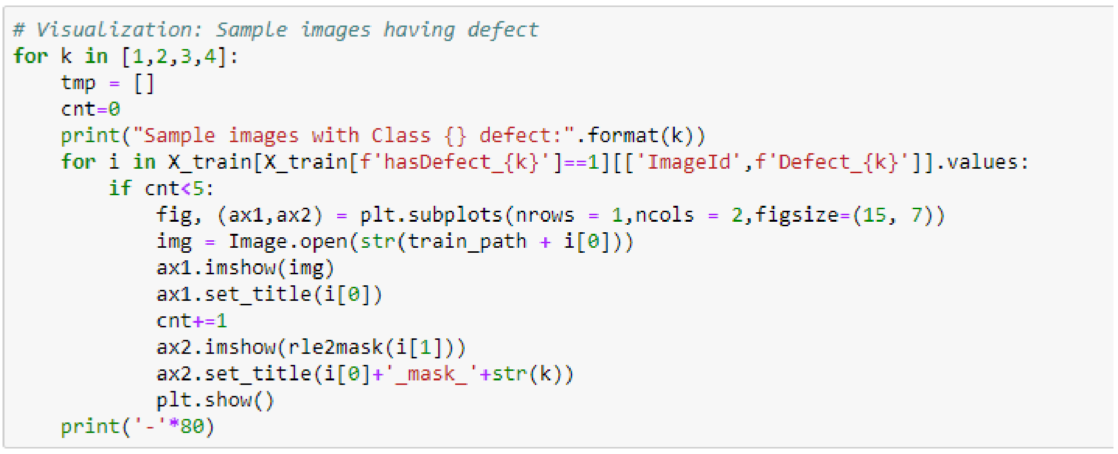
```python
# We need a function to convert EncodedPixels into mask
# https://www.kaggle.com/paulorzp/rle-functions-run-lenght-encode-decode

def rle2mask(mask_rle, shape=(1600,256)):
    '''
    mask_rle: run-length as string formated (start length)
    shape: (width,height) of array to return
    Returns numpy array, 1 - mask, 0 - background
    This function is specific to this competition

    '''
    s = mask_rle.split()
    starts, lengths = [np.asarray(x, dtype=int) for x in (s[0:][::2], s[1:][::2])]
    starts -= 1
    ends = starts + lengths
    img = np.zeros(shape[0]*shape[1], dtype=np.uint8)
    for lo, hi in zip(starts, ends):
        img[lo:hi] = 1
    return img.reshape(shape).T

def mask2rle(img):
    '''
    img: numpy array, 1 - mask, 0 - background
    Returns run length as string formated
    This function is specific to this competition
    '''
    pixels= img.T.flatten()
    pixels = np.concatenate([[0], pixels, [0]])
    runs = np.where(pixels[1:] != pixels[:-1])[0] + 1
    runs[1::2] -= runs[::2]
    return ' '.join(str(x) for x in runs)
```

Utility Functions for conversions between Run Length Encodings and Image Masks

```
Sample images with no defects:
```



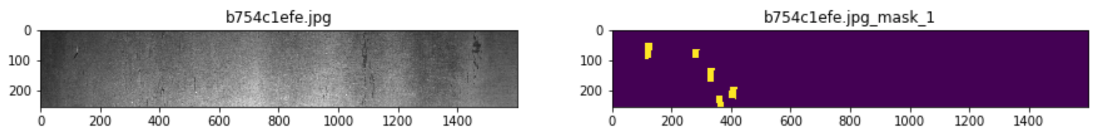
Observation: The surface of the non-defective steel may contain different features or profile. It has to be noted that that presence of defect is limited to the 4 types of defects in this dataset. The steel surface may contain other defects but those should not be detected.

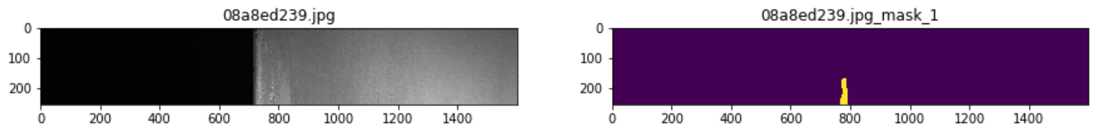


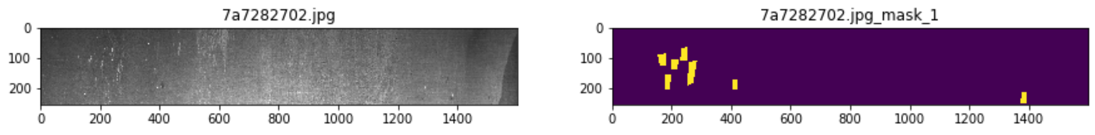
```
# Visualization: Sample images having defect
for k in [1,2,3,4]:
    tmp = []
    cnt=0
    print("Sample images with Class {} defect:".format(k))
    for i in X_train[X_train[f'hasDefect_{k}']==1][['ImageId',f'Defect_{k}']].values:
        if cnt<5:
            fig, (ax1,ax2) = plt.subplots(nrows = 1,ncols = 2,figsize=(15, 7))
            img = Image.open(str(train_path + 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('-'*80)
```

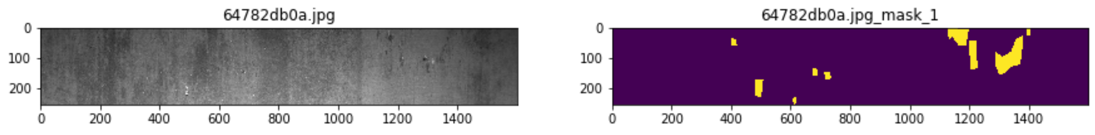
Code for train images visualization

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

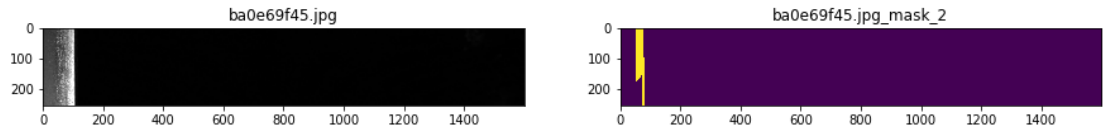


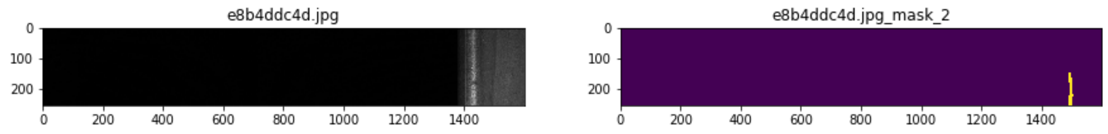


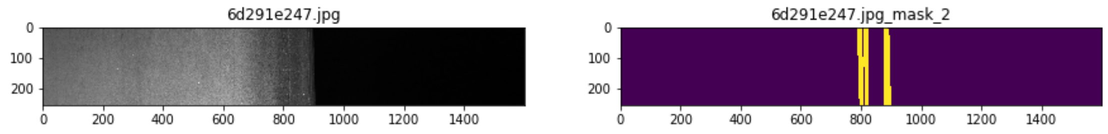


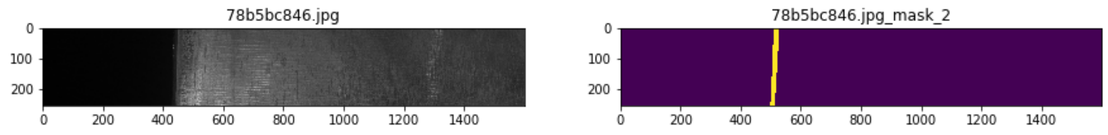


```
--------------------------------------------------------------------------------
Sample images with Class 2 defect:
```

