VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SEVERSTAL: STEEL DEFECT DETECTION

MACHINE LEARNING (CSE4020)

PROJECT

By,

Shantanu 17BCE1161

Submitted To,

Prof. Nayeemulla Khan

November 2019

PROJECT DESCRIPTION

(From Kaggle)

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. To help make production of steel more efficient, this competition will help identify defects.

Severstal is leading the charge in efficient steel mining and production. They believe the future of metallurgy requires development across the economic, ecological, and social aspects of the industry—and they take corporate responsibility seriously. The company recently created the country's largest industrial data lake, with petabytes of data that were previously discarded. 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.

In this competition, you'll help engineers improve the algorithm by localizing and classifying surface defects on a steel sheet. If successful, you'll help keep manufacturing standards for steel high and enable Severstal to continue their innovation, leading to a stronger, more efficient world all around us.

EVALUATION OF THE PROJECT

This project is evaluated on 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. 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.

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.

File Format

Your submission file should be in csv format, with a header and columns names: ImageId_ClassId, EncodedPixels. Each row in your submission represents a single predicted defect segmentation for the given ImageId, and predicted ClassId, and you should have the same number of rows as num_images * num_defect_classes. The segment for each defect class will be encoded into a single row, even if there are several non-contiguous defect locations on an image.

```
ImageId_ClassId,EncodedPixels
004f40c73.jpg_1,1 1
004f40c73.jpg_2,1 1
004f40c73.jpg_3,2 409599
etc...
```

PROJECT STATEMENT

The aim of this project is to predict the location and type of defects found in steel manufacturing using the images provided. The images are named with a unique ImageId, and our task is to segment each image and classify the defects in the test set.

Data Description

DataTrain_images.Zip: Zip file containing all train images(12568 unique)

Test_images.zip: Zip file containing all train images(1801 unique)

train.csv: containing Imageid and Encoded Pixels columns

submission.csv: test csv file containing Imageid and Encoded Pixels columns

Each Image may have no defects, a defect of a single class, or defects of multiple classes (ClassId = [1, 2, 3, 4]).

Objective

Given an image, our task is to classify the defect and locate the segmentation of the defect. For each image you must segment the defects if it belongs to each of the class (ClassId = [1, 2, 3, 4]).

Class Labels and mask Information

There are two tasks associated with this problem:

1. classification of image into 4 defected classes (ClassId = [1, 2, 3, 4]).
2. predict the location of defects found(segmentation)

PROPOSED STRATEGIES

What is pipelining in Machine Learning?

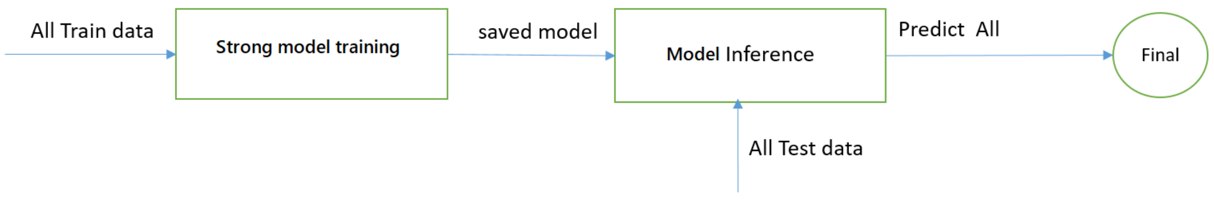
Machine learning pipelines are cyclical and iterative as every step is repeated to continuously improve the accuracy of the model and achieve a successful algorithm. To build better machine learning models, and get the most value from them, accessible, scalable and durable storage solutions are imperative, paving the way for on-premises object storage.

A machine learning pipeline is used to help automate machine learning workflows. They operate by enabling a sequence of data to be transformed and correlated together in a model that can be tested and evaluated to achieve an outcome, whether positive or negative.

As this problem deals with binary classification, multi label classification and regression, there are many approaches to solve this problem. Some of the various pipeline strategies are as follows:

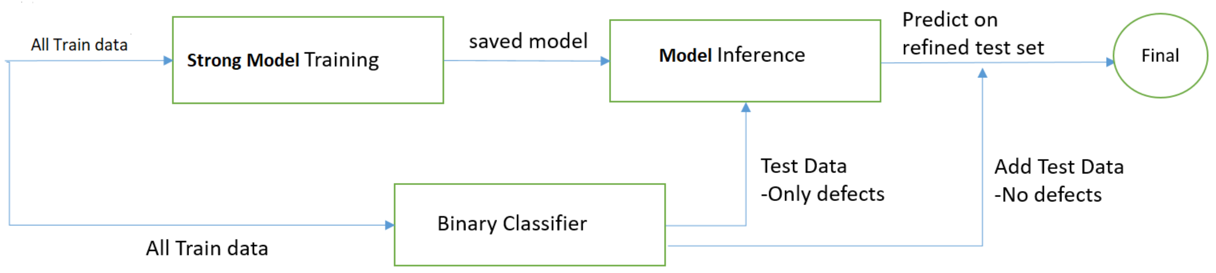
- ### Pipeline-1: Basic pipeline using segmentation models

  First and most efficient strategy is using a segmentation model, segmentation models used because they partition an image into regions and can classify objects in an image.



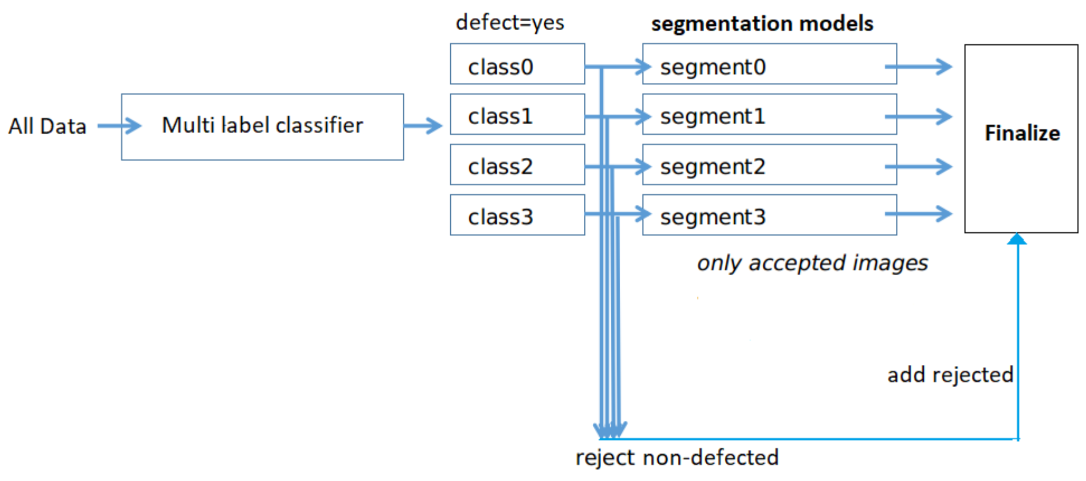
This is the straightforward pipeline, where all the input data is trained using a strong training model(most probably pre-trained).the output layer of the model is CNN(4),where for each input image we get four masks(images) with segmentation of that class, i.e. class([1,2,3,4]).

- **Pipeline-2: Segmentation using binary Classification**



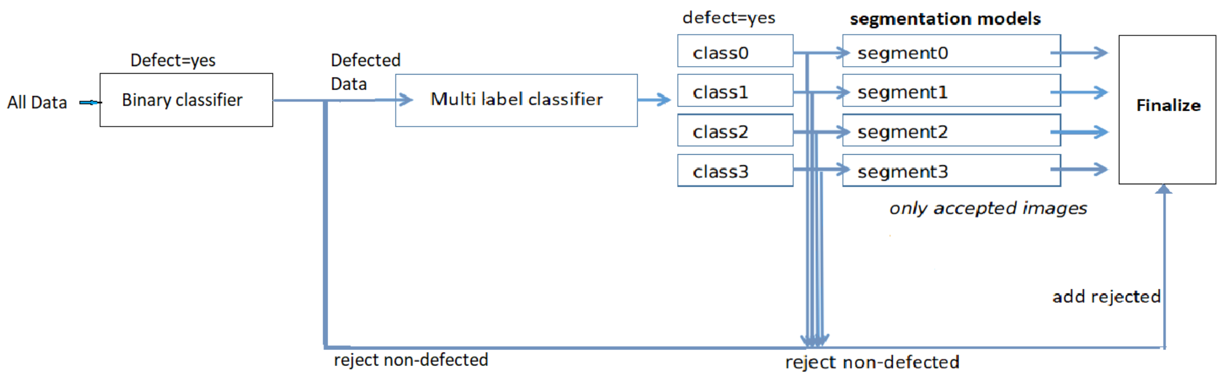
This strategy deals with building a segmentation model along with binary classification, a strong segmentation model(pre-trained) is built upon all training data and save model using model.save(model_name.h5). also save binary classification model similarly, while coming to test data, filter the defected images using pre-trained binary classifier and send only defected images to the segmentation model, in this way we can test the results faster as we are saving our model and also filtering defected images from the original data.

- **Pipeline-3: Segmentation using Multi label Classification**



This is a similar strategy to the previous one, instead of filtering defected images, we are using a pre-trained multi label classification model to get images which are defected, as this is a multi-label classifier one image can belong to more than one class. we directly take the output from the multi label classification pass to four segmentation models for class ([1,2,3,4]), here each model is a pre-trained segmentation model with respective defect class ([1,2,3,4]).

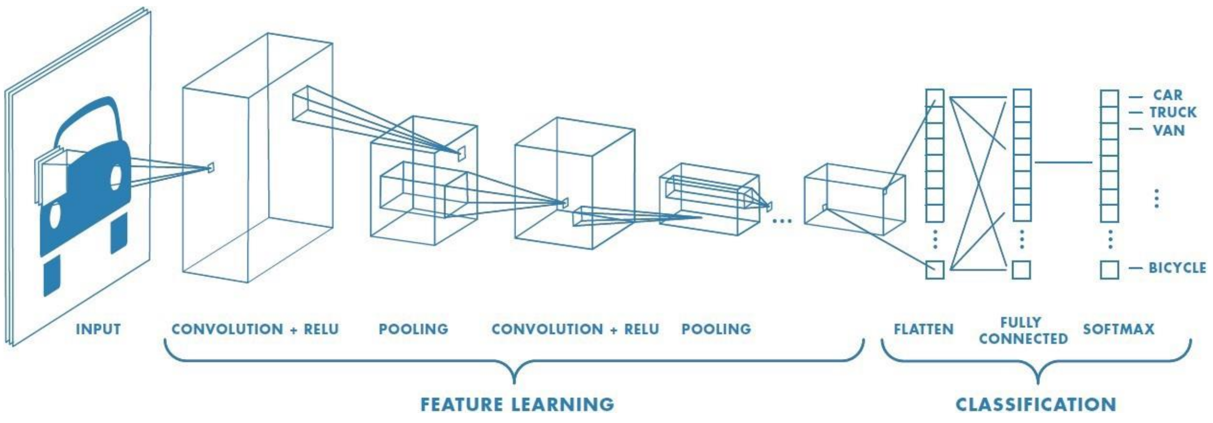
- **Pipeline-4: Segmentation model using Binary and Multi label Classification**



