SEVERSTAL IMAGE CLASSIFICATION

MACHINE LEARNING NANODEGREE
CAPSTONE PROJECT
RICARDO TERRA
NOVEMBER 2ND, 2020

SUMMARY

Inside this document you will find a capstone project for the final project of Machine Learning Nanodegree. It is included below a brief introduction to the problem statement, objectives, methods for problem solving and the results at the end. The first part is dedicated to explain and explore the chosen dataset, which includes pre-operations (cleaning and wrangling). The second part shows the machine learning classical steps: preparing data, splitting data in training, selecting a model and then feeding with data. The third and last part shows the results and final remarks.

TABLE OF CONTENTS

1. INTRODUCTION

The industry of steel is one of the most important sectors around the world, accounting for massive jobs, high cash flow and delivering steel for many applications. Since industrial revolution in late 17th century, steel is a raw material in high demand for infrastructure, vehicles, ships, etc. Steel can be produced in many shapes, such as: plates, tubes, wires and beams so quality control is utmost important to assure its functionality and safety along its life-cycle.

2. PROBLEM STATEMENT

Steel surface analysis is one of the techniques developed to watch the material condition by revealing cracks, imperfections and patterns that can indicate the need for replacement. Surface analysis is generally performed by a professional inspector, trained and certified for this task. However, this activity can be overwhelming since the number of images taken of a building or and industry plant. Moreover, sometimes this activity can be dangerous, for example, in top of buildings (height) and inside equipment (limited space).

One alternative to improve surface analysis is to train a model to learn steel surface patterns in order to classify whether a surface has an imperfection or not, and if possible, what kind of imperfection it has, using machine learning methods. Important notice that the early detection of any issue in steel might prevent incidents and help the structures and equipment to perform as it is intended.

3. CASE STUDY: SEVERSTAL

Severstal is the biggest siderurgy in Russia, its core business is mining, ore trading and metallurgy. One of the main concerns in this industry is the quality of its produced steel plates, as discussed above. In order to find out innovative solutions to steel quality, Severstal posted a competition in Kaggle [1] last year with the objective to achieve an algorithm capable of classify steel plate defects correctly. The competitors were invited to create new neural networks or to improve existing ones for this task.

The dataset is still available for download even after contest end, so I downloaded it for the purpose of using exclusively for this capstone project. I've tried to implement a classification system using convolutional neural networks (CNN) based on the well-established UNet [2].

One last disclaimer: Although, Udacity has provided some data examples, I've opted to search for a more specific dataset towards my professional formation: chemical engineer. The dataset I've chosen is a collection of steel plates images that can or cannot present an imperfection on its surface.

4. OBJECTIVES

The capstone project from Udacity is the final term of the machine learning nanodegree, students are encouraged to show everything they learn during the 3-month course. It starts by the student choosing a pre-selected case study or looking for a more specific one.

The aim of this project is to build a machine learning application from end-to-end, starting from dataset and pre-operations, training a model and then deploying it for validation. Some def functions will be developed to preprocess the dataframe, train the model and predict the results based on classification. Finally, the code needs to be posted in a Github repository and all the progress written down in this report.

This project might follow all steps from data gathering up to machine learning results. So, the main objectives of my project is aligned with Udacity rubric, passing through all topics in 3 marks:

(i) Download, analyze, clean and explore dataset
(ii) Create a strategy to use machine learning for prediction
(iii) Implement a machine learning solution end-to-end

Note that some steps can be iterative, since we have to return to previous step in order to change parameters values or resample data, for example.

5. EDA

The dataset is originally divided in 2 folders: train and test. Train folder contains approximately 12k images of steel plates in shades of gray, most images shows parts of steel and some contains imperfections that are classified 1 up to 4. While test folder contains about 5,5k images. Further investigation in dataset will be done in exploratory data analysis step. The main structure of this project is shown in Figure 1, note that the folder **test_images** and **sample_submission.csv** aren't considered because it provides neither help nor useful information for this project.

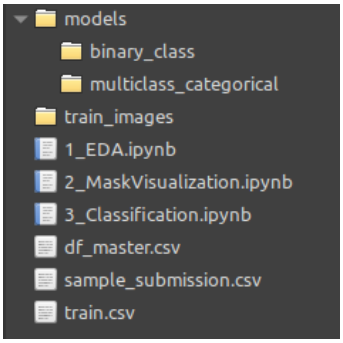

Figure 1 - Main folder structure for this project.