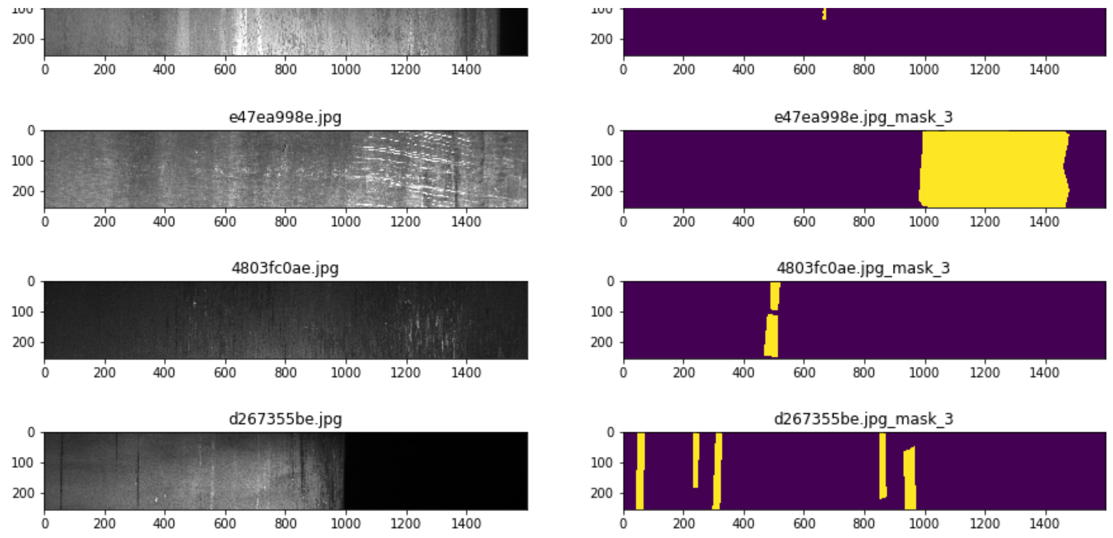




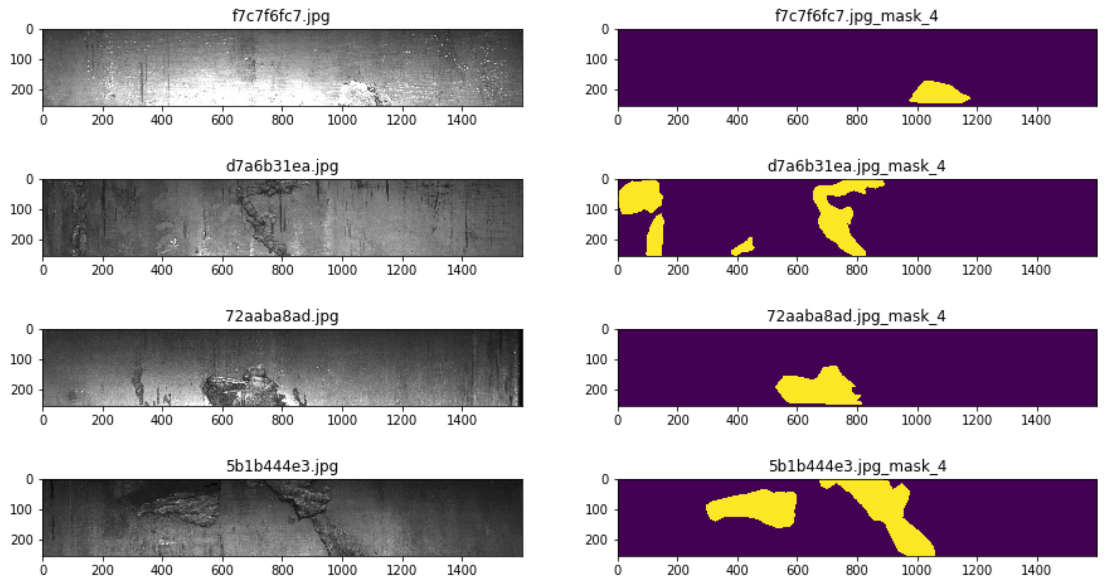




```
--------------------------------------------------------------------------------
Sample images with Class 3 defect:
```



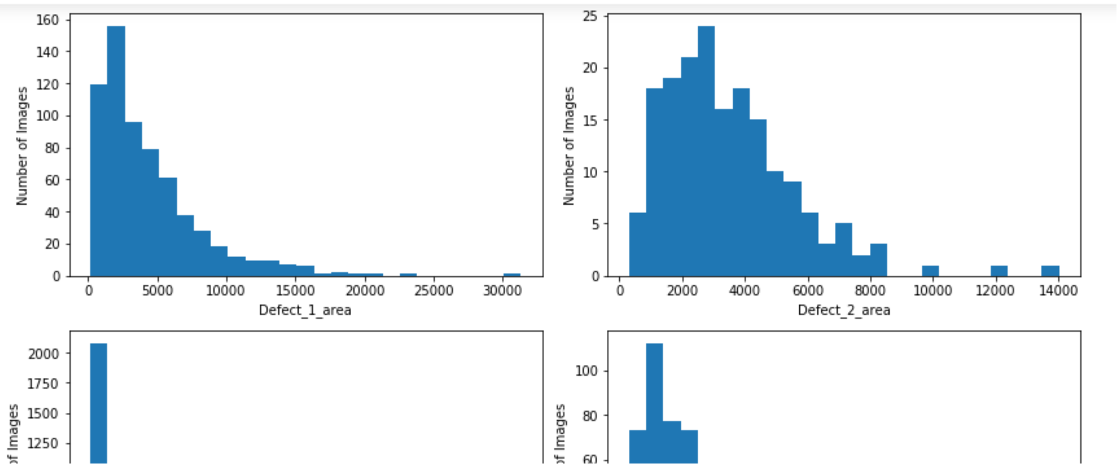
```
--------------------------------------------------------------------------------
Sample images with Class 4 defect:
```

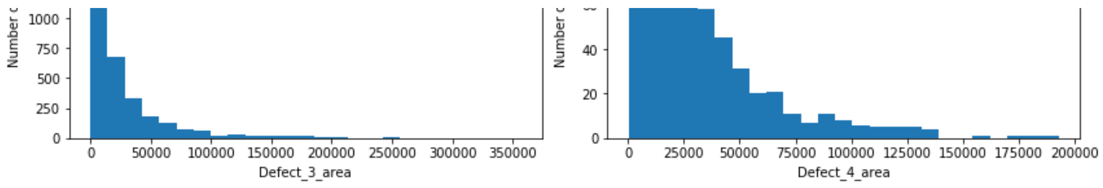


Observation: The regional profile on the masks of defect containing steel surfaces can be seen to be distinguishable among different classes. Defect type 1 can be seen to have multiple small size regions and defect type 4 images have multiple regions of medium size. Defect type 3 images can be seen to also contain multiple regions of medium size. While defect type 2 and type 3 images can be seen to share some regional characteristics.

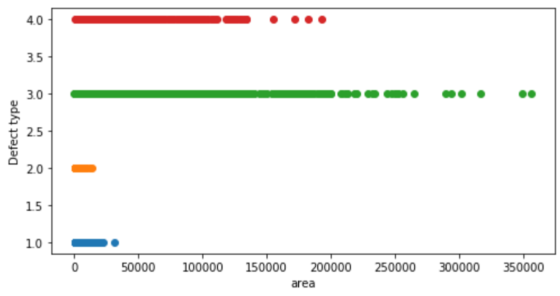
4.1 'area' as a new feature

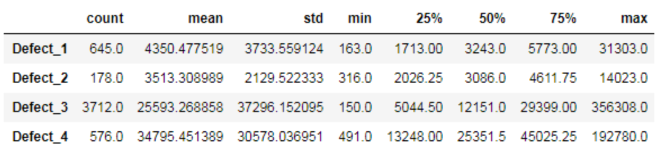
Used for thresholding masks after generating predictions





removing areas below 2 percentile and above 98 percentile to threshold area of predicted masks

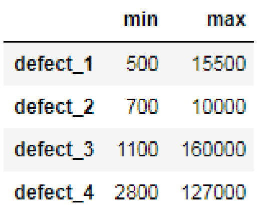




	count	mean	std	min	25%	50%	75%	max
Defect_1	645.0	4350.477519	3733.559124	163.0	1713.00	3243.0	5773.00	31303.0
Defect_2	178.0	3513.308989	2129.522333	316.0	2026.25	3086.0	4611.75	14023.0
Defect_3	3712.0	25593.268858	37296.152095	150.0	5044.50	12151.0	29399.00	356308.0
Defect_4	576.0	34795.451389	30578.036951	491.0	13248.00	25351.5	45025.25	192780.0

Observation:

There is considerable overlap in the range of area. Minimum area for each defect type can be seen closer to each other. While the maximum is largely different. We can use the minimum and maximum values of area in training images to threshold test image defect predictions.



	min	max
defect_1	500	15500
defect_2	700	10000
defect_3	1100	160000
defect_4	2800	127000

'area' thresholds selected

Summary:

> *Based on range of area for each defect, we will threshold predictions to filter outliers. For e.g. some predicted masks have only 4 pixels that have value 1. Such an image will reduce the performance of the model on the final metric.*

4.2 EDA conclusion:

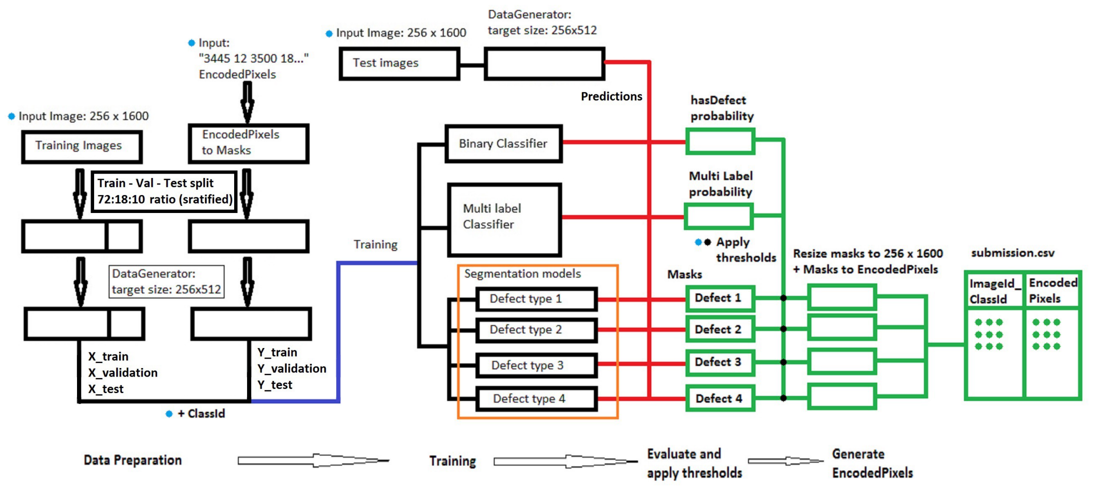
- The dataset is imbalanced thus we will use stratified sampling for splitting the dataset into train and validation datasets.
- This is a multi-label image segmentation problem. As there are around 50% of images with no defects, it is equally important to identify images with no defects.
- Based on area thresholds from 'test_thresolds' dataframe and class probability thresholds (which are to be determined after predictions from neural networks), we will ensure that number of predicted images per defect will be closer to the values in 'count' column.