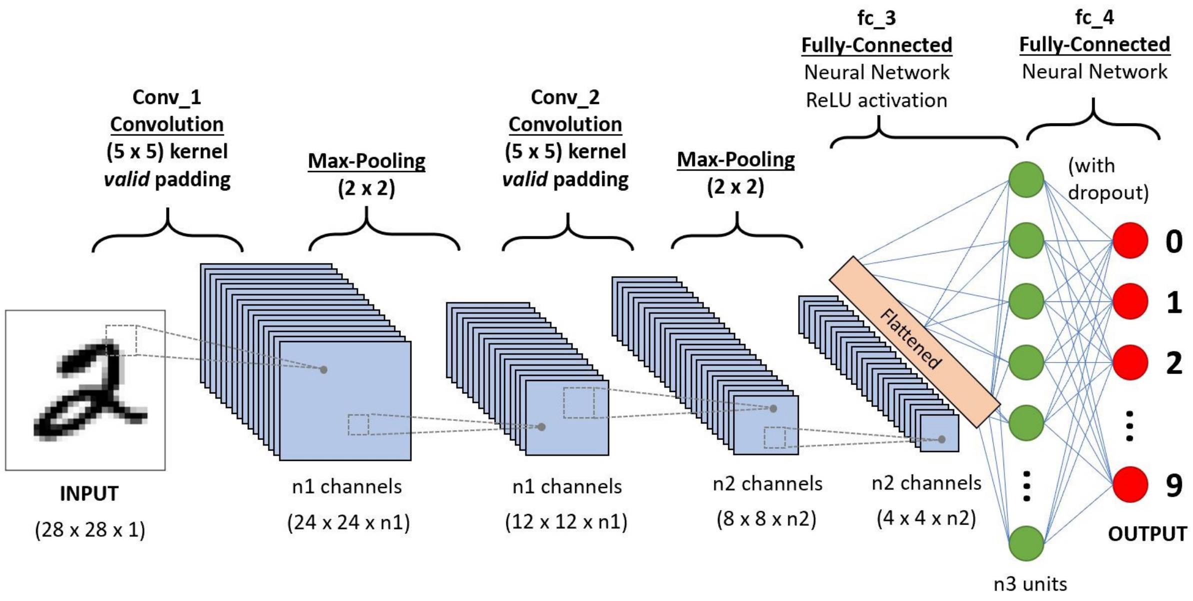
It is combination of pipeline-3 and pipeline-4. This can take less time to test results because we are filtering out the non-defective images and sending only defective images to segmentation model.

In this project, the Pipeline-2 (Segmentation using Binary Classification) is implemented. Here, we discard all of the training data that have all 4 masks missing. We only train on images that have at least 1 mask, so that our classifier does not overfit on empty masks. For that, we create a lightweight CNN to predict if a certain image has no defect (i.e., it has 4 missing masks). This gives us the pre-trained model that we use in the binary classification stage.

CONVOLUTIONAL NEURAL NETWORKS

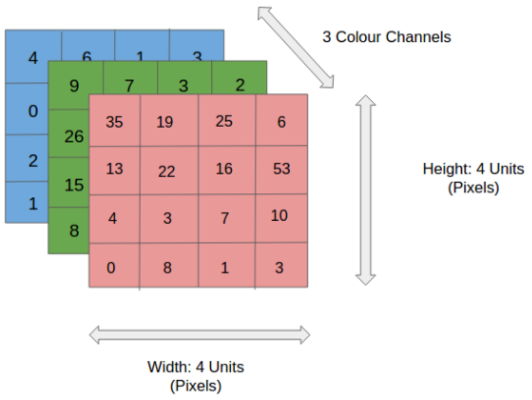


A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.



The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.

Input Image



In the figure, we have an RGB image which has been separated by its three colour planes — Red, Green, and Blue. There are a number of such colour spaces in which images exist — Grayscale, RGB, HSV, CMYK, etc.

You can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320). The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.

Convolution Layer — The Kernel

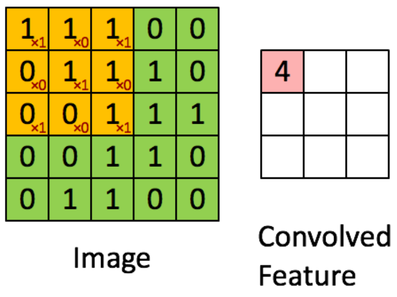


Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, e.g. RGB)

In the above demonstration, the green section resembles our 5x5x1 input image, I. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K, represented in the colour yellow. We have selected K as a 3x3x1 matrix.

The Kernel shifts 9 times because of Stride Length = 1 (Non-Strided), every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering.

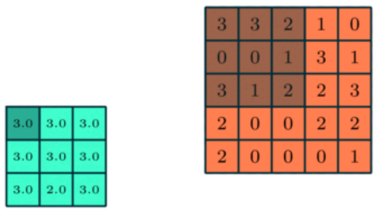
The objective of the Convolution Operation is to **extract the high-level features** such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network, which has the wholesome understanding of images in the dataset, similar to how we would.

There are two types of results to the operation — one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying **Valid Padding** in case of the former, or **Same Padding** in the case of the latter.

When we augment the 5x5x1 image into a 6x6x1 image and then apply the 3x3x1 kernel over it, we find that the convolved matrix turns out to be of dimensions 5x5x1. Hence the name — **Same Padding**.

On the other hand, if we perform the same operation without padding, we are presented with a matrix which has dimensions of the Kernel (3x3x1) itself — **Valid Padding**.

Pooling Layer



Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to **decrease the computational power required to process the data** through dimensionality reduction. Furthermore, it is useful for **extracting dominant features** which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

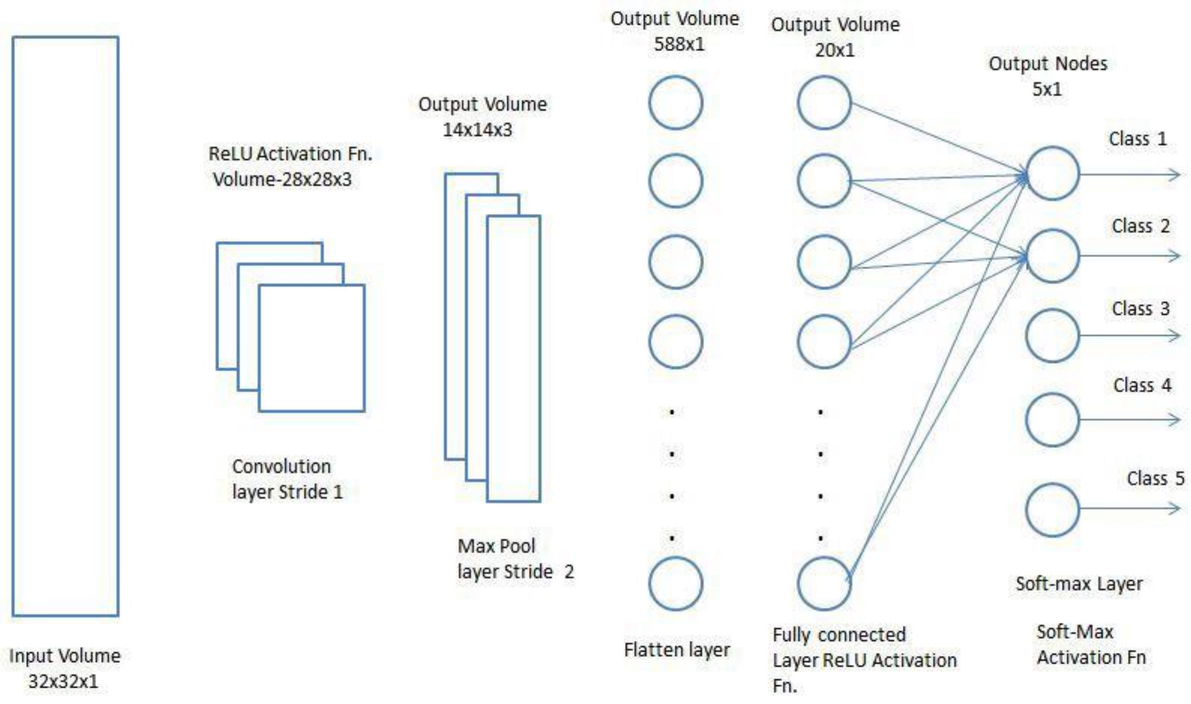
There are two types of Pooling: Max Pooling and Average Pooling. **Max Pooling** returns the **maximum value** from the portion of the image covered by the Kernel. On the other hand, **Average Pooling** returns the **average of all the values** from the portion of the image covered by the Kernel.

Max Pooling also performs as a **Noise Suppressant**. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that **Max Pooling performs a lot better than Average Pooling**.

Classification — Fully Connected Layer (FC Layer)

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the **SoftMax Classification** technique.



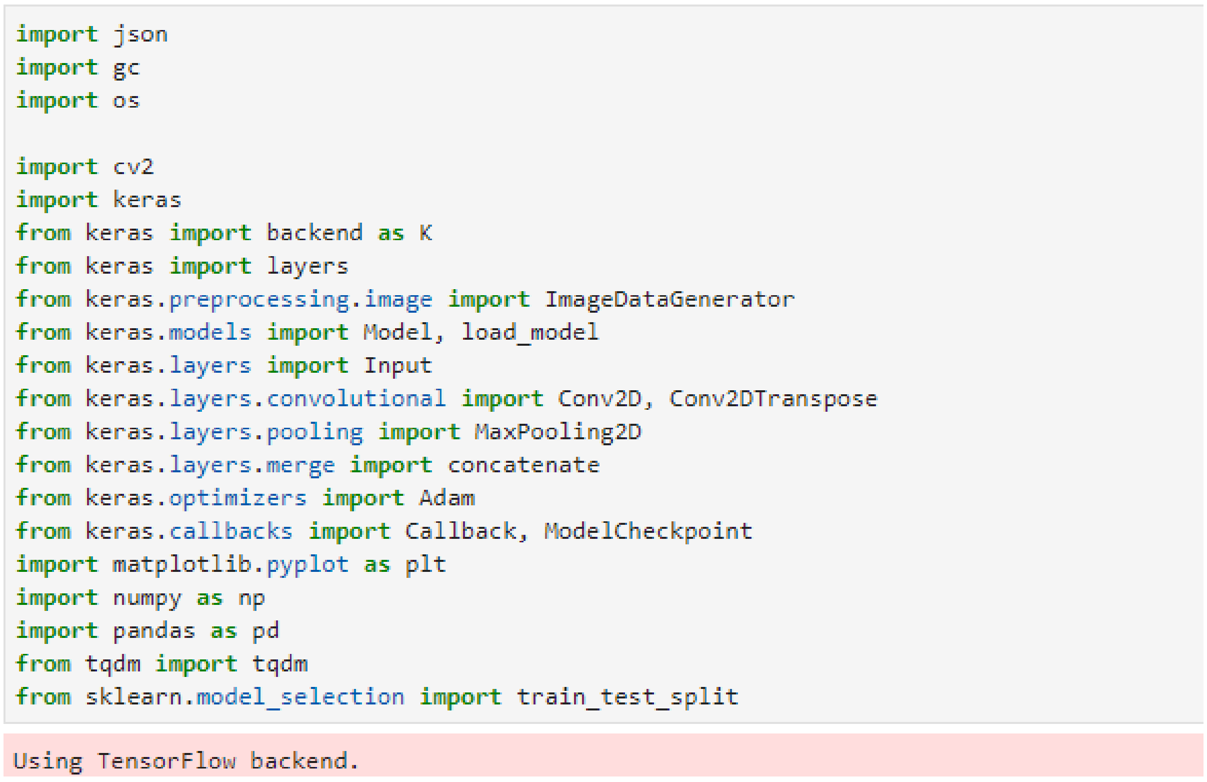
There are various architectures of CNNs available which have been key in building algorithms which power and shall power AI as a whole in the foreseeable future. Some of them have been listed below:

1. LeNet
2. AlexNet
3. VGGNet
4. GoogLeNet
5. ResNet
6. ZFNet

IMPLEMENTATION

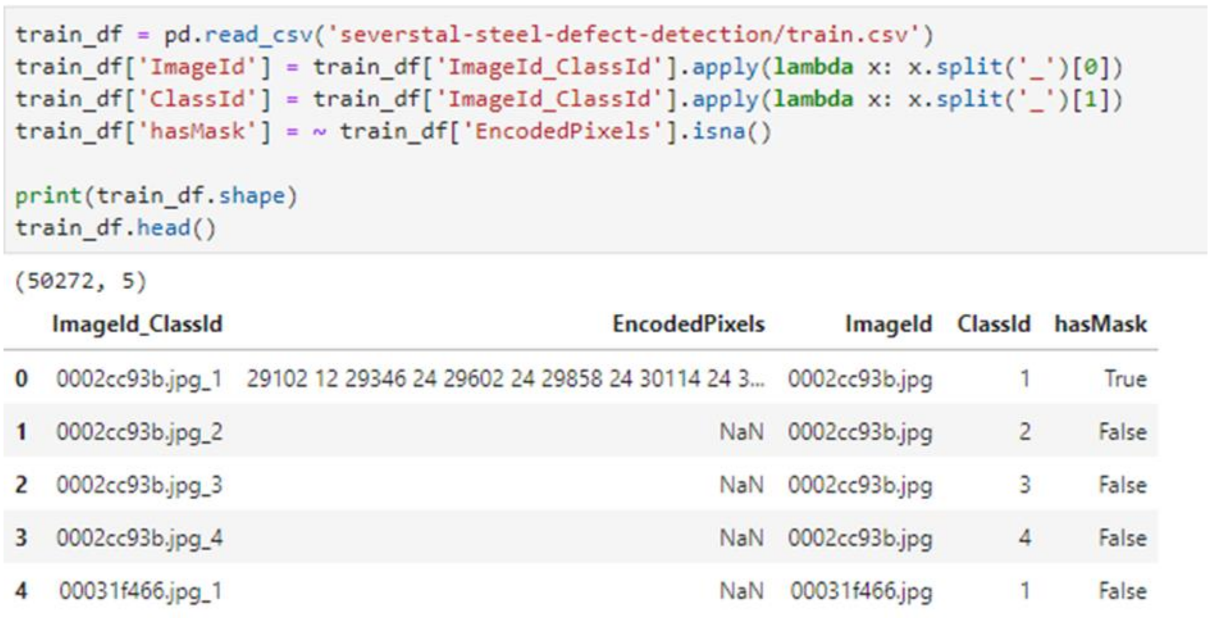
Pre-processing:

Discard all of the training data that have all 4 masks missing. We will only train on images that have at least 1 mask.



```python
import json
import gc
import os

import cv2
import keras
from keras import backend as K
from keras import layers
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Model, load_model
from keras.layers import Input
from keras.layers.convolutional import Conv2D, Conv2DTranspose
from keras.layers.pooling import MaxPooling2D
from keras.layers.merge import concatenate
from keras.optimizers import Adam
from keras.callbacks import Callback, ModelCheckpoint
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tqdm import tqdm
from sklearn.model_selection import train_test_split
```

```
Using TensorFlow backend.
```



```python
train_df = pd.read_csv('severstal-steel-defect-detection/train.csv')
train_df['ImageId'] = train_df['ImageId_ClassId'].apply(lambda x: x.split('_')[0])
train_df['ClassId'] = train_df['ImageId_ClassId'].apply(lambda x: x.split('_')[1])
train_df['hasMask'] = ~ train_df['EncodedPixels'].isna()

print(train_df.shape)
train_df.head()
```

```
(50272, 5)
```

	ImageId_ClassId	EncodedPixels	ImageId	ClassId	hasMask
0	0002cc93b.jpg_1	29102 12 29346 24 29602 24 29858 24 30114 24 3...	0002cc93b.jpg	1	True
1	0002cc93b.jpg_2	NaN	0002cc93b.jpg	2	False
2	0002cc93b.jpg_3	NaN	0002cc93b.jpg	3	False
3	0002cc93b.jpg_4	NaN	0002cc93b.jpg	4	False
4	00031f466.jpg_1	NaN	00031f466.jpg	1	False

```python
mask_count_df = train_df.groupby('ImageId').agg(np.sum).reset_index()
mask_count_df.sort_values('hasMask', ascending=False, inplace=True)
print(mask_count_df.shape)
mask_count_df.head()
```

```
(12568, 2)
```

	ImageId	hasMask
10803	db4867ee8.jpg	3.0
11776	ef24da2ba.jpg	3.0
6284	7f30b9c64.jpg	2.0
9421	bf0c81db6.jpg	2.0
9615	c314f43f3.jpg	2.0

```python
sub_df = pd.read_csv('severstal-steel-defect-detection/sample_submission.csv')
sub_df['ImageId'] = sub_df['ImageId_ClassId'].apply(lambda x: x.split('_')[0])
test_imgs = pd.DataFrame(sub_df['ImageId'].unique(), columns=['ImageId'])
test_imgs.head()
```

	ImageId
0	004f40c73.jpg
1	006f39c41.jpg
2	00b7fb703.jpg
3	00bbcd9af.jpg
4	0108ce457.jpg

```python
non_missing_train_idx = mask_count_df[mask_count_df['hasMask'] > 0]
non_missing_train_idx.head()
```

	ImageId	hasMask
10803	db4867ee8.jpg	3.0
11776	ef24da2ba.jpg	3.0
6284	7f30b9c64.jpg	2.0
9421	bf0c81db6.jpg	2.0
9615	c314f43f3.jpg	2.0

Discard Images

Use the DenseNet classifier trained to predict all of the test images that will have all 4 masks missing. We will automatically set the RLEs of those images to null.

```
def load_img(code, base, resize=True):
    path = f'{base}/{code}'
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    if resize:
        img = cv2.resize(img, (256, 256))

    return img

def validate_path(path):
    if not os.path.exists(path):
        os.makedirs(path)
```

```
BATCH_SIZE = 64
def create_test_gen():
    return ImageDataGenerator(rescale=1/255.).flow_from_dataframe(
        test_imgs,
        directory='severstal-steel-defect-detection/test_images',
        x_col='ImageId',
        class_mode=None,
        target_size=(256, 256),
        batch_size=BATCH_SIZE,
        shuffle=False
    )

test_gen = create_test_gen()
```

```
Found 1801 validated image filenames.
```

```
remove_model = load_model('severstal-steel-defect-detection/model.h5')
remove_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
densenet121 (Model)          (None, 8, 8, 1024)        7037504   
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1024)              0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 1024)              4096      
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0         
```

```
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
batch_normalization_2 (Batch (None, 512)               2048
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 7,568,961
Trainable params: 7,482,241
Non-trainable params: 86,720
_________________________________________________________________
```

```
test_missing_pred = remove_model.predict_generator(
    test_gen,
    steps=len(test_gen),
    verbose=1
)

test_imgs['allMissing'] = test_missing_pred
test_imgs.head()
```

```
29/29 [==============================] - ETA: 1:09:5 - ETA: 43:01  - ETA: 33:4 - ETA: 28:3 - ETA: 25:0 - ETA: 22:3 - ETA: 20:3 - ETA: 18:5 - ETA: 17:3 - ETA: 16:1 - ET
A: 15:1 - ETA: 14:0 - ETA: 13:0 - ETA: 12:1 - ETA: 11:1 - ETA: 10:2 - ETA: 9:3 - ETA: 8: - ETA: 7: - ETA: 7: - ETA: 6: - ETA: 5: - ETA: 4: - ETA: 3: - ETA: 3: - ETA:
2: - ETA: 1: - ETA: 45s - 1271s 44s/step
```

	ImageId	allMissing
0	004f40c73.jpg	0.535308
1	006f39c41.jpg	0.999956
2	00b7fb703.jpg	0.997788
3	00bbcd9af.jpg	0.000324
4	0108ce457.jpg	0.114875

```
filtered_mask = sub_df['ImageId'].isin(filtered_test_imgs["ImageId"].values)
filtered_sub_df = sub_df[filtered_mask].copy()
null_sub_df = sub_df[~filtered_mask].copy()
null_sub_df['EncodedPixels'] = null_sub_df['EncodedPixels'].apply(
    lambda x: ' ')

filtered_sub_df.reset_index(drop=True, inplace=True)
filtered_test_imgs.reset_index(drop=True, inplace=True)

print(filtered_sub_df.shape)
print(null_sub_df.shape)

filtered_sub_df.head()
```

```
filtered_test_imgs = test_imgs[test_imgs['allMissing'] < 0.5]
print(filtered_test_imgs.shape)
filtered_test_imgs.head()
```

```
(721, 2)
```

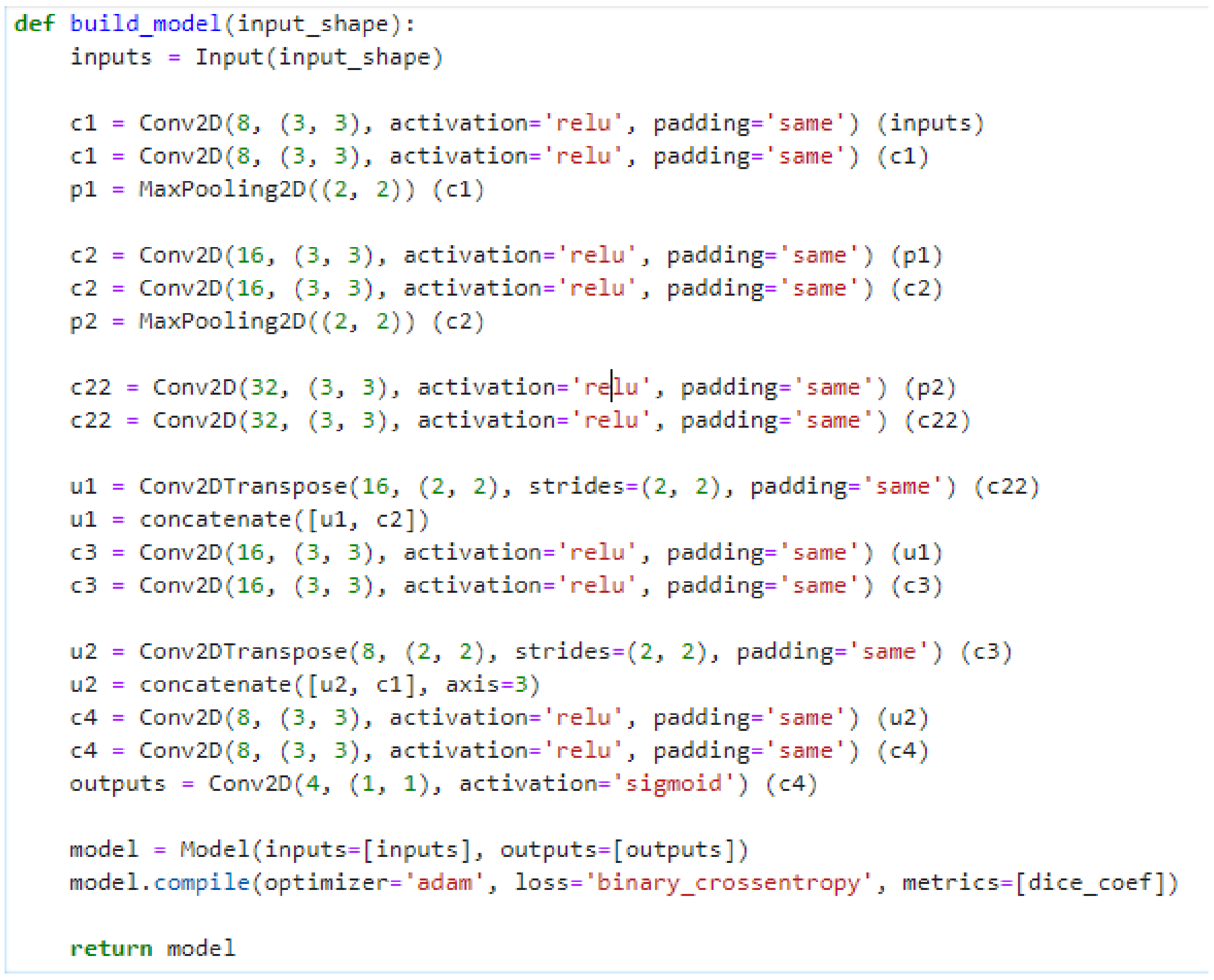
	ImageId	allMissing
3	00bbcd9af.jpg	0.000324
4	0108ce457.jpg	0.114875
6	010ec96b4.jpg	0.000624
8	017bd7ce3.jpg	0.042616
9	01b47d973.jpg	0.022378