There are 3 notebook files for this project: (1) EDA, (2) Masks and (3) Classification, each one containing, respectively, exploratory data analysis, plot for masks and visualizations and the last one is for model processing. Dataset is listed in "**train.csv**" and its head command is shown below in Figure 2, **ImageId** is for image name plus format (.jpg), **ClassId** represents each class defect and **EncodedPixels** is the

correspondent localization of pixels for a given class in run length encoding format (RLE). This dataframe is composed by 7.095 lines, each row contains a unique sequence of encoded pixels in RLE format for one specific **ClassId**. Notice that multiple lines can refer to same **ImageId**, that is, if a given image has 2 defects this **ImageId** will show up twice, each line will have a different **ClassId** and **Encoded Pixel**.

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

Figure 2 - Train.csv dataframe.

In order to check for a more detailed statistics and further use along this project, the above dataframe was transformed into a master dataframe containing all useful information (file: df_master.csv) showed in Figure 3.

	ImageId	Defect	Defect_no	ClassId	EncodedPixels
0	88df5f0d8.jpg	yes	1	[3]	['111052 12 111284 36 111517 58 111749 82 1119...
1	71b815a2d.jpg	yes	1	[3]	['53666 54 53816 160 54018 214 54274 214 54530...
2	e863b1467.jpg	yes	1	[3]	['92673 127 92929 253 93185 253 93441 253 9369...
3	a3fe8d17b.jpg	no	0	[0]	NaN
4	8e30601c8.jpg	no	0	[0]	NaN

Figure 3 - df_master.csv dataframe.

Additional columns were added to train.csv dataframe, namely, **Defect**, a boolean 'yes' for any defect or 'no' for no defect. Since any can image can contain more than one defect, **ClassId** was reshape to a list of defects [1,2,3,4], **Defect_no** shows the exactly number of defects one image has.

By analyzing **df_master**, it was possible to check that are 6.666 unique images in dataset with at least 1 defect as well as 5.902 unique images with no defects. Furthermore, the Figure 4 present the quantity of images and how many defects it has. Note that most images presented either exactly 1 defect or no defect, so that very few images presents more than 1 defect. Figure 5 shows the ClassId distribution count along dataset, it was possible to see a clearly class imbalance.

Figure 4 - Number of defects per image.

Figure 5 - Class distribution.

Most of images were classified as singles class, [3], [1], [4] and [2]. The imbalance lied in both classes and in quantity of images, for instance, class [3] were dominant with more than 70% of defect dataset and multi class were very sparsely distributed with less than 4%.

In the next chapter, we will discuss more about this imbalance in terms of pixels and how it must be dealt with prior to training step.

6. VISUALIZING MASKS

The following functions will be used to transform encoded pixels in a color map to be merged in original image. The resulting transformation will let us see the shape of the defect as well as its neighborhood. Note that the reversal functions are supplied for sanity check purposes. The main strategy is to use RLE (run-length-enconding) a 1D straight vector to support pixel mapping in images based on [3]. Four special functions were written to convert RLE to mask:

(i) **mask2rle:** input: mask [2D array vector]; outup: RLE
(ii) **rle2mask:** input: RLE; output: mask [2D array vector]
(iii) **build_mask:** input: df and image_name; output: colored mask (256,1600,4)
(iv) **build_rles:** input: colored mask (256,1600,4); output: RLE

Note that a normal image has the following dimensions: height=256, width=1600, RGB channel=3 (which means colorful images). Mask were constructed slightly different with the RGB channel=4, that is, 1 channel por each class defect. For example, if an image presented the defects in class = [3,4], its correspondent mask would be fitted by zeroes in (256,1600,0) and (256,1600,1) and fitted pixels in (256,1600,2) and (256,1600,3).

Another point of attention is that I created a color sign dictionary for each class: ClassId=[1] is blue, [2] is purple, [3] is yellow and [4] is red. Figure 6, 7, 8 and 9 shows some examples of masks applied to ClassId=[1], [2], [3] and [4].