Procedure:

- We will have a binary classification model to filter images with defects from no defect images.
- A 4-label classification model to predict probablities of images beloning to each class.
- 4 segmentation models for four different classes to generate masks for each test image.
- Convert masks to EncodedPixels and filter them as per classification probabilities.

We are generating a new solution to the business problem with available libraries: tensorflow, keras and segmentation_models.

4.3 Model Architecture:



Blue dots in the Architecture image indicates that an input is being given at that level, while black dot near "Apply threholds" correspond to the application of thresholds at the output of predicted masks. At the threshold application level images are filtered based on Defect presence probability, Defect type belongingness and area of the defect.

> **Note:** It is important to take care that right training data is fed into each model. The **effect of training data on loss function** guides us through this. Binary Classifier will be trained **with all images**. Multi-Label Classifier will be trained with **Images having defects**. The defined architecture has 4 output neurons which equals with the number of Classes. Multi-label classifier training images can include defect present images and defect absent images as well if 5 neurons were chosen 4 for defect classes and 5th for "no defect" class. Here, additional Binary Classifier model becomes redundant. When using Multi-label classifier with 4 output neurons, feeding no defect images(X) implies all target data(Y) is 0 ([0,0,0,0]) which results in 'zero' loss. *There will be no training or weight updates if loss is 'zero'.* Similarly segmentation models are trained on each defect separately. Thus, here we are using 4 segmentation models each trained separately on each defect. This is the scheme utilised in this approach while other schemes can be used and the training data fed into the model should be appropriate to the model

> defined. Loss function also plays a role on deciding what training data is used for the model.

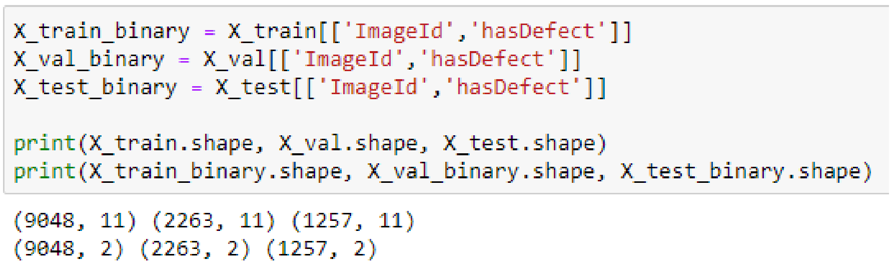
5. Data generators and Model Building

Look through Github Notebook for Data Generator definition and custom metrics.

Thresholding for high precision with slight compromise on overall recall is followed to get a good Competition metric.

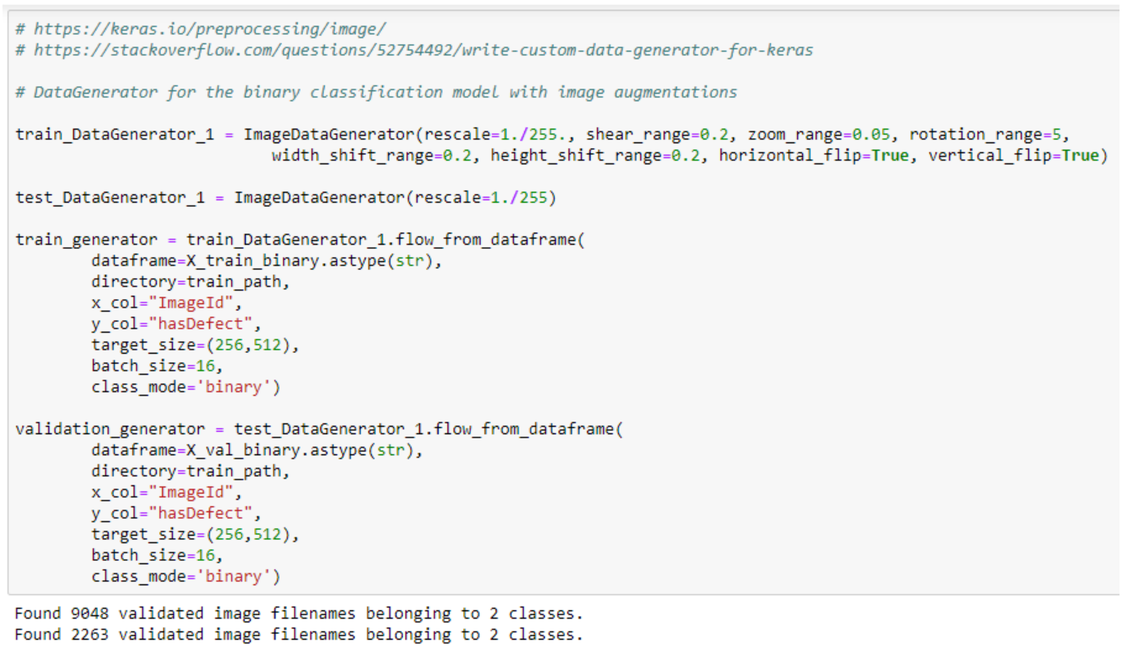
5.1 Binary Classifier:

- Train and predict the probability of presence of defects in images



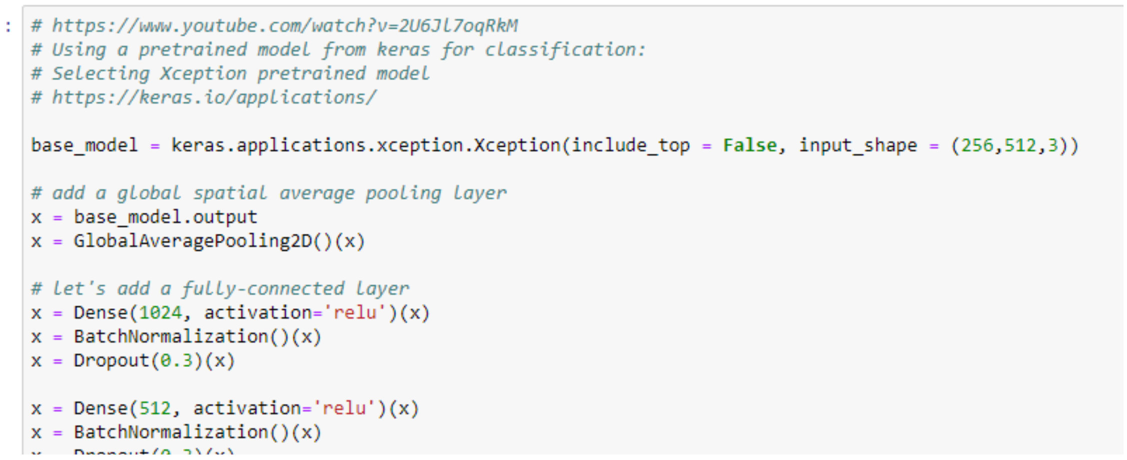
```
X_train_binary = X_train[['ImageId','hasDefect']]
X_val_binary = X_val[['ImageId','hasDefect']]
X_test_binary = X_test[['ImageId','hasDefect']]

print(X_train.shape, X_val.shape, X_test.shape)
print(X_train_binary.shape, X_val_binary.shape, X_test_binary.shape)
```

```
(9048, 11) (2263, 11) (1257, 11)
(9048, 2) (2263, 2) (1257, 2)
```



```
# https://keras.io/preprocessing/image/
# https://stackoverflow.com/questions/52754492/write-custom-data-generator-for-keras

# DataGenerator for the binary classification model with image augmentations

train_DataGenerator_1 = ImageDataGenerator(rescale=1./255., shear_range=0.2, zoom_range=0.05, rotation_range=5,
                           width_shift_range=0.2, height_shift_range=0.2, horizontal_flip=True, vertical_flip=True)

test_DataGenerator_1 = ImageDataGenerator(rescale=1./255)

train_generator = train_DataGenerator_1.flow_from_dataframe(
        dataframe=X_train_binary.astype(str),
        directory=train_path,
        x_col="ImageId",
        y_col="hasDefect",
        target_size=(256,512),
        batch_size=16,
        class_mode='binary')

validation_generator = test_DataGenerator_1.flow_from_dataframe(
        dataframe=X_val_binary.astype(str),
        directory=train_path,
        x_col="ImageId",
        y_col="hasDefect",
        target_size=(256,512),
        batch_size=16,
        class_mode='binary')
```

```
Found 9048 validated image filenames belonging to 2 classes.
Found 2263 validated image filenames belonging to 2 classes.
```