Keras U-Net

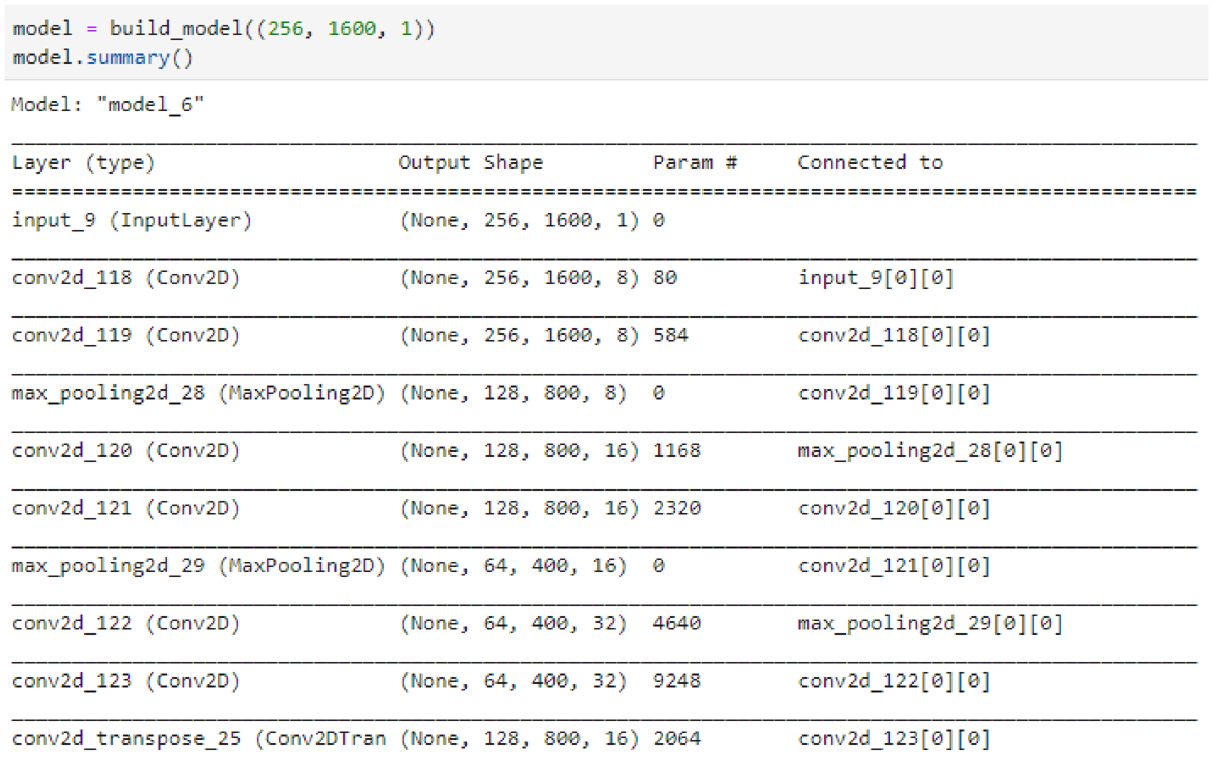
Train the CNN model on the "filtered" training data. Then, perform inference only on test images that were not discarded in step 1.

```python
BATCH_SIZE = 16

train_idx, val_idx = train_test_split(
    non_missing_train_idx.index,  # NOTICE DIFFERENCE
    random_state=2019,
    test_size=0.15
)

train_generator = DataGenerator(
    train_idx,
    df=mask_count_df,
    target_df=train_df,
    batch_size=BATCH_SIZE,
    n_classes=4
)

val_generator = DataGenerator(
    val_idx,
    df=mask_count_df,
    target_df=train_df,
    batch_size=BATCH_SIZE,
    n_classes=4
)
```

```python
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)
```



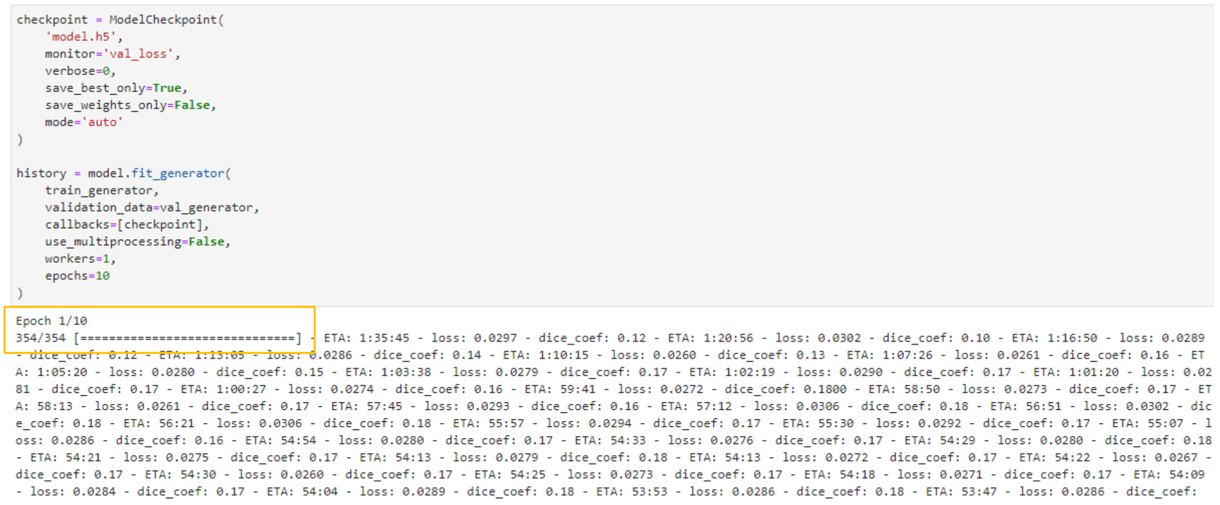
```python
def build_model(input_shape):
    inputs = Input(input_shape)

    c1 = Conv2D(8, (3, 3), activation='relu', padding='same') (inputs)
    c1 = Conv2D(8, (3, 3), activation='relu', padding='same') (c1)
    p1 = MaxPooling2D((2, 2)) (c1)

    c2 = Conv2D(16, (3, 3), activation='relu', padding='same') (p1)
    c2 = Conv2D(16, (3, 3), activation='relu', padding='same') (c2)
    p2 = MaxPooling2D((2, 2)) (c2)

    c22 = Conv2D(32, (3, 3), activation='relu', padding='same') (p2)
    c22 = Conv2D(32, (3, 3), activation='relu', padding='same') (c22)

    u1 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (c22)
    u1 = concatenate([u1, c2])
    c3 = Conv2D(16, (3, 3), activation='relu', padding='same') (u1)
    c3 = Conv2D(16, (3, 3), activation='relu', padding='same') (c3)

    u2 = Conv2DTranspose(8, (2, 2), strides=(2, 2), padding='same') (c3)
    u2 = concatenate([u2, c1], axis=3)
    c4 = Conv2D(8, (3, 3), activation='relu', padding='same') (u2)
    c4 = Conv2D(8, (3, 3), activation='relu', padding='same') (c4)
    outputs = Conv2D(4, (1, 1), activation='sigmoid') (c4)

    model = Model(inputs=[inputs], outputs=[outputs])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[dice_coef])

    return model
```



```python
model = build_model((256, 1600, 1))
model.summary()
```

```
Model: "model_6"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_9 (InputLayer)            (None, 256, 1600, 1) 0
__________________________________________________________________________________________________
conv2d_118 (Conv2D)             (None, 256, 1600, 8) 80          input_9[0][0]
__________________________________________________________________________________________________
conv2d_119 (Conv2D)             (None, 256, 1600, 8) 584         conv2d_118[0][0]
__________________________________________________________________________________________________
max_pooling2d_28 (MaxPooling2D) (None, 128, 800, 8)  0           conv2d_119[0][0]
__________________________________________________________________________________________________
conv2d_120 (Conv2D)             (None, 128, 800, 16) 1168        max_pooling2d_28[0][0]
__________________________________________________________________________________________________
conv2d_121 (Conv2D)             (None, 128, 800, 16) 2320        conv2d_120[0][0]
__________________________________________________________________________________________________
max_pooling2d_29 (MaxPooling2D) (None, 64, 400, 16)  0           conv2d_121[0][0]
__________________________________________________________________________________________________
conv2d_122 (Conv2D)             (None, 64, 400, 32)  4640        max_pooling2d_29[0][0]
__________________________________________________________________________________________________
conv2d_123 (Conv2D)             (None, 64, 400, 32)  9248        conv2d_122[0][0]
__________________________________________________________________________________________________
conv2d_transpose_25 (Conv2DTran (None, 128, 800, 16) 2064        conv2d_123[0][0]
____________________________________________________________________________________________________
concatenate_25 (Concatenate)    (None, 128, 800, 32) 0          conv2d_transpose_25[0][0]
                                                                 conv2d_121[0][0]
____________________________________________________________________________________________________
conv2d_124 (Conv2D)             (None, 128, 800, 16) 4624       concatenate_25[0][0]
____________________________________________________________________________________________________
conv2d_125 (Conv2D)             (None, 128, 800, 16) 2320       conv2d_124[0][0]
____________________________________________________________________________________________________
conv2d_transpose_26 (Conv2DTran (None, 256, 1600, 8) 520        conv2d_125[0][0]
____________________________________________________________________________________________________
concatenate_26 (Concatenate)    (None, 256, 1600, 16 0          conv2d_transpose_26[0][0]
                                                                 conv2d_119[0][0]
____________________________________________________________________________________________________
conv2d_126 (Conv2D)             (None, 256, 1600, 8) 1160       concatenate_26[0][0]
____________________________________________________________________________________________________
conv2d_127 (Conv2D)             (None, 256, 1600, 8) 584        conv2d_126[0][0]
____________________________________________________________________________________________________
conv2d_128 (Conv2D)             (None, 256, 1600, 4) 36         conv2d_127[0][0]
====================================================================================================
Total params: 29,348
Trainable params: 29,348
Non-trainable params: 0
____________________________________________________________________________________________________
```

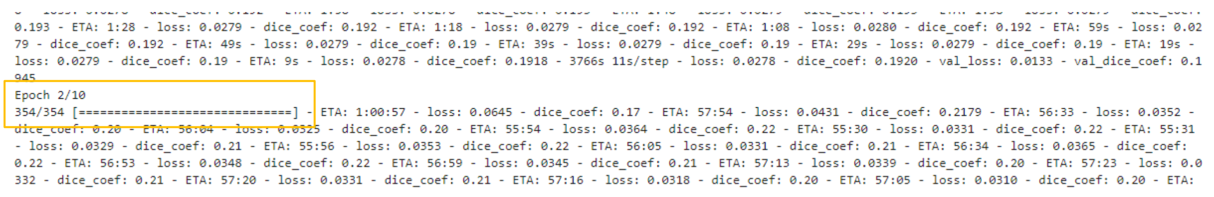
Epoch 1/10



```python
checkpoint = ModelCheckpoint(
    'model.h5',
    monitor='val_loss',
    verbose=0,
    save_best_only=True,
    save_weights_only=False,
    mode='auto'
)

history = model.fit_generator(
    train_generator,
    validation_data=val_generator,
    callbacks=[checkpoint],
    use_multiprocessing=False,
    workers=1,
    epochs=10
)
```