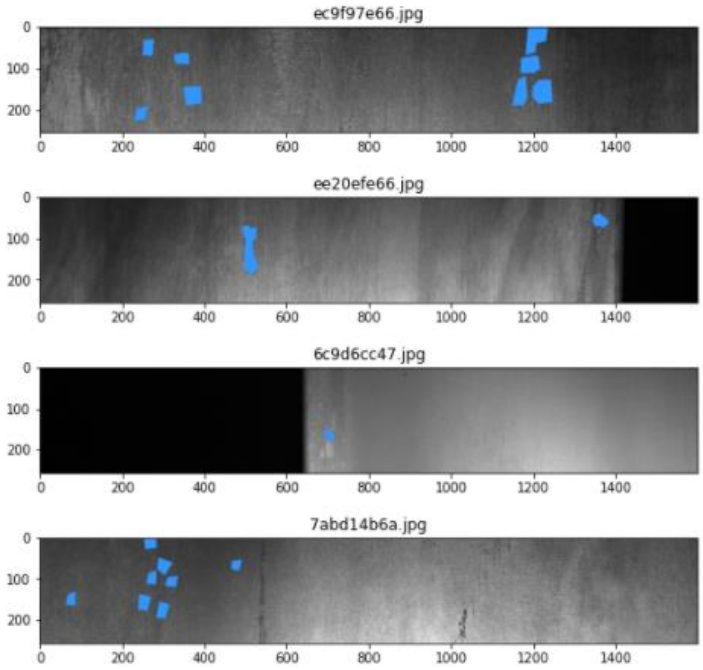

Figure 6 - Masks for ClassId=[1]

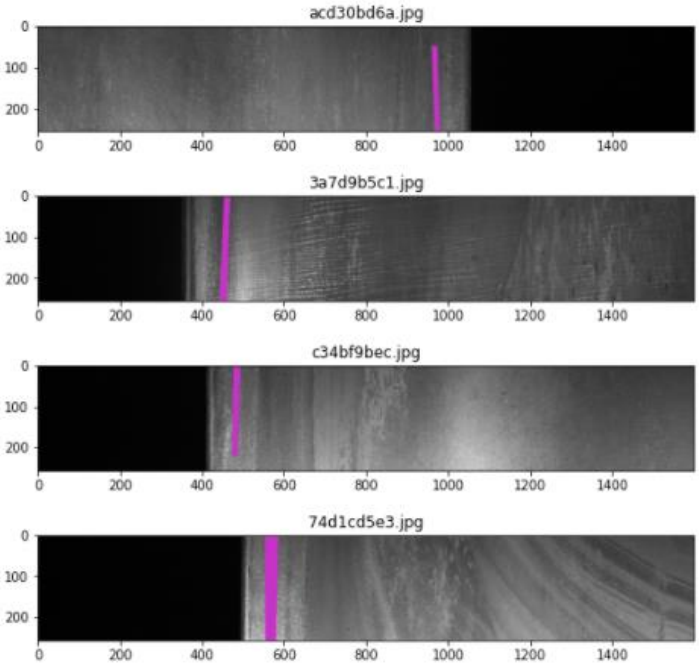

Figure 7 - Mask for ClassId=[2]

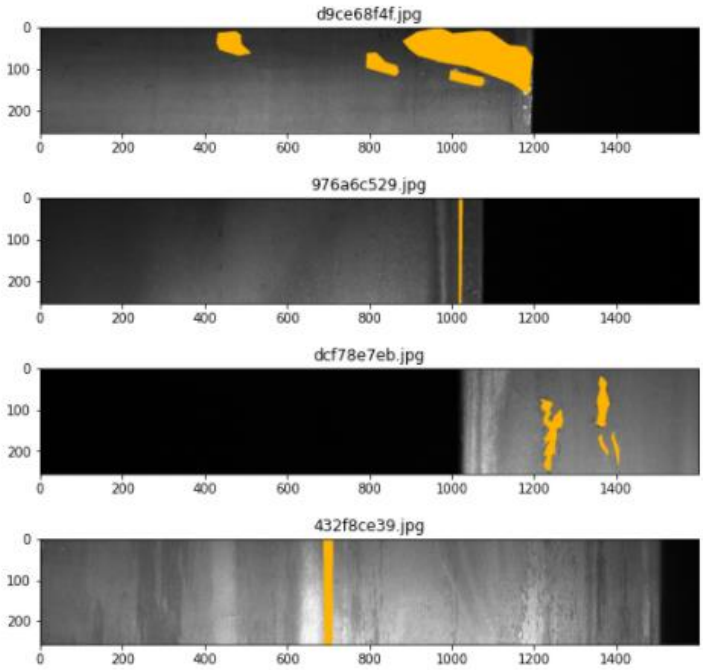

Figure 8 - Mask for ClassId=[3]