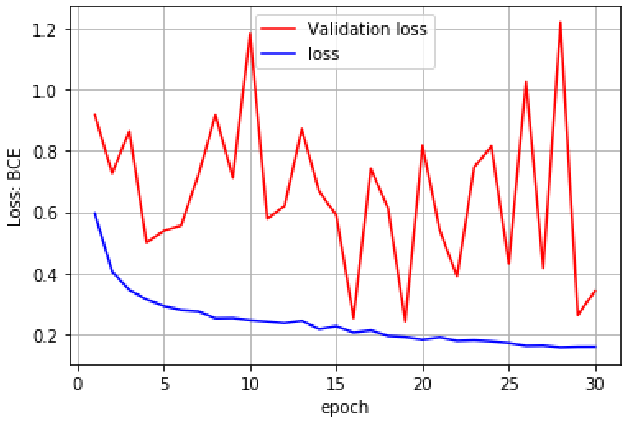


```
# https://www.youtube.com/watch?v=2U6Jl7oqRkM
# Using a pretrained model from keras for classification:
# Selecting Xception pretrained model
# https://keras.io/applications/

base_model = keras.applications.xception.Xception(include_top = False, input_shape = (256,512,3))

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)

# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.3)(x)

x = Dense(512, activation='relu')(x)
x = BatchNormalization()(x)
```

```
x = Dropout(0.3)(x)

x = Dense(64, activation='relu')(x)

# and the prediction layer
predictions = Dense(1, activation='sigmoid')(x)

# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)
```

Binary Classifier model definition

```
Total params: 23,523,497
Trainable params: 23,465,897
Non-trainable params: 57,600
```



Binary cross entropy loss of the model can be seen to have large variations on validation set. This implies that the model is having tough time generalizing on unseen dataset when predicting presence of defects.

- **Best weights found @19epoch:**

```
Train set evaluation score:
```

binary_crossentropy	0.202241
acc	0.923630
f1_score_m	0.921999
precision_m	0.949316
recall_m	0.905966

```
Validation set evaluation score:
```

binary_crossentropy	0.240638
acc	0.912064
f1_score_m	0.912423
precision_m	0.937087
recall_m	0.898664

```
Test set evaluation score:
```

binary_crossentropy	0.194755
acc	0.926810
f1_score_m	0.921435
precision_m	0.955327
recall_m	0.902135

> ***Summary:*** *The model is having good performance on train, validation and test dataset. The values of loss and metrics can be seen to be similar in these datasets. This tells that the model is not overfitting on dataset. The f1_score of 0.921 on validation dataset is acceptable*

5.2 MultiLabel Classifier:

- Predict probability of presence of each defect in an image

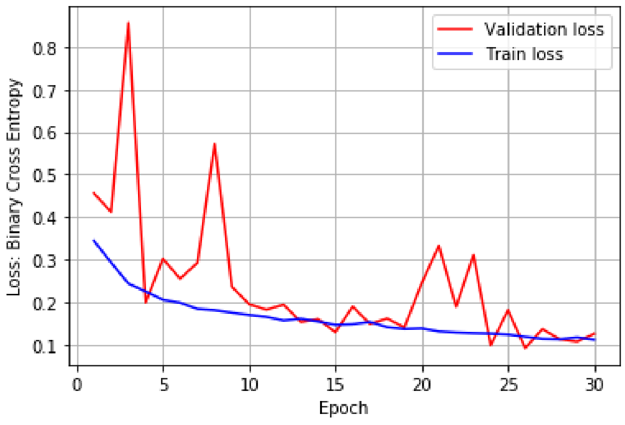
```
X_train_multi = X_train[['ImageId','hasDefect_1','hasDefect_2','hasDefect_3','hasDefect_4']][X_train['hasDefect']==1]
X_val_multi = X_val[['ImageId','hasDefect_1','hasDefect_2','hasDefect_3','hasDefect_4']][X_val['hasDefect']==1]
X_test_multi = X_test[['ImageId','hasDefect_1','hasDefect_2','hasDefect_3','hasDefect_4']][X_test['hasDefect']==1]

print(X_train.shape, X_val.shape, X_test.shape)
print(X_train_multi.shape, X_val_multi.shape, X_test_multi.shape)
```

```
(9048, 11) (2263, 11) (1257, 11)
(4799, 5) (1200, 5) (667, 5)
```

```
x = Dense(64, activation='relu')(x)

# and the prediction layer
predictions = Dense(4, activation='sigmoid')(x)

# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)
```

Model similar to Binary Classifier with 4 output neurons

```
Total params: 23,523,692
Trainable params: 23,466,092
Non-trainable params: 57,600
```



Training better than Binary Classifier

```
Train set evaluation score:
```

binary_crossentropy	0.081054
acc	0.968118
f1_score_m	0.940510
precision_m	0.945815
recall_m	0.937232

```
Validation set evaluation score:
```

binary_crossentropy	0.092119
acc	0.962500
f1_score_m	0.929417
precision_m	0.929264
recall_m	0.931588

```
Test set evaluation score:
```

binary_crossentropy	0.094178
acc	0.965517
f1_score_m	0.936398
precision_m	0.941134
recall_m	0.933854

> ***Summary:*** *The multi-label classification model is generalizing well on unseen data (the values of evaluation on test set and validation set are closer to train set).*

5.3 Image Segmentation:

- Data preparation:

```
# Dividing the datasets w.r.t. Class Label (defect type)
train_data_1 = X_train[X_train['hasDefect_1']==1][['ImageId','Defect_1']]
train_data_2 = X_train[X_train['hasDefect_2']==1][['ImageId','Defect_2']]
train_data_3 = X_train[X_train['hasDefect_3']==1][['ImageId','Defect_3']]
train_data_4 = X_train[X_train['hasDefect_4']==1][['ImageId','Defect_4']]
```

Data preparation for each model

- Model definition:

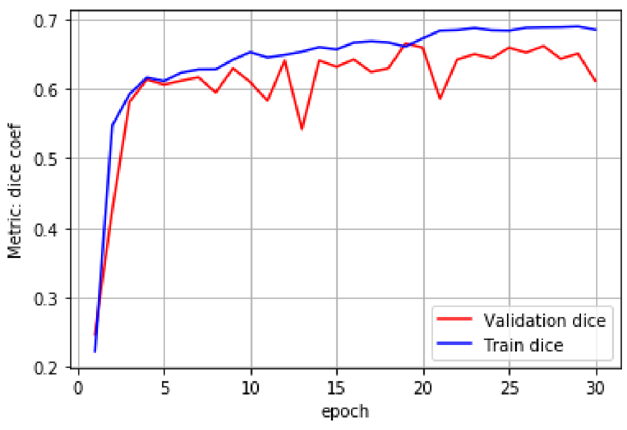
```
# https://github.com/tensorflow/tpu/blob/master/models/official/efficientnet/preprocessing.py
# preprocesses image to input to the segmentation_model, generally image pixel value standardization
preprocess = get_preprocessing('efficientnetb1')

# https://github.com/qubvel/segmentation_models
# segmentation using pretrained weights for faster convergence
model = Unet('efficientnetb1', classes=1, activation='sigmoid', encoder_weights='imagenet')
```

Legendary UNet with EfficientNetB1 backbone is used for Segmentation purposes.

```
Total params: 12,641,169
Trainable params: 12,577,137
Non-trainable params: 64,032
```

5.3.1 Defect Label 1:

- Dice coefficient vs epoch plot for training the segmentation model on defect 1



Segmentation model Training: Defect 1

Train set evaluation score:

dice_loss	0.285742
dice_coef	0.714258

Validation set evaluation score:

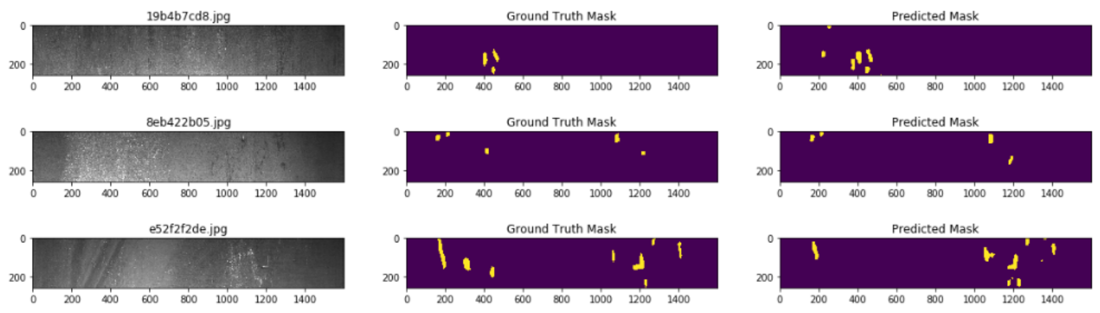
dice_loss	0.334879
dice_coef	0.665121

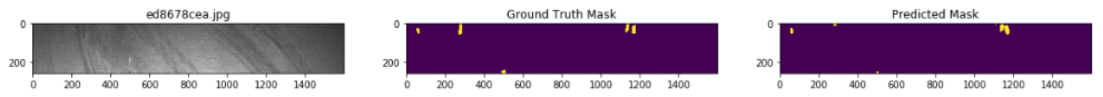
Test set evaluation score:

dice_loss	0.388797
dice_coef	0.611203

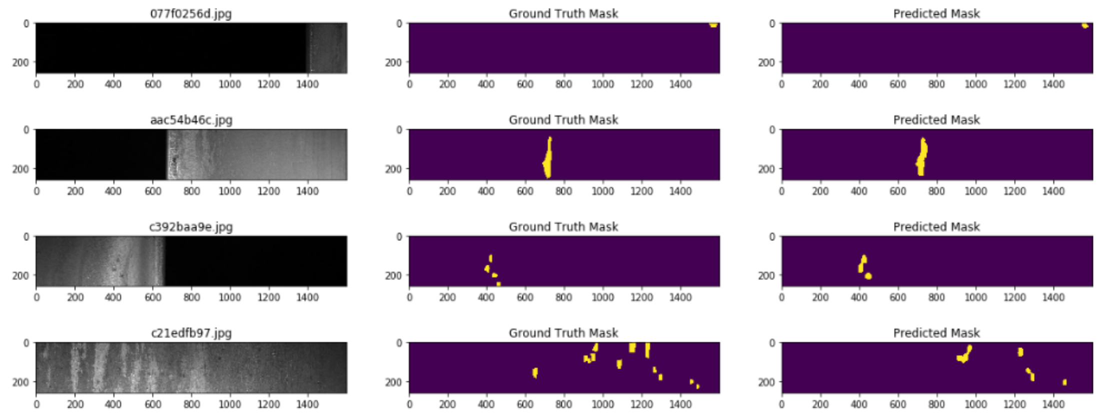
Evaluations: Defect 1

Note: Dice coefficient is also known as F1_score.

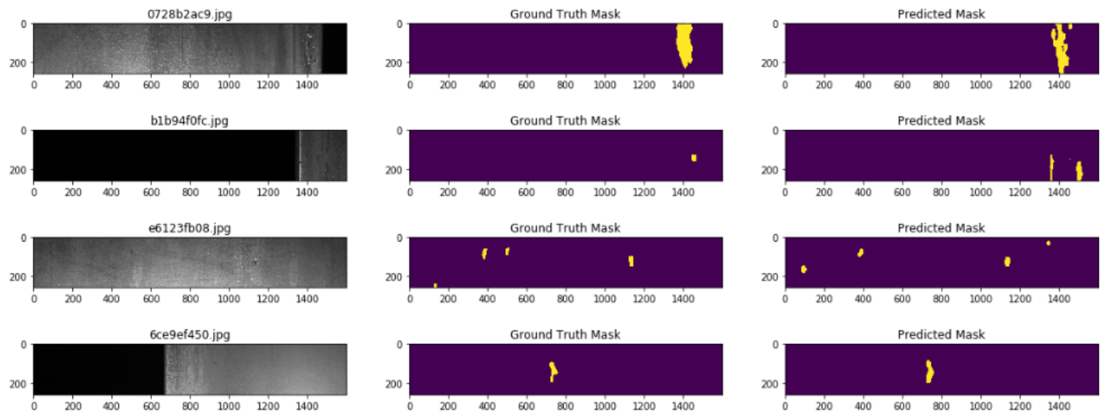




Defect 1 Train set Images



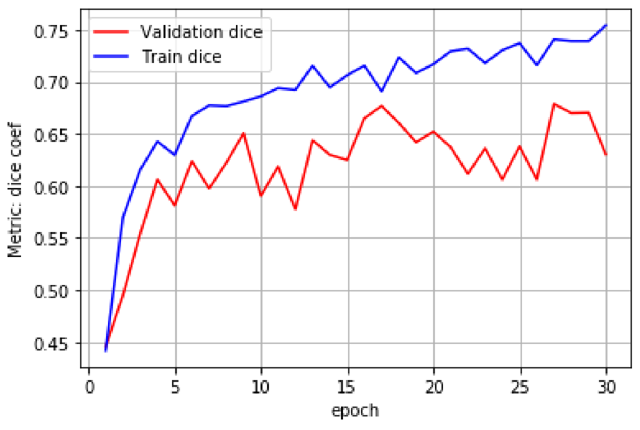
Defect 1 Validation set



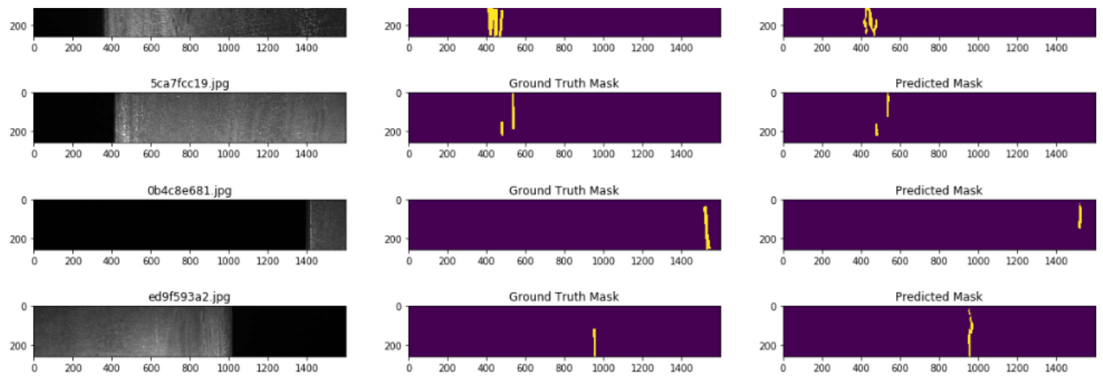
Defect 1 Test set (Unseen data)

5.3.2 Defect Label 2:

- Dice coefficient vs epoch plot for training the segmentation model on defect 2



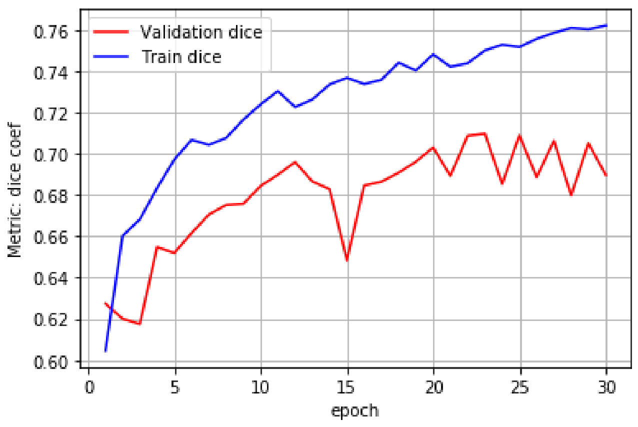
Segmentation model Training: Defect 2



Test set

5.3.3 Defect Label 3:

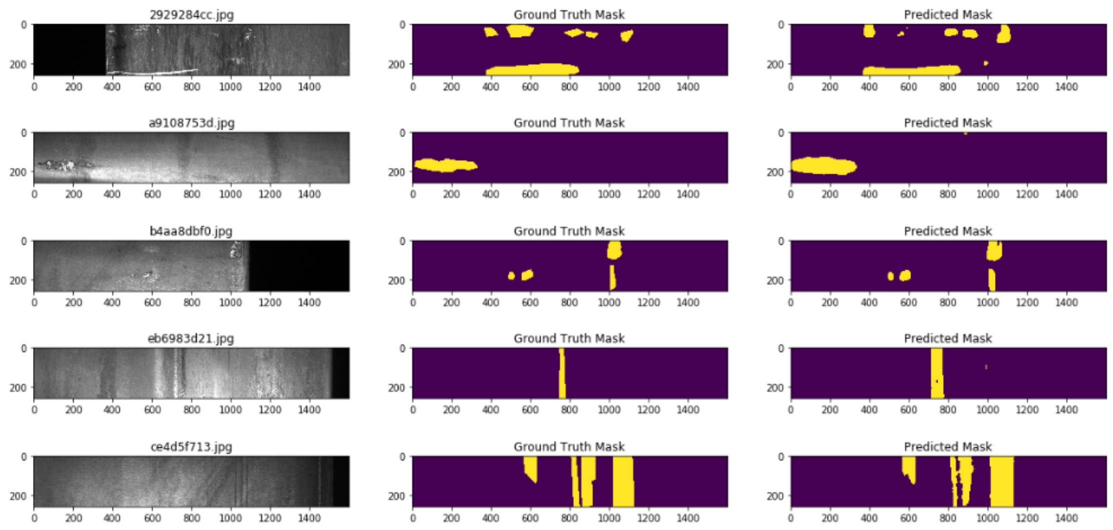
- Dice coefficient vs epoch plot for training the segmentation model on defect 2