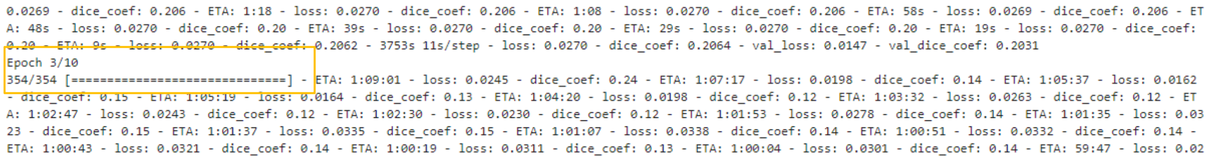
```
Epoch 1/10
354/354 [==============================] - ETA: 1:35:45 - loss: 0.0297 - dice_coef: 0.12 - ETA: 1:20:56 - loss: 0.0302 - dice_coef: 0.10 - ETA: 1:16:50 - loss: 0.0289
- dice_coef: 0.12 - ETA: 1:13:05 - loss: 0.0286 - dice_coef: 0.14 - ETA: 1:10:15 - loss: 0.0260 - dice_coef: 0.13 - ETA: 1:07:26 - loss: 0.0261 - dice_coef: 0.16 - ET
A: 1:05:20 - loss: 0.0280 - dice_coef: 0.15 - ETA: 1:03:38 - loss: 0.0279 - dice_coef: 0.17 - ETA: 1:02:19 - loss: 0.0290 - dice_coef: 0.17 - ETA: 1:01:20 - loss: 0.02
81 - dice_coef: 0.17 - ETA: 1:00:27 - loss: 0.0274 - dice_coef: 0.16 - ETA: 59:41 - loss: 0.0272 - dice_coef: 0.1800 - ETA: 58:50 - loss: 0.0273 - dice_coef: 0.17 - ET
A: 58:13 - loss: 0.0261 - dice_coef: 0.17 - ETA: 57:45 - loss: 0.0293 - dice_coef: 0.16 - ETA: 57:12 - loss: 0.0306 - dice_coef: 0.18 - ETA: 56:51 - loss: 0.0302 - dic
e_coef: 0.18 - ETA: 56:21 - loss: 0.0306 - dice_coef: 0.18 - ETA: 55:57 - loss: 0.0294 - dice_coef: 0.17 - ETA: 55:30 - loss: 0.0292 - dice_coef: 0.17 - ETA: 55:07 - l
oss: 0.0286 - dice_coef: 0.16 - ETA: 54:54 - loss: 0.0280 - dice_coef: 0.17 - ETA: 54:33 - loss: 0.0276 - dice_coef: 0.17 - ETA: 54:29 - loss: 0.0280 - dice_coef: 0.18
- ETA: 54:21 - loss: 0.0275 - dice_coef: 0.17 - ETA: 54:13 - loss: 0.0279 - dice_coef: 0.18 - ETA: 54:13 - loss: 0.0272 - dice_coef: 0.17 - ETA: 54:22 - loss: 0.0267 -
dice_coef: 0.17 - ETA: 54:30 - loss: 0.0260 - dice_coef: 0.17 - ETA: 54:25 - loss: 0.0273 - dice_coef: 0.17 - ETA: 54:18 - loss: 0.0271 - dice_coef: 0.17 - ETA: 54:09
- loss: 0.0284 - dice_coef: 0.17 - ETA: 54:04 - loss: 0.0289 - dice_coef: 0.18 - ETA: 53:53 - loss: 0.0286 - dice_coef: 0.18 - ETA: 53:47 - loss: 0.0286 - dice_coef:
```

Epoch 2/10



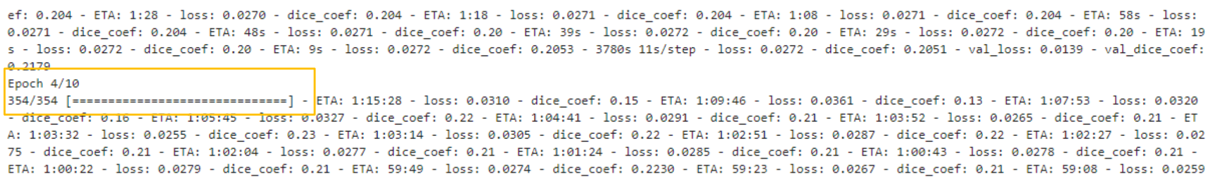
```
0.193 - ETA: 1:28 - loss: 0.0279 - dice_coef: 0.192 - ETA: 1:18 - loss: 0.0279 - dice_coef: 0.192 - ETA: 1:08 - loss: 0.0280 - dice_coef: 0.192 - ETA: 59s - loss: 0.02
79 - dice_coef: 0.192 - ETA: 49s - loss: 0.0279 - dice_coef: 0.19 - ETA: 39s - loss: 0.0279 - dice_coef: 0.19 - ETA: 29s - loss: 0.0279 - dice_coef: 0.19 - ETA: 19s -
loss: 0.0279 - dice_coef: 0.19 - ETA: 9s - loss: 0.0278 - dice_coef: 0.1918 - 3766s 11s/step - loss: 0.0278 - dice_coef: 0.1920 - val_loss: 0.0133 - val_dice_coef: 0.1
945
Epoch 2/10
354/354 [==============================] - ETA: 1:00:57 - loss: 0.0645 - dice_coef: 0.17 - ETA: 57:54 - loss: 0.0431 - dice_coef: 0.2179 - ETA: 56:33 - loss: 0.0352 -
dice_coef: 0.20 - ETA: 56:04 - loss: 0.0325 - dice_coef: 0.20 - ETA: 55:54 - loss: 0.0364 - dice_coef: 0.22 - ETA: 55:30 - loss: 0.0331 - dice_coef: 0.22 - ETA: 55:31
- loss: 0.0329 - dice_coef: 0.21 - ETA: 55:56 - loss: 0.0353 - dice_coef: 0.22 - ETA: 56:05 - loss: 0.0331 - dice_coef: 0.21 - ETA: 56:34 - loss: 0.0365 - dice_coef:
0.22 - ETA: 56:53 - loss: 0.0348 - dice_coef: 0.22 - ETA: 56:59 - loss: 0.0345 - dice_coef: 0.21 - ETA: 57:13 - loss: 0.0339 - dice_coef: 0.20 - ETA: 57:23 - loss: 0.0
332 - dice_coef: 0.21 - ETA: 57:20 - loss: 0.0331 - dice_coef: 0.21 - ETA: 57:16 - loss: 0.0318 - dice_coef: 0.20 - ETA: 57:05 - loss: 0.0310 - dice_coef: 0.20 - ETA:
```

Epoch 3/10



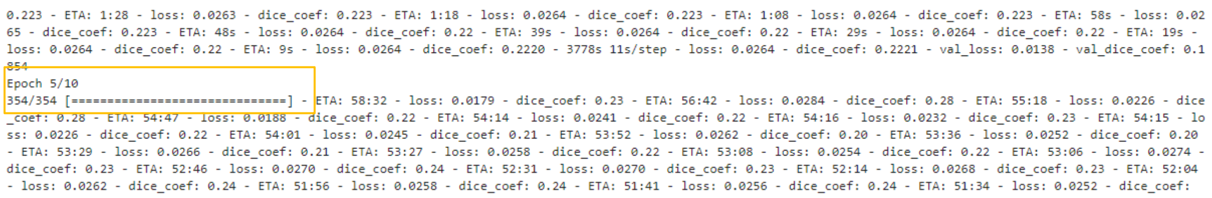
```
0.0269 - dice_coef: 0.206 - ETA: 1:18 - loss: 0.0270 - dice_coef: 0.206 - ETA: 1:08 - loss: 0.0270 - dice_coef: 0.206 - ETA: 58s - loss: 0.0269 - dice_coef: 0.206 - ET
A: 48s - loss: 0.0270 - dice_coef: 0.20 - ETA: 39s - loss: 0.0270 - dice_coef: 0.20 - ETA: 29s - loss: 0.0270 - dice_coef: 0.20 - ETA: 19s - loss: 0.0270 - dice_coef:
0.20 - ETA: 9s - loss: 0.0270 - dice_coef: 0.2062 - 3753s 11s/step - loss: 0.0270 - dice_coef: 0.2064 - val_loss: 0.0147 - val_dice_coef: 0.2031
Epoch 3/10
354/354 [==============================] - ETA: 1:09:01 - loss: 0.0245 - dice_coef: 0.24 - ETA: 1:07:17 - loss: 0.0198 - dice_coef: 0.14 - ETA: 1:05:37 - loss: 0.0162
- dice_coef: 0.15 - ETA: 1:05:19 - loss: 0.0164 - dice_coef: 0.13 - ETA: 1:04:20 - loss: 0.0198 - dice_coef: 0.12 - ETA: 1:03:32 - loss: 0.0263 - dice_coef: 0.12 - ET
A: 1:02:47 - loss: 0.0243 - dice_coef: 0.12 - ETA: 1:02:30 - loss: 0.0230 - dice_coef: 0.12 - ETA: 1:01:53 - loss: 0.0278 - dice_coef: 0.14 - ETA: 1:01:35 - loss: 0.03
23 - dice_coef: 0.15 - ETA: 1:01:37 - loss: 0.0335 - dice_coef: 0.15 - ETA: 1:01:07 - loss: 0.0338 - dice_coef: 0.14 - ETA: 1:00:51 - loss: 0.0332 - dice_coef: 0.14 -
ETA: 1:00:43 - loss: 0.0321 - dice_coef: 0.14 - ETA: 1:00:19 - loss: 0.0311 - dice_coef: 0.13 - ETA: 1:00:04 - loss: 0.0301 - dice_coef: 0.14 - ETA: 59:47 - loss: 0.02
```

Epoch 4/10



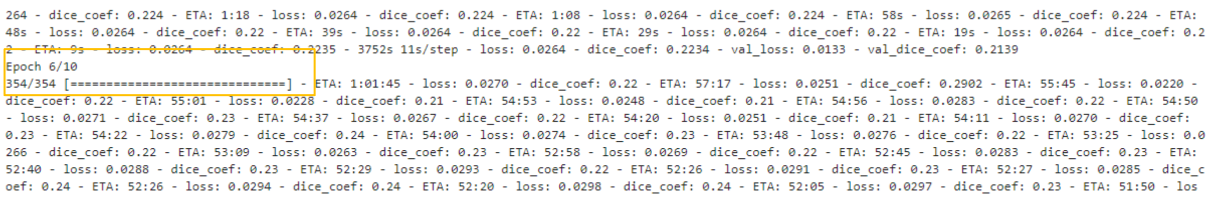
```
ef: 0.204 - ETA: 1:28 - loss: 0.0270 - dice_coef: 0.204 - ETA: 1:18 - loss: 0.0271 - dice_coef: 0.204 - ETA: 1:08 - loss: 0.0271 - dice_coef: 0.204 - ETA: 58s - loss:
0.0271 - dice_coef: 0.204 - ETA: 48s - loss: 0.0271 - dice_coef: 0.20 - ETA: 39s - loss: 0.0272 - dice_coef: 0.20 - ETA: 29s - loss: 0.0272 - dice_coef: 0.20 - ETA: 19
s - loss: 0.0272 - dice_coef: 0.20 - ETA: 9s - loss: 0.0272 - dice_coef: 0.2053 - 3780s 11s/step - loss: 0.0272 - dice_coef: 0.2051 - val_loss: 0.0139 - val_dice_coef:
0.2179
Epoch 4/10
354/354 [==============================] - ETA: 1:15:28 - loss: 0.0310 - dice_coef: 0.15 - ETA: 1:09:46 - loss: 0.0361 - dice_coef: 0.13 - ETA: 1:07:53 - loss: 0.0320
- dice_coef: 0.16 - ETA: 1:05:45 - loss: 0.0327 - dice_coef: 0.22 - ETA: 1:04:41 - loss: 0.0291 - dice_coef: 0.21 - ETA: 1:03:52 - loss: 0.0265 - dice_coef: 0.21 - ET
A: 1:03:32 - loss: 0.0255 - dice_coef: 0.23 - ETA: 1:03:14 - loss: 0.0305 - dice_coef: 0.22 - ETA: 1:02:51 - loss: 0.0287 - dice_coef: 0.22 - ETA: 1:02:27 - loss: 0.02
75 - dice_coef: 0.21 - ETA: 1:02:04 - loss: 0.0277 - dice_coef: 0.21 - ETA: 1:01:24 - loss: 0.0285 - dice_coef: 0.21 - ETA: 1:00:43 - loss: 0.0278 - dice_coef: 0.21 -
ETA: 1:00:22 - loss: 0.0279 - dice_coef: 0.21 - ETA: 59:49 - loss: 0.0274 - dice_coef: 0.2230 - ETA: 59:23 - loss: 0.0267 - dice_coef: 0.21 - ETA: 59:08 - loss: 0.0259
```