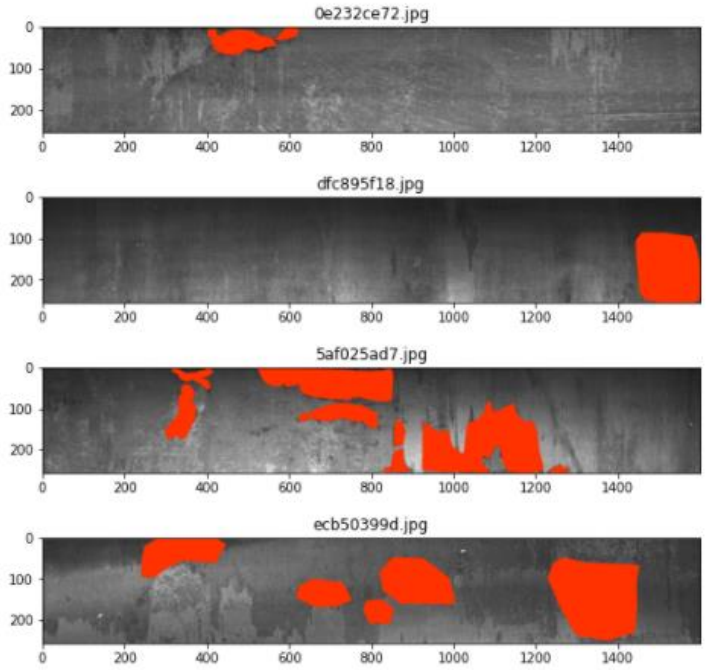

Figure 9 - Masks for ClassId=[4]

Each class has its own defect signature, class 1 is mostly composed by small squared shaped areas coming together in a given region. Class 2 is composed by vertical traces, they come alone and are very sparse. Class 3 also presents vertical traces, but they are more concentrated and some can be also be in horizontal shape. At last, class 4 can be defined as rounded regions very spread to the plate at any position. Figure 10 shows some example of mixed defect in the same plate.

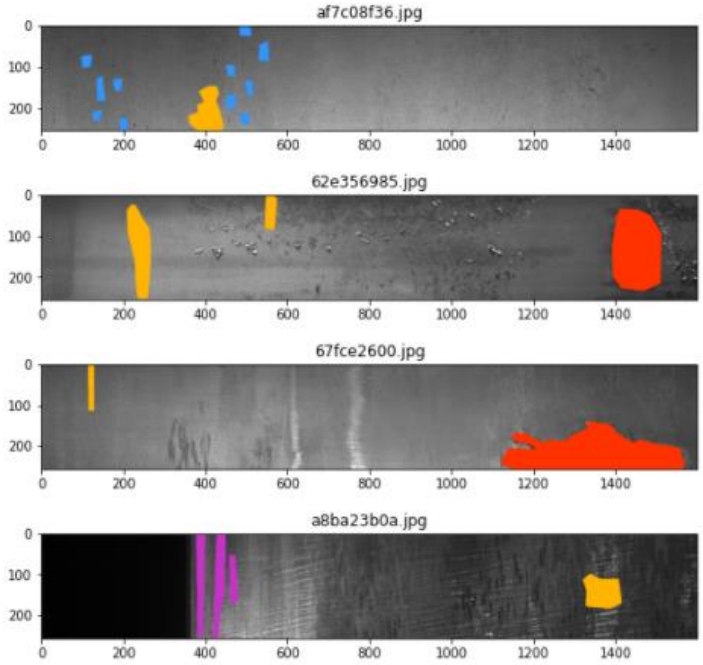

Figure 10 - Masks for multiclass defect.

Besides class imbalance, it's also important to check the relationship between pixels and class itself. Any classification method will analyze image feature by screening though its pixels in order to produce features for class recognition pattern. Pixel count is important can reveal some interesting information with regard to defect area of each class. To make calculation more feasible it's necessary to transform the image background into a full black one RGB (0,0,0). This transformation will result in huge contrast as well as gradient between area defect and image, some examples are shown in Figure 11.

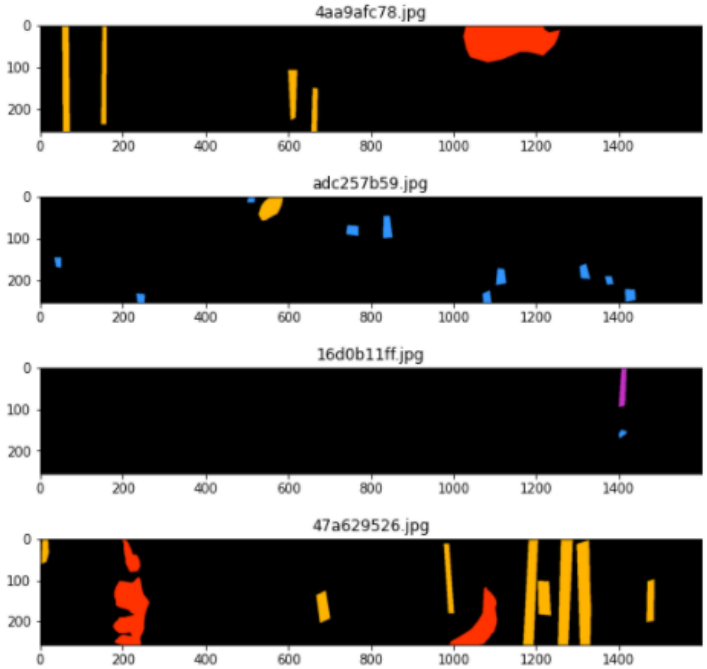

Figure 11 - Black masks for better contrast and pixel count.