Segmentation model Training: Defect 3

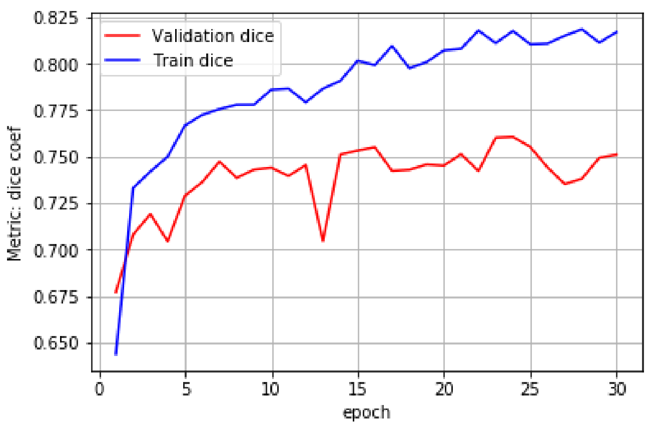
```
Test set evaluation score:
```

dice_loss	0.301452
dice_coef	0.698548



Test set

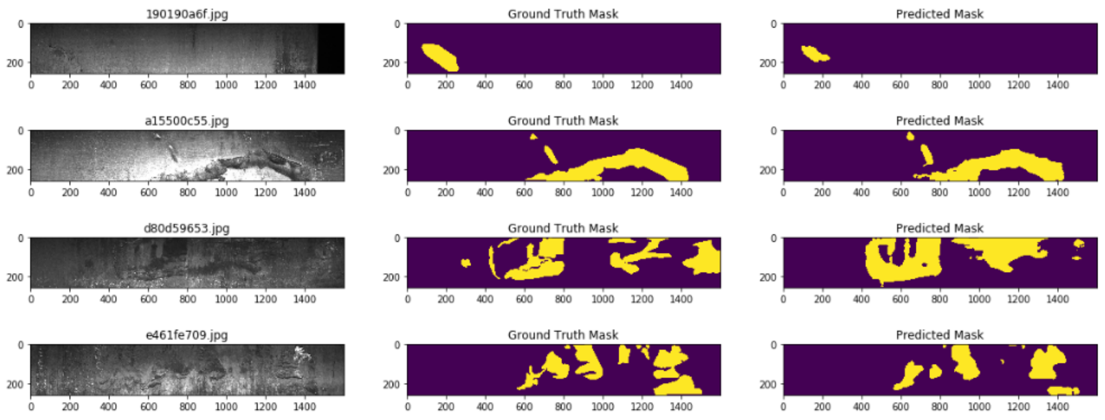
5.3.4 Defect Label 4:



Segmentation model Training: Defect 4

```
Test set evaluation score:
```

dice_loss	0.21178
dice_coef	0.78822



Test set

Performance of the above trained models:

Binary Classifier:

Dataset	binary_crossentropy	acc	f1_score_m	precision_m	recall_m
X_train	0.202241	0.923630	0.921999	0.949316	0.905966
X_val	0.240638	0.912064	0.912423	0.937087	0.898664
X_test	0.194755	0.926810	0.921435	0.955327	0.902135

Multi Label Classifier:

Dataset	binary_crossentropy	acc	f1_score_m	precision_m	recall_m
X_train	0.081054	0.968118	0.940510	0.945815	0.937232
X_val	0.092119	0.962500	0.929417	0.929264	0.931588
X_test	0.094178	0.965517	0.936398	0.941134	0.933854

Segmentation models: Dice Coefficient:

Dataset	Defect 1 model	Defect 2 model	Defect 3 model	Defect 4 model
X_train	0.714258	0.766948	0.735519	0.821943

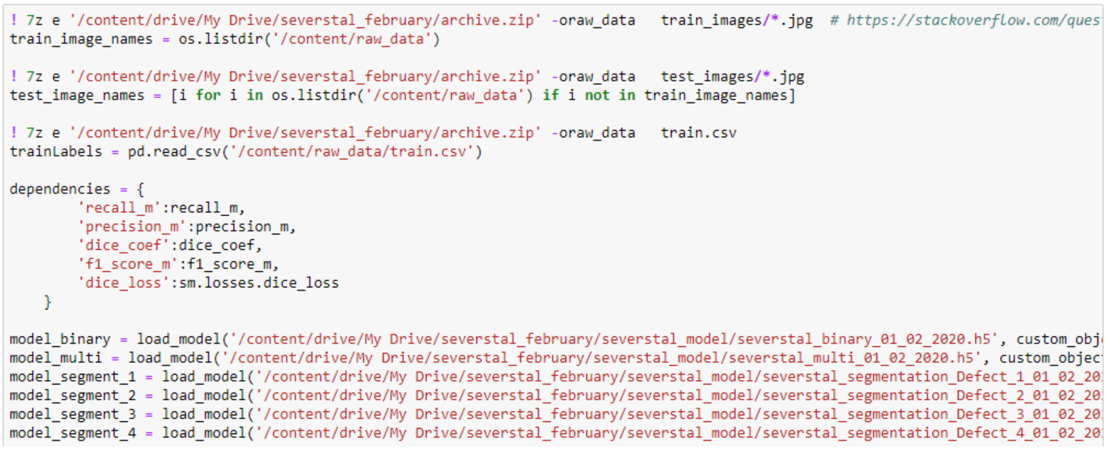
X_val	0.665121	0.678812	0.709641	0.76066
X_test	0.611203	0.655394	0.698548	0.78822

> *Well, the training of the models was easy. Let's see their prediction capability.*

6. Inference:

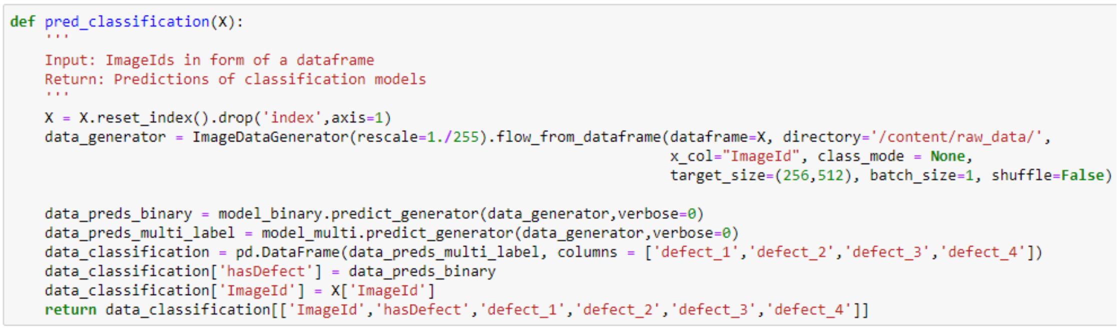
- Best models, from the training above, are saved to make inferences on images.