Epoch 5/10



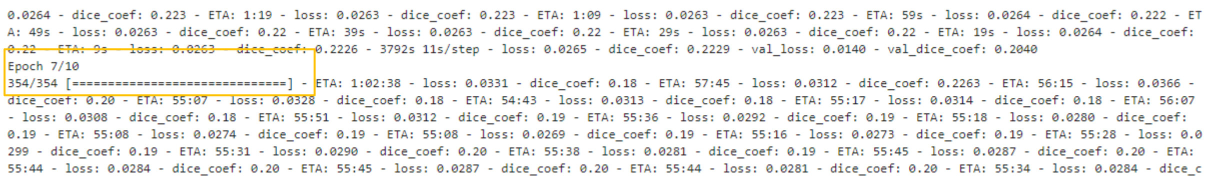
```
0.223 - ETA: 1:28 - loss: 0.0263 - dice_coef: 0.223 - ETA: 1:18 - loss: 0.0264 - dice_coef: 0.223 - ETA: 1:08 - loss: 0.0264 - dice_coef: 0.223 - ETA: 58s - loss: 0.02
65 - dice_coef: 0.223 - ETA: 48s - loss: 0.0264 - dice_coef: 0.22 - ETA: 39s - loss: 0.0264 - dice_coef: 0.22 - ETA: 29s - loss: 0.0264 - dice_coef: 0.22 - ETA: 19s -
loss: 0.0264 - dice_coef: 0.22 - ETA: 9s - loss: 0.0264 - dice_coef: 0.2220 - 3778s 11s/step - loss: 0.0264 - dice_coef: 0.2221 - val_loss: 0.0138 - val_dice_coef: 0.1
854
Epoch 5/10
354/354 [==============================] - ETA: 58:32 - loss: 0.0179 - dice_coef: 0.23 - ETA: 56:42 - loss: 0.0284 - dice_coef: 0.28 - ETA: 55:18 - loss: 0.0226 - dice
_coef: 0.28 - ETA: 54:47 - loss: 0.0188 - dice_coef: 0.22 - ETA: 54:14 - loss: 0.0241 - dice_coef: 0.22 - ETA: 54:16 - loss: 0.0232 - dice_coef: 0.23 - ETA: 54:15 - lo
ss: 0.0226 - dice_coef: 0.22 - ETA: 54:01 - loss: 0.0245 - dice_coef: 0.21 - ETA: 53:52 - loss: 0.0262 - dice_coef: 0.20 - ETA: 53:36 - loss: 0.0252 - dice_coef: 0.20
- ETA: 53:29 - loss: 0.0266 - dice_coef: 0.21 - ETA: 53:27 - loss: 0.0258 - dice_coef: 0.22 - ETA: 53:08 - loss: 0.0254 - dice_coef: 0.22 - ETA: 53:06 - loss: 0.0274 -
dice_coef: 0.23 - ETA: 52:46 - loss: 0.0270 - dice_coef: 0.24 - ETA: 52:31 - loss: 0.0270 - dice_coef: 0.23 - ETA: 52:14 - loss: 0.0268 - dice_coef: 0.23 - ETA: 52:04
- loss: 0.0262 - dice_coef: 0.24 - ETA: 51:56 - loss: 0.0258 - dice_coef: 0.24 - ETA: 51:41 - loss: 0.0256 - dice_coef: 0.24 - ETA: 51:34 - loss: 0.0252 - dice_coef:
```

Epoch 6/10



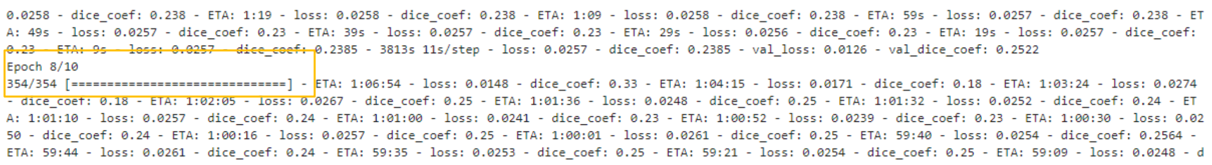
```
264 - dice_coef: 0.224 - ETA: 1:18 - loss: 0.0264 - dice_coef: 0.224 - ETA: 1:08 - loss: 0.0264 - dice_coef: 0.224 - ETA: 58s - loss: 0.0265 - dice_coef: 0.224 - ETA:
48s - loss: 0.0264 - dice_coef: 0.22 - ETA: 39s - loss: 0.0264 - dice_coef: 0.22 - ETA: 29s - loss: 0.0264 - dice_coef: 0.22 - ETA: 19s - loss: 0.0264 - dice_coef: 0.2
2 - ETA: 9s - loss: 0.0264 - dice_coef: 0.2235 - 3752s 11s/step - loss: 0.0264 - dice_coef: 0.2234 - val_loss: 0.0133 - val_dice_coef: 0.2139
Epoch 6/10
354/354 [==============================] - ETA: 1:01:45 - loss: 0.0270 - dice_coef: 0.22 - ETA: 57:17 - loss: 0.0251 - dice_coef: 0.2902 - ETA: 55:45 - loss: 0.0220 -
dice_coef: 0.22 - ETA: 55:01 - loss: 0.0228 - dice_coef: 0.21 - ETA: 54:53 - loss: 0.0248 - dice_coef: 0.21 - ETA: 54:56 - loss: 0.0283 - dice_coef: 0.22 - ETA: 54:50
- loss: 0.0271 - dice_coef: 0.23 - ETA: 54:37 - loss: 0.0267 - dice_coef: 0.22 - ETA: 54:20 - loss: 0.0251 - dice_coef: 0.21 - ETA: 54:11 - loss: 0.0270 - dice_coef:
0.23 - ETA: 54:22 - loss: 0.0279 - dice_coef: 0.24 - ETA: 54:00 - loss: 0.0274 - dice_coef: 0.23 - ETA: 53:48 - loss: 0.0276 - dice_coef: 0.22 - ETA: 53:25 - loss: 0.0
266 - dice_coef: 0.22 - ETA: 53:09 - loss: 0.0263 - dice_coef: 0.23 - ETA: 52:58 - loss: 0.0269 - dice_coef: 0.22 - ETA: 52:45 - loss: 0.0283 - dice_coef: 0.23 - ETA:
52:40 - loss: 0.0288 - dice_coef: 0.23 - ETA: 52:29 - loss: 0.0293 - dice_coef: 0.22 - ETA: 52:26 - loss: 0.0291 - dice_coef: 0.23 - ETA: 52:27 - loss: 0.0285 - dice_c
oef: 0.24 - ETA: 52:26 - loss: 0.0294 - dice_coef: 0.24 - ETA: 52:20 - loss: 0.0298 - dice_coef: 0.24 - ETA: 52:05 - loss: 0.0297 - dice_coef: 0.23 - ETA: 51:50 - los
```

Epoch 7/10



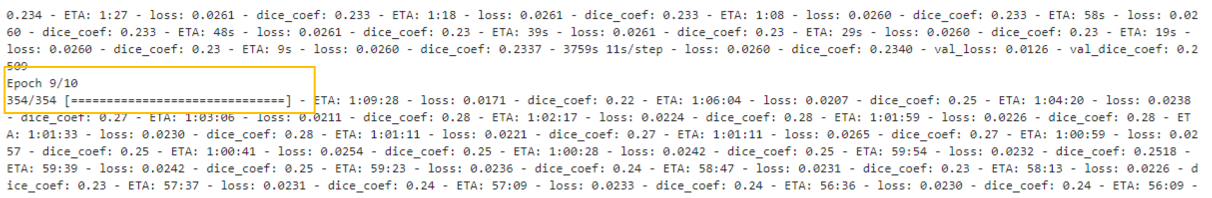
```
0.0264 - dice_coef: 0.223 - ETA: 1:19 - loss: 0.0263 - dice_coef: 0.223 - ETA: 1:09 - loss: 0.0263 - dice_coef: 0.223 - ETA: 59s - loss: 0.0264 - dice_coef: 0.222 - ET
A: 49s - loss: 0.0263 - dice_coef: 0.22 - ETA: 39s - loss: 0.0263 - dice_coef: 0.22 - ETA: 29s - loss: 0.0263 - dice_coef: 0.22 - ETA: 19s - loss: 0.0264 - dice_coef:
0.22 - ETA: 9s - loss: 0.0263 - dice_coef: 0.2226 - 3792s 11s/step - loss: 0.0265 - dice_coef: 0.2229 - val_loss: 0.0140 - val_dice_coef: 0.2040
Epoch 7/10
354/354 [==============================] - ETA: 1:02:38 - loss: 0.0331 - dice_coef: 0.18 - ETA: 57:45 - loss: 0.0312 - dice_coef: 0.2263 - ETA: 56:15 - loss: 0.0366 -
dice_coef: 0.20 - ETA: 55:07 - loss: 0.0328 - dice_coef: 0.18 - ETA: 54:43 - loss: 0.0313 - dice_coef: 0.18 - ETA: 55:17 - loss: 0.0314 - dice_coef: 0.18 - ETA: 56:07
- loss: 0.0308 - dice_coef: 0.18 - ETA: 55:51 - loss: 0.0312 - dice_coef: 0.19 - ETA: 55:36 - loss: 0.0292 - dice_coef: 0.19 - ETA: 55:18 - loss: 0.0280 - dice_coef:
0.19 - ETA: 55:08 - loss: 0.0274 - dice_coef: 0.19 - ETA: 55:08 - loss: 0.0269 - dice_coef: 0.19 - ETA: 55:16 - loss: 0.0273 - dice_coef: 0.19 - ETA: 55:28 - loss: 0.0
299 - dice_coef: 0.19 - ETA: 55:31 - loss: 0.0290 - dice_coef: 0.20 - ETA: 55:38 - loss: 0.0281 - dice_coef: 0.19 - ETA: 55:45 - loss: 0.0287 - dice_coef: 0.20 - ETA:
55:44 - loss: 0.0284 - dice_coef: 0.20 - ETA: 55:45 - loss: 0.0287 - dice_coef: 0.20 - ETA: 55:44 - loss: 0.0281 - dice_coef: 0.20 - ETA: 55:34 - loss: 0.0284 - dice_c
```