After turning all color masks to black masks, the ones with black background, it's more feasible to apply a function that counts all non-zeroes in an array. With this number and the number of total pixels (total pixels = height * width), it's now possible to calculate the defect area % in terms of pixels for each image. Figure 12 shows the main statistics about pixels for each grouped class.

	source	mean	median	std	max	min
0	All_data	12.1565	5.64624	17.6303	179.805	0.0561523
1	Class_1	3.2164	2.48218	2.65488	22.927	0.213135
2	Class_2	2.4916	2.12183	1.45352	10.2708	0.231445
3	Class_3	12.8643	5.99756	19.016	179.805	0.0561523
4	Class_4	17.2609	12.4595	15.2689	94.1309	0.239746
5	Class_multi	18.6143	13.3442	16.2094	117.14	0.597168

Figure 12 - Pixels statistics for each class.

Note that **all_data** refers to all data with at least 1 defect mixed together, it's here only for comparison reasons. As discussed above, **Class_1** is composed by small squared shapes hence, in general, it presents a little pixel total area. **Class_2** has an even smaller area compared to **Class_1** due to its long thin shaped vertical traces very sparsely distributed. However, **Class_3** and **Class_4** presents a higher mean due to larger defect area and more dense concentration of pixels along the plate. As expected, **Class_multi** has the highest mean due to more than one defect per plate.

By the analysis of minimum and maximum, we can see that each class can be present since low percentages up to medium-high percentages. Hence, it's not safe to establish a direct relationship between area of defect and classes itself.

Moreover, in cases with more than one defect per plate, there's no pre-determined pattern of how much area each class will occupy. This being said, the proposed classification system will be constructed using only single class images to predict which class certain plate belongs to.

CLASSIFICATION

Since we have a dataset of images, it's a good approach to start working on a convolutional neural network (CNN) in order to get the images features that are more related to the imperfections. This stated, we can work on a classification network to detect features and classify defect in images.

There are many classification models available in literature, they basically do semantic segmentation, which is pixel classification towards a pre-trained feature or category.

Nowadays, the most prominent image classification models are based on convolutional neural network (CNN) with some grade of modification in either input vector or hidden layer. For example, we can cite: FCN (fully connected), ParseNet (a modification in FCN), ResNet (microsoft net) and UNet (enconding-decoding).

More recently, Faster R-CNN and Mask R-CNN were developed to account pooling schemes to speed up the computing process by selecting zones of interest instead of processing whole image. The abovementioned strategy can be achieved by the following steps: (i) pre-operations (data cleaning and wrangling); (ii) train and test split; (iii) convert images to matrices (NxM; where N is height and M width); (iv) training a pre-selected model; (v) evaluate model hyperparameters; (vi) re-train model and/ or tuning hyperparameters if applicable; (vi) analyze results and make predictions to validate model.

As discussed above there are many models in literature that apply classification in images, however there are some particularities one might keep in mind before selecting a model. For instance, we can highlight UNet as a benchmark in computer vision. Note that this model is based on deep learning kernels so they are also considered heavy and demand high computational power. Furthermore, they do well in academic research, however for some real-time classification or some other edge device, one must rely on less sophisticated models.

In order to solve the classification problem, we might choose a model that perform well in image classification as well as have a moderate processing time due to our dataset. Preliminary tests will be done first using a UNet model written in Keras, this model has good approach and allow fully customization of its hidden layers. UNet is composed by a series on convolutional layers and pooling tasks, forming a U shape net (that's why the name U Net), the first part is responsible for encoding the image data, that is extract its features. The second part is responsible for decoding the image and re-localize its feature and achieve a 'meaning'. UNet has presented good results in many research papers and moderate computational effort since its network well known and pre-programmed.

Figure 13 shows a UNet architecture, each step is either convolutional net or maxpooling sequence, originating a CNN with many hidden layers able to detect patterns in images.