Loading Data and saved Models



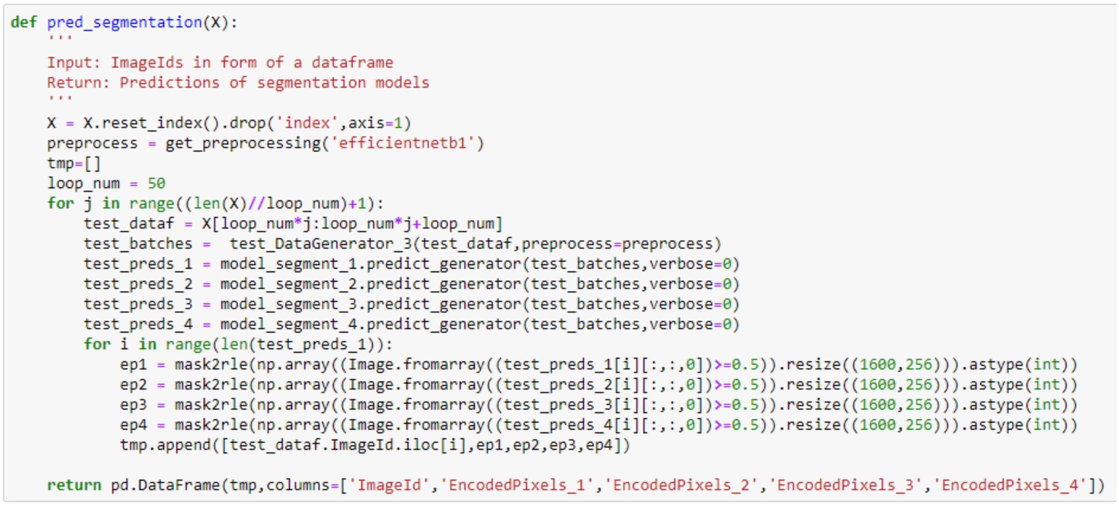
```
! 7z e '/content/drive/My Drive/severstal_february/archive.zip' -oraw_data   train_images/*.jpg  # https://stackoverflow.com/ques
train_image_names = os.listdir('/content/raw_data')

! 7z e '/content/drive/My Drive/severstal_february/archive.zip' -oraw_data   test_images/*.jpg
test_image_names = [i for i in os.listdir('/content/raw_data') if i not in train_image_names]

! 7z e '/content/drive/My Drive/severstal_february/archive.zip' -oraw_data   train.csv
trainLabels = pd.read_csv('/content/raw_data/train.csv')

dependencies = {
        'recall_m':recall_m,
        'precision_m':precision_m,
        'dice_coef':dice_coef,
        'f1_score_m':f1_score_m,
        'dice_loss':sm.losses.dice_loss
    }

model_binary = load_model('/content/drive/My Drive/severstal_february/severstal_model/severstal_binary_01_02_2020.h5', custom_obj
model_multi = load_model('/content/drive/My Drive/severstal_february/severstal_model/severstal_multi_01_02_2020.h5', custom_objec
model_segment_1 = load_model('/content/drive/My Drive/severstal_february/severstal_model/severstal_segmentation_Defect_1_01_02_20
model_segment_2 = load_model('/content/drive/My Drive/severstal_february/severstal_model/severstal_segmentation_Defect_2_01_02_20
model_segment_3 = load_model('/content/drive/My Drive/severstal_february/severstal_model/severstal_segmentation_Defect_3_01_02_20
model_segment_4 = load_model('/content/drive/My Drive/severstal_february/severstal_model/severstal_segmentation_Defect_4_01_02_20
```

Defining dependencies for loading saved models is important while using custom metrics



```
def pred_classification(X):
    '''
    Input: ImageIds in form of a dataframe
    Return: Predictions of classification models
    '''
    X = X.reset_index().drop('index',axis=1)
    data_generator = ImageDataGenerator(rescale=1./255).flow_from_dataframe(dataframe=X, directory='/content/raw_data/',
                                                                            x_col="ImageId", class_mode = None,
                                                                            target_size=(256,512), batch_size=1, shuffle=False)

    data_preds_binary = model_binary.predict_generator(data_generator,verbose=0)
    data_preds_multi_label = model_multi.predict_generator(data_generator,verbose=0)
    data_classification = pd.DataFrame(data_preds_multi_label, columns = ['defect_1','defect_2','defect_3','defect_4'])
    data_classification['hasDefect'] = data_preds_binary
    data_classification['ImageId'] = X['ImageId']
    return data_classification[['ImageId','hasDefect','defect_1','defect_2','defect_3','defect_4']]
```

For generating predictions using Classifier models



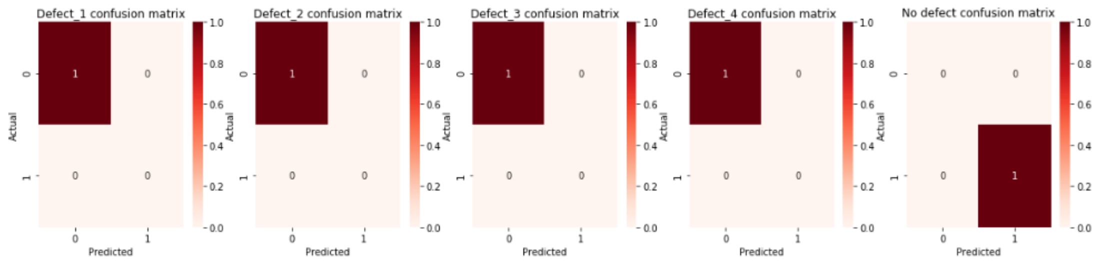
```
def pred_segmentation(X):
    '''
    Input: ImageIds in form of a dataframe
    Return: Predictions of segmentation models
    '''
    X = X.reset_index().drop('index',axis=1)
    preprocess = get_preprocessing('efficientnetb1')
    tmp=[]
    loop_num = 50
    for j in range((len(X)//loop_num)+1):
        test_dataf = X[loop_num*j:loop_num*j+loop_num]
        test_batches =  test_DataGenerator_3(test_dataf,preprocess=preprocess)
        test_preds_1 = model_segment_1.predict_generator(test_batches,verbose=0)
        test_preds_2 = model_segment_2.predict_generator(test_batches,verbose=0)
        test_preds_3 = model_segment_3.predict_generator(test_batches,verbose=0)
        test_preds_4 = model_segment_4.predict_generator(test_batches,verbose=0)
        for i in range(len(test_preds_1)):
            ep1 = mask2rle(np.array((Image.fromarray((test_preds_1[i][:,:,0])>=0.5)).resize((1600,256))).astype(int))
            ep2 = mask2rle(np.array((Image.fromarray((test_preds_2[i][:,:,0])>=0.5)).resize((1600,256))).astype(int))
            ep3 = mask2rle(np.array((Image.fromarray((test_preds_3[i][:,:,0])>=0.5)).resize((1600,256))).astype(int))
            ep4 = mask2rle(np.array((Image.fromarray((test_preds_4[i][:,:,0])>=0.5)).resize((1600,256))).astype(int))
            tmp.append([test_dataf.ImageId.iloc[i],ep1,ep2,ep3,ep4])

    return pd.DataFrame(tmp,columns=['ImageId','EncodedPixels_1','EncodedPixels_2','EncodedPixels_3','EncodedPixels_4'])
```

For generating predictions using Segmentation models

- Area thresholds and Classification thresholds are applied to the predictions of the models.
- **steel_prediction()** and **steel_evaluation()** are final functions defined for generating predictions and evaluations. [Code available in Github notebook]

Applying steel_evaluation():

```
Dice coefficient of each segmentation mask compared with true mask: [defect_1, defect_2, defect_3, defect_4]
[1.0, 1.0, 1.0, 1.0]
--------------------------------------------------
Classification Report:
              precision    recall  f1-score   support

 hasDefect_1       0.00      0.00      0.00         0
 hasDefect_2       0.00      0.00      0.00         0
 hasDefect_3       0.00      0.00      0.00         0
 hasDefect_4       0.00      0.00      0.00         0
    NoDefect       1.00      1.00      1.00         1

   micro avg       1.00      1.00      1.00         1
   macro avg       0.20      0.20      0.20         1
weighted avg       1.00      1.00      1.00         1
 samples avg       1.00      1.00      1.00         1

--------------------------------------------------
Confusion matrix:
```

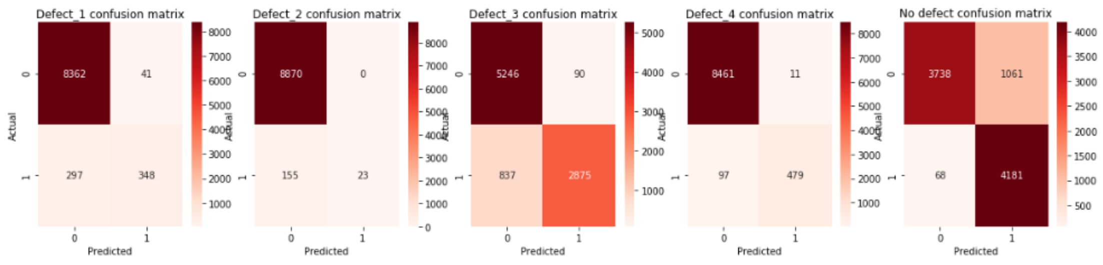


```
--------------------------------------------------
Mean Dice Coefficient: (Competition Metric)
1.0
```

Sample evaluation on a single image: This image has no defect and the models have perfectly detected that image has no defect.