Epoch 8/10



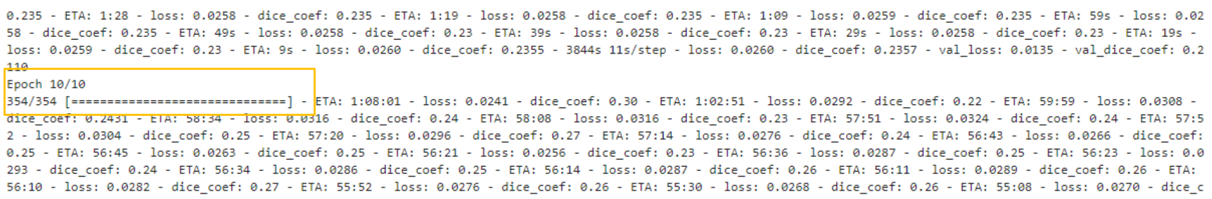
```
0.0258 - dice_coef: 0.238 - ETA: 1:19 - loss: 0.0258 - dice_coef: 0.238 - ETA: 1:09 - loss: 0.0258 - dice_coef: 0.238 - ETA: 59s - loss: 0.0257 - dice_coef: 0.238 - ET
A: 49s - loss: 0.0257 - dice_coef: 0.23 - ETA: 39s - loss: 0.0257 - dice_coef: 0.23 - ETA: 29s - loss: 0.0256 - dice_coef: 0.23 - ETA: 19s - loss: 0.0257 - dice_coef:
0.23 - ETA: 9s - loss: 0.0257 - dice_coef: 0.2385 - 3813s 11s/step - loss: 0.0257 - dice_coef: 0.2385 - val_loss: 0.0126 - val_dice_coef: 0.2522
Epoch 8/10
354/354 [==============================] - ETA: 1:06:54 - loss: 0.0148 - dice_coef: 0.33 - ETA: 1:04:15 - loss: 0.0171 - dice_coef: 0.18 - ETA: 1:03:24 - loss: 0.0274
- dice_coef: 0.18 - ETA: 1:02:05 - loss: 0.0267 - dice_coef: 0.25 - ETA: 1:01:36 - loss: 0.0248 - dice_coef: 0.25 - ETA: 1:01:32 - loss: 0.0252 - dice_coef: 0.24 - ET
A: 1:01:10 - loss: 0.0257 - dice_coef: 0.24 - ETA: 1:01:00 - loss: 0.0241 - dice_coef: 0.23 - ETA: 1:00:52 - loss: 0.0239 - dice_coef: 0.23 - ETA: 1:00:30 - loss: 0.02
50 - dice_coef: 0.24 - ETA: 1:00:16 - loss: 0.0257 - dice_coef: 0.25 - ETA: 1:00:01 - loss: 0.0261 - dice_coef: 0.25 - ETA: 59:40 - loss: 0.0254 - dice_coef: 0.2564 -
ETA: 59:44 - loss: 0.0261 - dice_coef: 0.24 - ETA: 59:35 - loss: 0.0253 - dice_coef: 0.25 - ETA: 59:21 - loss: 0.0254 - dice_coef: 0.25 - ETA: 59:09 - loss: 0.0248 - d
```

Epoch 9/10



```
0.234 - ETA: 1:27 - loss: 0.0261 - dice_coef: 0.233 - ETA: 1:18 - loss: 0.0261 - dice_coef: 0.233 - ETA: 1:08 - loss: 0.0260 - dice_coef: 0.233 - ETA: 58s - loss: 0.02
60 - dice_coef: 0.233 - ETA: 48s - loss: 0.0261 - dice_coef: 0.23 - ETA: 39s - loss: 0.0261 - dice_coef: 0.23 - ETA: 29s - loss: 0.0260 - dice_coef: 0.23 - ETA: 19s -
loss: 0.0260 - dice_coef: 0.23 - ETA: 9s - loss: 0.0260 - dice_coef: 0.2337 - 3759s 11s/step - loss: 0.0260 - dice_coef: 0.2340 - val_loss: 0.0126 - val_dice_coef: 0.2
509
Epoch 9/10
354/354 [==============================] - ETA: 1:09:28 - loss: 0.0171 - dice_coef: 0.22 - ETA: 1:06:04 - loss: 0.0207 - dice_coef: 0.25 - ETA: 1:04:20 - loss: 0.0238
- dice_coef: 0.27 - ETA: 1:03:06 - loss: 0.0211 - dice_coef: 0.28 - ETA: 1:02:17 - loss: 0.0224 - dice_coef: 0.28 - ETA: 1:01:59 - loss: 0.0226 - dice_coef: 0.28 - ET
A: 1:01:33 - loss: 0.0230 - dice_coef: 0.28 - ETA: 1:01:11 - loss: 0.0221 - dice_coef: 0.27 - ETA: 1:01:11 - loss: 0.0265 - dice_coef: 0.27 - ETA: 1:00:59 - loss: 0.02
57 - dice_coef: 0.25 - ETA: 1:00:41 - loss: 0.0254 - dice_coef: 0.25 - ETA: 1:00:28 - loss: 0.0242 - dice_coef: 0.25 - ETA: 59:54 - loss: 0.0232 - dice_coef: 0.2518 -
ETA: 59:39 - loss: 0.0242 - dice_coef: 0.25 - ETA: 59:23 - loss: 0.0236 - dice_coef: 0.24 - ETA: 58:47 - loss: 0.0231 - dice_coef: 0.23 - ETA: 58:13 - loss: 0.0226 - d
ice_coef: 0.23 - ETA: 57:37 - loss: 0.0231 - dice_coef: 0.24 - ETA: 57:09 - loss: 0.0233 - dice_coef: 0.24 - ETA: 56:36 - loss: 0.0230 - dice_coef: 0.24 - ETA: 56:09 -
```