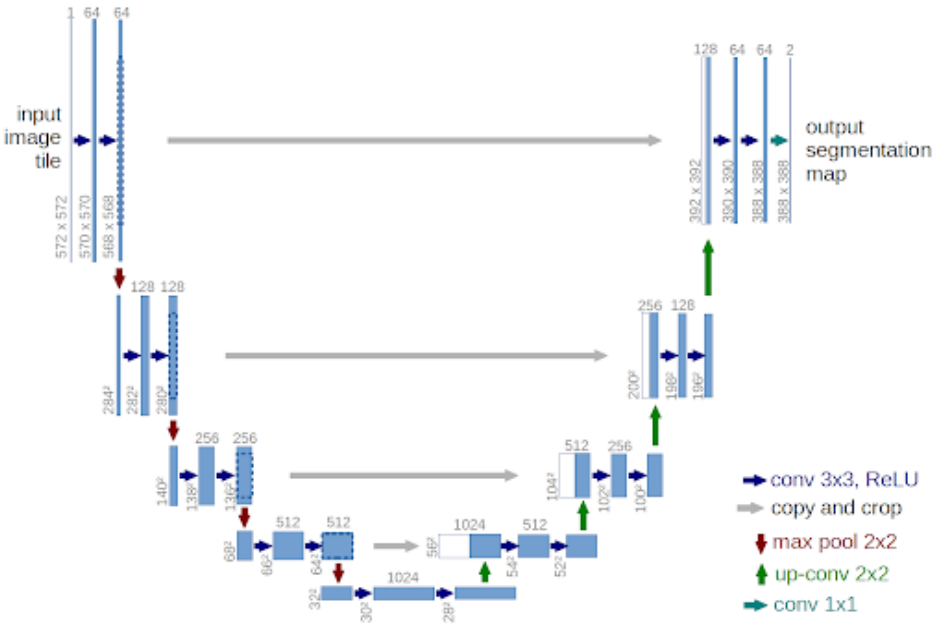

Figure 13 - UNet architecture.

The classification system proposed in this project consists of 2 step classification: (i) first step is a binary classification to predict if an image has a defect or not; (ii) second part is a multiclass classification to predict which class is the most probable. Figure 14 shows a diagram for the classification strategy in this work.

In order to assure a balanced dataset, I'm imposing that each defect class will contain exactly 180 images, randomly selected, consequently, entire defect dataset will contain 720 (4*180) unique images. Balance was required to avoid a prediction bias towards the class with most images in favor of the class with least images.

The binary classificator will work on 1.440 labeled images, split in, 720 with at least 1 defect plus 720 with no defect. The multiclass classificator will work on 720 images, split in, 180 per each class. Both classificator systems will be assisted by a dataloader that splits data in train-test-validation. The dataloader also performed data augmentation itself.

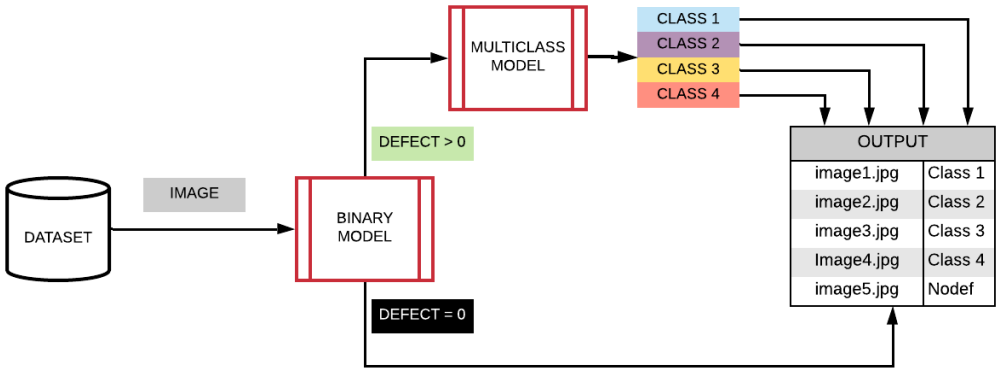

Figure 14 - Classification scheme.

Following the above schematic, the first classification system has the objective to identify if a given image has whether a defect or not. To achieve this goal, I'll be using a classification model very similar to UNet written in Keras, the input is an image reshape in length only from (256,1600,3) to (256,512,3). Inside the model you'll find many convolutional layers, max polling layers and dropouts.

The binary output was setup as a densenet with 1 output with sigmoid as activation function. However, the multiclass output was setup as a densenet with 4 outputs (one output for each single class) with sigmoid as activation function.