6.1 Train set:

```
Mean Dice coefficient of each defect: [defect_1, defect_2, defect_3, defect_4]
[0.9524329106288681, 0.9823117540130416, 0.8203102531465517, 0.9787975158344384]
--------------------------------------------------
Classification Report:
              precision    recall  f1-score   support

 hasDefect_1       0.89      0.54      0.67       645
 hasDefect_2       1.00      0.13      0.23       178
 hasDefect_3       0.97      0.77      0.86      3712
 hasDefect_4       0.98      0.83      0.90       576
    NoDefect       0.80      0.98      0.88      4249

   micro avg       0.87      0.84      0.86      9360
   macro avg       0.93      0.65      0.71      9360
weighted avg       0.89      0.84      0.85      9360
 samples avg       0.87      0.86      0.86      9360

--------------------------------------------------
Confusion matrix:
```



```
--------------------------------------------------
Mean Dice Coefficient (Overall): (Competition Metric)
0.933463108405725
```

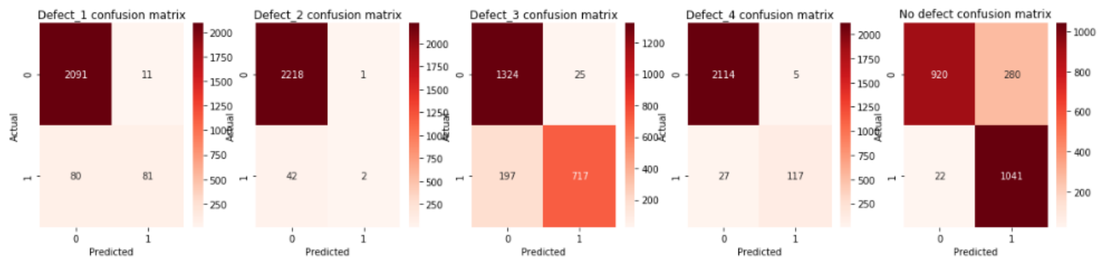
Evaluations on X_Train set

> ***Note:*** *If we want to move one FN to TP, more than one TN become FPs due to high imbalance in the dataset. The contribution of reduction of FP is higher than the contribution of reduction of FN in the final competition metric (Mean Dice Coefficient).*

- **Classification Report:** The model has tried to generate high precision for multilabel classification and high recall for binary classification tasks. It is important to have less False positives overall.
- **Confusion Matrix: Observation:** The model can be seen to have some confusions. It is evident that the model has tried its best to reduce False Positives.

6.2 Validation set:

```
Mean Dice coefficient of each defect: [defect_1, defect_2, defect_3, defect_4]
[0.947992110658418, 0.9806248055236414, 0.816231618882015, 0.973803152841361]
--------------------------------------------------
Classification Report:
              precision    recall  f1-score   support

 hasDefect_1       0.88      0.50      0.64       161
 hasDefect_2       0.67      0.05      0.09        44
 hasDefect_3       0.97      0.78      0.87       914
 hasDefect_4       0.96      0.81      0.88       144
    NoDefect       0.79      0.98      0.87      1063

   micro avg       0.86      0.84      0.85      2326
   macro avg       0.85      0.62      0.67      2326
weighted avg       0.87      0.84      0.84      2326
 samples avg       0.86      0.85      0.85      2326

--------------------------------------------------
Confusion matrix:
```

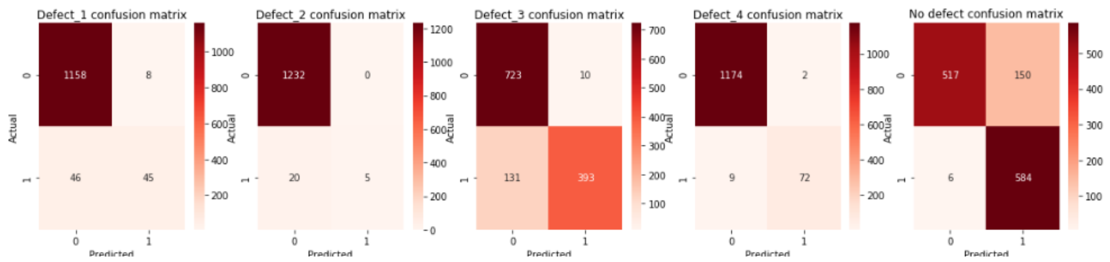


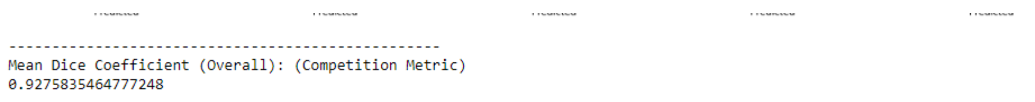
```
--------------------------------------------------
Mean Dice Coefficient (Overall): (Competition Metric)
0.9296629219763589
```

Evaluations on X_Val set

6.3 Test set:

```
Mean Dice coefficient of each defect: [defect_1, defect_2, defect_3, defect_4]
[0.9442220594192524, 0.9826448541766108, 0.8047021536276849, 0.9787651186873508]
--------------------------------------------------
Classification Report:
              precision    recall  f1-score   support

 hasDefect_1       0.85      0.49      0.62        91
 hasDefect_2       1.00      0.20      0.33        25
 hasDefect_3       0.98      0.75      0.85       524
 hasDefect_4       0.97      0.89      0.93        81
    NoDefect       0.80      0.99      0.88       590

   micro avg       0.87      0.84      0.85      1311
   macro avg       0.92      0.66      0.72      1311
weighted avg       0.89      0.84      0.84      1311
 samples avg       0.87      0.85      0.86      1311

--------------------------------------------------
Confusion matrix:
```

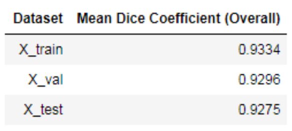




```
--------------------------------------------------
Mean Dice Coefficient (Overall): (Competition Metric)
0.9275835464777248
```

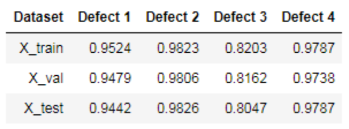
Evaluations on X_Test set

Mean Dice Coefficient: (Competition Metric)



Dataset	Mean Dice Coefficient (Overall)
X_train	0.9334
X_val	0.9296
X_test	0.9275

Mean Dice Coefficient (defect wise):



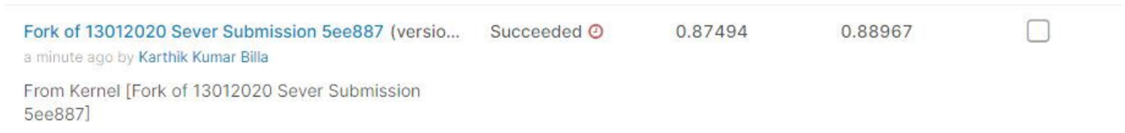
Dataset	Defect 1	Defect 2	Defect 3	Defect 4
X_train	0.9524	0.9823	0.8203	0.9787
X_val	0.9479	0.9806	0.8162	0.9738
X_test	0.9442	0.9826	0.8047	0.9787

Each metric can be compared to see that the model did not overfit on train set, validated well on validation dataset and generalizing well on unseen test set.

7. Summary:

- Images and its masks (in form of EncodedPixels) are provided to train a Deep Learning Model to Detect and Classify defects in steel. (Multi-label Classification). The competition is hosted by Severstal on Kaggle.
- Exploratory Data Analysis revealed that the dataset is imbalanced. A new feature 'area' is created to clip predictions with segmentation areas within a determined range. Different classes are observed to overlap on smaller values of area feature. This makes class separation not possible based solely on 'area' feature. It was observed that most of the images either contain one defect or do not have a defect.
- 6 model architecture is generated to train and test on this dataset. One binary classifier, One Multi-Label Classifier and Four segmentation models are used for the task.
- Image data contains minimal preprocessing. Pixel value scaling and Image augmentations for Model training are achieved using DataGenerators.
- Minority class priority based stratified sampling is performed on the dataset to split train set into train and validation sets.
- Pre-trained Deep Learning models are used: Xception architecture for Classification and legendary Unet architecture with efficientnetb1 backbone trained on ImageNet dataset for Segmentation.
- Tenosorboard is utilized for saving logs and visualizing model performance at each epoch. It has been observed that the models have satisfactory performance on defined metrics. It can also be deduced that a certain degree of confusion exists in both classification and segmentation models as the defect detection and localization are not perfect.

8. Kaggle Screenshot (Best submission):


Fork of 13012020 Sever Submission 5ee887 (versio... Succeeded 0.87494 0.88967
a minute ago by Karthik Kumar Billa
From Kernel [Fork of 13012020 Sever Submission 5ee887]

Due to limited compute, I have stopped working further on this dataset.

9. Future Work:

Higher compute will allow us to include a larger Batch size for training all the models(increasing from 8 to 16 or 32).

- A single strong model (possible to define easily with Pytorch version of segmentation_models library) can improve the performance a lot. Multiple models have this performance multiplier effect which reduces overall performance (<1 x <1 x … =<<1).
- Different architectures can be experimented such as combining the Binary and Multi-label Classifier into a Single Classifier model.
- Improving the quality of training data fed into the Neural Networks defines the performance. Techniques such as Test Time Augmentations can be experimented while Defect region blackouts can be used to increase number of training images(converting regions of defects to black pixel intensities converts defect present images to no defect image). Resolution of the output from ImageDataGenerators can be varied.

10. References:

- Data Source: https://www.kaggle.com/c/severstal-steel-defect-detection/data
- For Classification: Xception: https://keras.io/applications/#xception
- For Segmentation: Unet — EfficientNetB1: https://github.com/qubvel/segmentation_models
- Training and predictions platform: Google Colab https://colab.research.google.com/
- Course: https://www.appliedaicourse.com/course/11/Applied-Machine-learning-course
- Utility functions: Run Length Encodings, metrics

Github: https://github.com/rook0falcon

LinkedIn: https://www.linkedin.com/in/karthik-kumar-billa/

Steel Defect Image Segmentation Keras