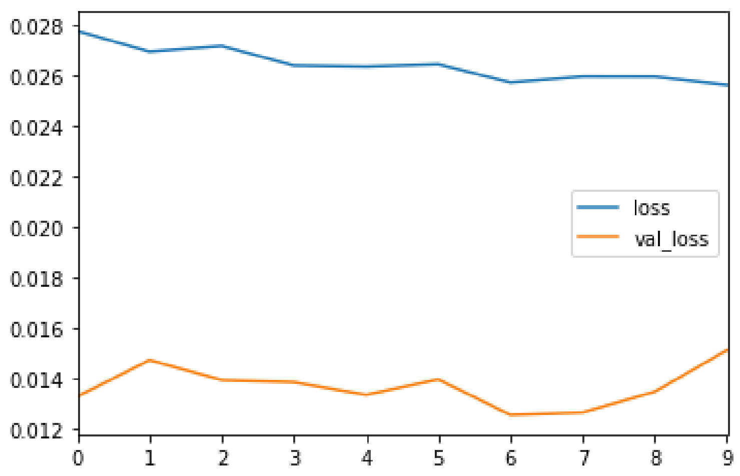
Epoch 10/10

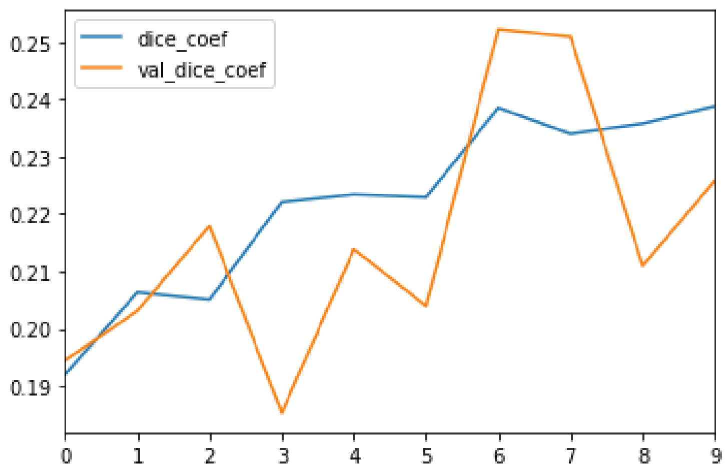


```
0.235 - ETA: 1:28 - loss: 0.0258 - dice_coef: 0.235 - ETA: 1:19 - loss: 0.0258 - dice_coef: 0.235 - ETA: 1:09 - loss: 0.0259 - dice_coef: 0.235 - ETA: 59s - loss: 0.02
58 - dice_coef: 0.235 - ETA: 49s - loss: 0.0258 - dice_coef: 0.23 - ETA: 39s - loss: 0.0258 - dice_coef: 0.23 - ETA: 29s - loss: 0.0258 - dice_coef: 0.23 - ETA: 19s -
loss: 0.0259 - dice_coef: 0.23 - ETA: 9s - loss: 0.0260 - dice_coef: 0.2355 - 3844s 11s/step - loss: 0.0260 - dice_coef: 0.2357 - val_loss: 0.0135 - val_dice_coef: 0.2
110
Epoch 10/10
354/354 [==============================] - ETA: 1:08:01 - loss: 0.0241 - dice_coef: 0.30 - ETA: 1:02:51 - loss: 0.0292 - dice_coef: 0.22 - ETA: 59:59 - loss: 0.0308 -
dice_coef: 0.2431 - ETA: 58:34 - loss: 0.0316 - dice_coef: 0.24 - ETA: 58:08 - loss: 0.0316 - dice_coef: 0.23 - ETA: 57:51 - loss: 0.0324 - dice_coef: 0.24 - ETA: 57:5
2 - loss: 0.0304 - dice_coef: 0.25 - ETA: 57:20 - loss: 0.0296 - dice_coef: 0.27 - ETA: 57:14 - loss: 0.0276 - dice_coef: 0.24 - ETA: 56:43 - loss: 0.0266 - dice_coef:
0.25 - ETA: 56:45 - loss: 0.0263 - dice_coef: 0.25 - ETA: 56:21 - loss: 0.0256 - dice_coef: 0.23 - ETA: 56:36 - loss: 0.0287 - dice_coef: 0.25 - ETA: 56:23 - loss: 0.0
293 - dice_coef: 0.24 - ETA: 56:34 - loss: 0.0286 - dice_coef: 0.25 - ETA: 56:14 - loss: 0.0287 - dice_coef: 0.26 - ETA: 56:11 - loss: 0.0289 - dice_coef: 0.26 - ETA:
56:10 - loss: 0.0282 - dice_coef: 0.27 - ETA: 55:52 - loss: 0.0276 - dice_coef: 0.26 - ETA: 55:30 - loss: 0.0268 - dice_coef: 0.26 - ETA: 55:08 - loss: 0.0270 - dice_c
```

```
history_df = pd.DataFrame(history.history)
history_df[['loss', 'val_loss']].plot()
history_df[['dice_coef', 'val_dice_coef']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x27c2bae5080>
```





```python
model.load_weights('model.h5')
test_df = []

for i in range(0, filtered_test_imgs.shape[0], 300):
    batch_idx = list(
        range(i, min(filtered_test_imgs.shape[0], i + 300))
    )

    test_generator = DataGenerator(
        batch_idx,
        df=filtered_test_imgs,
        shuffle=False,
        mode='predict',
        base_path='severstal-steel-defect-detection/test_images',
        target_df=filtered_sub_df,
        batch_size=1,
        n_classes=4
    )
    batch_pred_masks = model.predict_generator(
        test_generator,
        workers=1,
        verbose=1,
        use_multiprocessing=False
    )

    for j, b in tqdm(enumerate(batch_idx)):
        filename = filtered_test_imgs['ImageId'].iloc[b]
        image_df = filtered_sub_df[filtered_sub_df['ImageId'] == filename].copy()

        pred_masks = batch_pred_masks[j, ].round().astype(int)
        pred_rles = build_rles(pred_masks)

        image_df['EncodedPixels'] = pred_rles
        test_df.append(image_df)

    gc.collect()
```

```
300/300 [==============================] - ETA: 18:4 - ETA: 10:1 - ETA: 7:1 - ETA: 5: - ETA: 5: - ETA: 4: - ETA: 4: - ETA: 3: - ETA: 3: - ETA: 3: - ETA: 3: - ETA: 3: -
ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 1: - ETA: 1:
- ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA:
1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ET
A: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: -
ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1:
- ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA:
1: - ETA: 59s - ETA: 59 - ETA: 59 - ETA: 59 - ETA: 59 - ETA: 59 - ETA: 58 - ETA: 58 - ETA: 58 - ETA: 57 - ETA: 57 - ETA: 56 - ETA: 56 - ETA: 55 - ETA: 55 - ETA: 55 - E
TA: 54 - ETA: 54 - ETA: 53 - ETA: 53 - ETA: 53 - ETA: 52 - ETA: 52 - ETA: 52 - ETA: 51 - ETA: 51 - ETA: 50 - ETA: 50 - ETA: 50 - ETA: 49 - ETA: 49 - ETA: 48 - ETA: 48
- ETA: 48 - ETA: 47 - ETA: 47 - ETA: 47 - ETA: 46 - ETA: 46 - ETA: 45 - ETA: 45 - ETA: 45 - ETA: 44 - ETA: 44 - ETA: 44 - ETA: 44 - ETA: 43 - ETA: 43 - ETA: 43 - ETA:
43 - ETA: 42 - ETA: 42 - ETA: 42 - ETA: 41 - ETA: 41 - ETA: 41 - ETA: 40 - ETA: 40 - ETA: 40 - ETA: 39 - ETA: 39 - ETA: 38 - ETA: 38 - ETA: 38 - ETA: 37 - ETA: 37 - ET
A: 37 - ETA: 36 - ETA: 36 - ETA: 36 - ETA: 35 - ETA: 35 - ETA: 35 - ETA: 34 - ETA: 34 - ETA: 34 - ETA: 34 - ETA: 33 - ETA: 33 - ETA: 33 - ETA: 33 - ETA: 32 - ETA: 32 -
ETA: 32 - ETA: 32 - ETA: 32 - ETA: 31 - ETA: 31 - ETA: 31 - ETA: 30 - ETA: 30 - ETA: 30 - ETA: 29 - ETA: 29 - ETA: 29 - ETA: 28 - ETA: 28 - ETA: 28 - ETA: 27 - ETA: 27
- ETA: 27 - ETA: 26 - ETA: 26 - ETA: 26 - ETA: 25 - ETA: 25 - ETA: 25 - ETA: 24 - ETA: 24 - ETA: 24 - ETA: 23 - ETA: 23 - ETA: 23 - ETA: 22 - ETA: 22 - ETA: 22 - ETA:
21 - ETA: 21 - ETA: 21 - ETA: 20 - ETA: 20 - ETA: 20 - ETA: 19 - ETA: 19 - ETA: 19 - ETA: 19 - ETA: 18 - ETA: 18 - ETA: 18 - ETA: 17 - ETA: 17 - ETA: 17 - ETA: 16 - ET
A: 16 - ETA: 16 - ETA: 15 - ETA: 15 - ETA: 15 - ETA: 14 - ETA: 14 - ETA: 14 - ETA: 13 - ETA: 13 - ETA: 13 - ETA: 12 - ETA: 12 - ETA: 12 - ETA: 12 - ETA: 11 - ETA: 11 -
ETA: 11 - ETA: 10 - ETA: 10 - ETA: 10 - ETA: 9 - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:
- ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - 92s 306ms/step
300it [00:48,  6.08it/s]
300/300 [==============================] - ETA: 8: - ETA: 4: - ETA: 3: - ETA: 3: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 1: - ETA: 1: - ETA:
1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ET
A: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: -
ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1:
- ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA:
1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ET
A: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 59s - ETA: 59 - ETA: 59 - ETA: 58 - ETA: 58 - ETA: 58 - ETA: 57 - ETA: 57 - ETA: 57 - ETA: 56
- ETA: 56 - ETA: 56 - ETA: 56 - ETA: 55 - ETA: 55 - ETA: 55 - ETA: 54 - ETA: 54 - ETA: 54 - ETA: 53 - ETA: 53 - ETA: 53 - ETA: 52 - ETA: 52 - ETA: 52 - ETA: 51 - ETA:
51 - ETA: 51 - ETA: 50 - ETA: 50 - ETA: 49 - ETA: 49 - ETA: 49 - ETA: 48 - ETA: 48 - ETA: 48 - ETA: 47 - ETA: 47 - ETA: 47 - ETA: 46 - ETA: 46 - ETA: 46 - ETA: 45 - ET
```

```
300it [00:48,  6.08it/s]
300/300 [==============================] - ETA: 8: - ETA: 4: - ETA: 3: - ETA: 3: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 2: - ETA: 1: - ETA: 1: - ETA:
1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ET
A: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: -
ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1:
- ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA:
1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ET
A: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 1: - ETA: 59s - ETA: 59 - ETA: 59 - ETA: 58 - ETA: 58 - ETA: 58 - ETA: 57 - ETA: 57 - ETA: 57 - ETA: 56
- ETA: 56 - ETA: 56 - ETA: 56 - ETA: 55 - ETA: 55 - ETA: 55 - ETA: 54 - ETA: 54 - ETA: 54 - ETA: 53 - ETA: 53 - ETA: 53 - ETA: 52 - ETA: 52 - ETA: 52 - ETA: 51 - ETA:
51 - ETA: 51 - ETA: 50 - ETA: 50 - ETA: 49 - ETA: 49 - ETA: 49 - ETA: 48 - ETA: 48 - ETA: 48 - ETA: 47 - ETA: 47 - ETA: 47 - ETA: 46 - ETA: 46 - ETA: 46 - ETA: 45 - ET
A: 45 - ETA: 45 - ETA: 44 - ETA: 44 - ETA: 44 - ETA: 43 - ETA: 43 - ETA: 43 - ETA: 43 - ETA: 42 - ETA: 42 - ETA: 42 - ETA: 41 - ETA: 41 - ETA: 41 - ETA: 40 - ETA: 40 -
ETA: 40 - ETA: 40 - ETA: 39 - ETA: 39 - ETA: 39 - ETA: 38 - ETA: 38 - ETA: 38 - ETA: 37 - ETA: 37 - ETA: 37 - ETA: 36 - ETA: 36 - ETA: 36 - ETA: 36 - ETA: 35 - ETA: 35
- ETA: 35 - ETA: 34 - ETA: 34 - ETA: 34 - ETA: 34 - ETA: 33 - ETA: 33 - ETA: 33 - ETA: 33 - ETA: 33 - ETA: 32 - ETA: 32 - ETA: 32 - ETA: 32 - ETA: 31 - ETA: 31 - ETA:
31 - ETA: 30 - ETA: 30 - ETA: 30 - ETA: 30 - ETA: 30 - ETA: 30 - ETA: 29 - ETA: 29 - ETA: 29 - ETA: 28 - ETA: 28 - ETA: 28 - ETA: 27 - ETA: 27 - ETA: 27 - ETA: 26 - ET
A: 26 - ETA: 26 - ETA: 25 - ETA: 25 - ETA: 25 - ETA: 24 - ETA: 24 - ETA: 24 - ETA: 23 - ETA: 23 - ETA: 23 - ETA: 22 - ETA: 22 - ETA: 22 - ETA: 21 - ETA: 21 - ETA: 21 -
ETA: 21 - ETA: 20 - ETA: 20 - ETA: 20 - ETA: 19 - ETA: 19 - ETA: 19 - ETA: 18 - ETA: 18 - ETA: 18 - ETA: 17 - ETA: 17 - ETA: 17 - ETA: 16 - ETA: 16 - ETA: 16 - ETA: 16
- ETA: 15 - ETA: 15 - ETA: 15 - ETA: 14 - ETA: 14 - ETA: 14 - ETA: 13 - ETA: 13 - ETA: 13 - ETA: 12 - ETA: 12 - ETA: 12 - ETA: 12 - ETA: 11 - ETA: 11 - ETA: 11 - ETA:
10 - ETA: 10 - ETA: 10 - ETA: 9 - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:
- ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - 93s 311ms/step
300it [00:58,  7.24it/s]
121/121 [==============================] - ETA: 1: - ETA: 1: - ETA: 53s - ETA: 46 - ETA: 42 - ETA: 40 - ETA: 38 - ETA: 36 - ETA: 35 - ETA: 34 - ETA: 33 - ETA: 32 - ET
A: 31 - ETA: 31 - ETA: 30 - ETA: 30 - ETA: 30 - ETA: 29 - ETA: 29 - ETA: 29 - ETA: 28 - ETA: 28 - ETA: 27 - ETA: 27 - ETA: 26 - ETA: 26 - ETA: 26 - ETA: 25 - ETA: 25 -
ETA: 24 - ETA: 24 - ETA: 24 - ETA: 23 - ETA: 23 - ETA: 23 - ETA: 22 - ETA: 22 - ETA: 22 - ETA: 22 - ETA: 21 - ETA: 21 - ETA: 21 - ETA: 20 - ETA: 20 - ETA: 20 - ETA: 19
- ETA: 19 - ETA: 19 - ETA: 19 - ETA: 18 - ETA: 18 - ETA: 18 - ETA: 17 - ETA: 17 - ETA: 17 - ETA: 17 - ETA: 16 - ETA: 16 - ETA: 16 - ETA: 15 - ETA: 15 - ETA: 15 - ETA:
15 - ETA: 14 - ETA: 14 - ETA: 14 - ETA: 14 - ETA: 13 - ETA: 13 - ETA: 13 - ETA: 12 - ETA: 12 - ETA: 12 - ETA: 12 - ETA: 11 - ETA: 11 - ETA: 11 - ETA: 11 - ETA: 10 - ET
A: 10 - ETA: 10 - ETA: 10 - ETA: 9 - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ET
A:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - 3
1s 259ms/step
121it [00:18,  5.99it/s]
```

Output

```python
test_df = pd.concat(test_df)
print(test_df.shape)
test_df.head()
```

```
(2884, 3)
```

	ImageId_ClassId	EncodedPixels	ImageId
0	00bbcd9af.jpg_1		00bbcd9af.jpg
1	00bbcd9af.jpg_2		00bbcd9af.jpg
2	00bbcd9af.jpg_3	159404 1 159406 5 159659 9 159912 13 160168 1 ...	00bbcd9af.jpg
3	00bbcd9af.jpg_4		00bbcd9af.jpg
4	0108ce457.jpg_1		0108ce457.jpg

```python
final_submission_df = pd.concat([test_df, null_sub_df])
print(final_submission_df.shape)
final_submission_df.head()
```

```
(7204, 3)
```

	ImageId_ClassId	EncodedPixels	ImageId
0	00bbcd9af.jpg_1		00bbcd9af.jpg
1	00bbcd9af.jpg_2		00bbcd9af.jpg
2	00bbcd9af.jpg_3	159404 1 159406 5 159659 9 159912 13 160168 1 ...	00bbcd9af.jpg
3	00bbcd9af.jpg_4		00bbcd9af.jpg
4	0108ce457.jpg_1		0108ce457.jpg

```python
final_submission_df[['ImageId_ClassId', 'EncodedPixels']].to_csv('submission.csv', index=False)
```

CONCLUSION

Here, since we don't have the values of the test dataset, so as to verify our outcome on it, and get the accuracy, we just predict for the test dataset. And, we don't actually get any accuracy on test dataset, rather we get the validation accuracy.

We divide the training set into – training set and validation set. Data is trained on the training set and the model trained is verified on the validation dataset. Validation is one of the two techniques to overcome the problem of overfitting (the other technique is Regularization).

Overfitting occurs when the model learns the target function along with the noise. In order to try to fit the data completely, it also fits noise into it, thus causing overfitting.

Overfitting decreases as the count of training data increases. This can be shown by the dip which occurs in the value of E_{out} as N increases (where N is the number of training data points).

In the above scenario, the val_dice_coeff fails to track dice_coeff, which means that the model has overfit the data. This can be seen as a result of setting 10 epochs which is large when compared to other parameters.

Solution to this problem would be Early Stopping (Stopping the training at a point would actually prevent E_{out} from loosing track of E_{in}).

Cross-Validation also may help in preventing too much overfitting.

However, as of now, a Convolutional Neural Network coupled with a binary classification has been successfully implemented, and the test images are classified as per the defects they are predicted to contain, and the result is stored in submission.csv as per the kaggle guidelines.