For the binary model, I've chosen Adam as a standard optimizer (default choice based on [4]), binary cross entropy (BCE) loss, because it's best suitable for problem involving binary classifications. For the multiclass model, I've chosen Adam as a standard optimizer (default choice), categorical cross entropy (CCE) loss, because it's best suitable for problem involving multiclass classification problems. For specific information with regard to each model, binary of multiclass, please refer to notebook files.

The model will be evaluated using a couple metrics that make sense for image classification, namely, accuracy, precision, recall and dice. All these metrics are explained below:

(i) Accuracy, after model is computed and testing dataset is performed, we can evaluate how many images were classified correctly over all images.

(ii) Precision, after model is computed and testing dataset is performed, we can evaluate how many images were classified with a certain feature (positive) over all tagged images (either correct or incorrectly tagged), focus on detect overzealous models.

(iii) Recall, after model is computed and testing dataset is performed, we can evaluate how many images were classified with a certain feature (positive) over all correct tagged images plus incorrect ones, focus on detect negligent models.

(iv) F1 score, is a parameter that can serve as a loss function, it represents the 2 times [precision] time [recall] over the sum of [precision] plus [recall].

Below, it's possible to see math formulation where: tp - true positive, tn - true negative, fp – false positive, fn - false negative.

Accuracy = (tp + tn) / (tp + fp + tn + fn)

Recall = tp / (tp + fn)

Precision = tp / (tp + fp)

F1_score = 2*Precision*Recall / (Precision + Recall)

In terms of metrics, accuracy, precision and recall are default choices in addition to F1_score. Model data will be exported to JSON file for further inspection and 30 epochs were defined as enough for this task. I've used Google Cloud (Colaboratory) to process this training job for free (K80 nvidia).

RESULTS

The binary classification system was trained with no issues using gpu K80 at Colab. The selected dataset could be bigger, in order to achieve a higher accuracy. As discussed above, a total of 1.440 unique images were used as input dataset in a universe of more than 12k. Due to this small selected pool, model's weight lack of 'knowledge' about each defect feature.

Figure 15 shows the evolution of loss (left) and F1-score (right) for the binary model, we can see that along the epochs loss is consistently reduced from ~0.7 to 0.1 and F1-score was increased from ~0.6 to near 0.95. However, validation data showed some huge spikes along epochs but without losing its convergence tendence.

Batch size showed an interesting interaction with metrics, since I've trained models using 8, 16 and 32 sizes. Batch size of 32 was the sweetspot providing consistent results. Batch size of 16 was similar, but worse, and 8 showed very poor results (both low accuracy and huge loss after all epochs) in accord to [5].

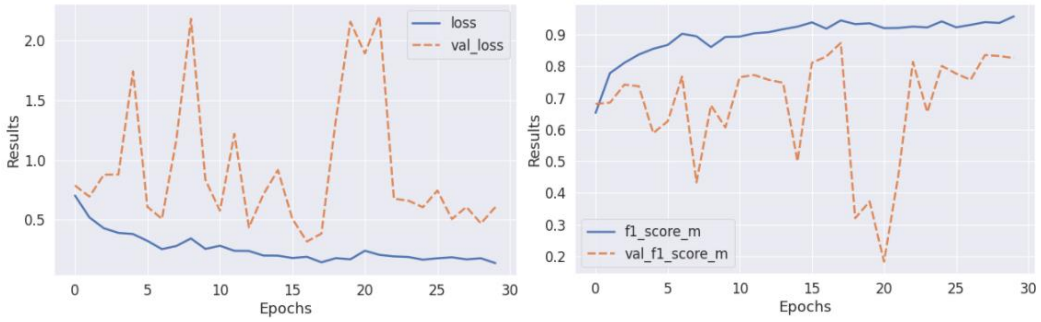

Figure 15 - Loss and F1-score evolution for binary model.

In terms of a training vs validation curves, we can see that 30 epochs were more than enough, both curves are still converging near epoch #25 so the model is not likely to be overfitted. All validation plots showed a very divergent value from training every 5 epochs, this is possibly due to data augmentation and image reshape that could've distorted defect features into no defect feature.

Figure 16 shows the results of model against training set (left) and test set (right) compiled into a table summarizing all metrics. The model was successful and got a good accuracy as well as F1-score.

binary_crossentropy	0.238817	**binary_crossentropy**	0.308860
acc	0.917954	**acc**	0.916667
f1_score_m	0.921498	**f1_score_m**	0.925556
precision_m	0.884169	**precision_m**	0.912857
recall_m	0.966085	**recall_m**	0.939231

Figure 16 - Binary model metrics results.

Finally, binary loss was ok, but could be improved if dataset has been tweaked with more images. This behavior is possibly due to small dataset that couldn't take into account all features available from classes, so the validation showed some divergence.

The multiclass classification system was trained with no issues using gpu K80 at Colab. The selected dataset could be bigger, in order to achieve a higher accuracy. As discussed above, a total of 720 unique images was used as input dataset in a universe of more than 12k, even smaller than binary classification. Due to this small selected pool, model's weight lack of 'knowledge' about each defect feature.

Figure 17 shows the evolution of loss (left) and F1 score (right) for the multiclass model, we can see that along the epochs loss is consistently reduced from ~0.5 to 0.3 and F1 score was increased from ~0.8 to near 0.9. However, validation data showed some huge spikes along epochs but without losing its convergence tendence.

In terms of a training vs validation curves, we can see that 30 epochs were more than enough, both curves were converged around epoch #25 so the model is likely to be a little overfitted post epoch #26-30. Validation loss presented a huge spike compared to training loss, but right after the curve got fitted again, this is behavior can be explained by the effect of data augmentation in small datasets. A similar spike was also observed in binary training around epoch #15-20.

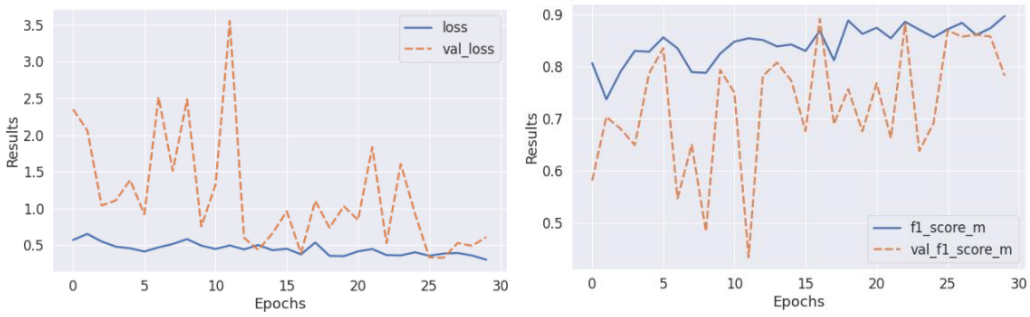

Figure 17 - Loss and F1-score for multiclass model.

The optimum batch size was found to be 16 providing consistent results. Either batch size of 8 or 32 provided low accuracy results with high accumulated loss after all epochs. This behavior was expected compared to binary classification that has the double number of images in dataset and optimum batch size of 32.

Figure 18 shows the results of model against training set (left) and test set (right) compiled into a table summarizing all metrics. The model was successful and got a good accuracy as well as F1-score.

categorical_crossentropy	0.281925	categorical_crossentropy	0.320426
acc	0.893822	acc	0.847222
f1_score_m	0.893985	f1_score_m	0.859759
precision_m	0.867077	precision_m	0.845915
recall_m	0.924242	recall_m	0.875000

Figure 18 - Multiclass model metrics results.

Important notice that categorical loss was very near binary loss, even using a smaller dataset, the reason for the change in batch size for both cases which compensated the dataset size. All validation plots showed a very divergent value from training every 5 epochs, this is possibly due to data augmentation and image reshape that could've distorted defect features into no defect feature.

FINAL REMARKS

The machine learning capstone project was successful because it comprised a real problem from steel industry as well as an end-to-end machine learning application. All the rubric items were included as intended. In terms of project we can highlight the following topics:

(a) Dataset was obtained from a reliable external source: Kaggle, it was selected after careful consideration
(b) Data exploration and cleaning was a simple but effective task
(c) Mask visualization were explored beyond single masks to binary masks with colors and black background for a better contrast
(d) Model search and selection were done accordingly for the classification task
(e) Good results were achieved after metrics calculation
(f) Final product was obtained: a classification method in 2 steps

For the next steps, I'd recommend training both model with a bigger volume data in a feasible way to check model consistency. I would also recommend trying to implement a segmentation model to predict the area, shape and localization of the defect.

REFERENCES

[1] Severstal: Steel Defect Detection. <https://www.kaggle.com/c/severstal-steel-defect-detection>

[2] Understanding Semantic Segmentation with UNET. <https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47>

[3] Generating masks from Encoded Pixels — Semantic Segmentation. <https://medium.com/analytics-vidhya/generating-masks-from-encoded-pixels-semantic-segmentation-18635e834ad0>

[4] Keras Optimizers. <https://keras.io/api/optimizers/adam/>

[5] Effect of batch size on training dynamics. <https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e>