

Leonardo Goldstein  
Machine Learning Engineer  
27 August 2018

# Capstone Project

## Bur classification on industrial production

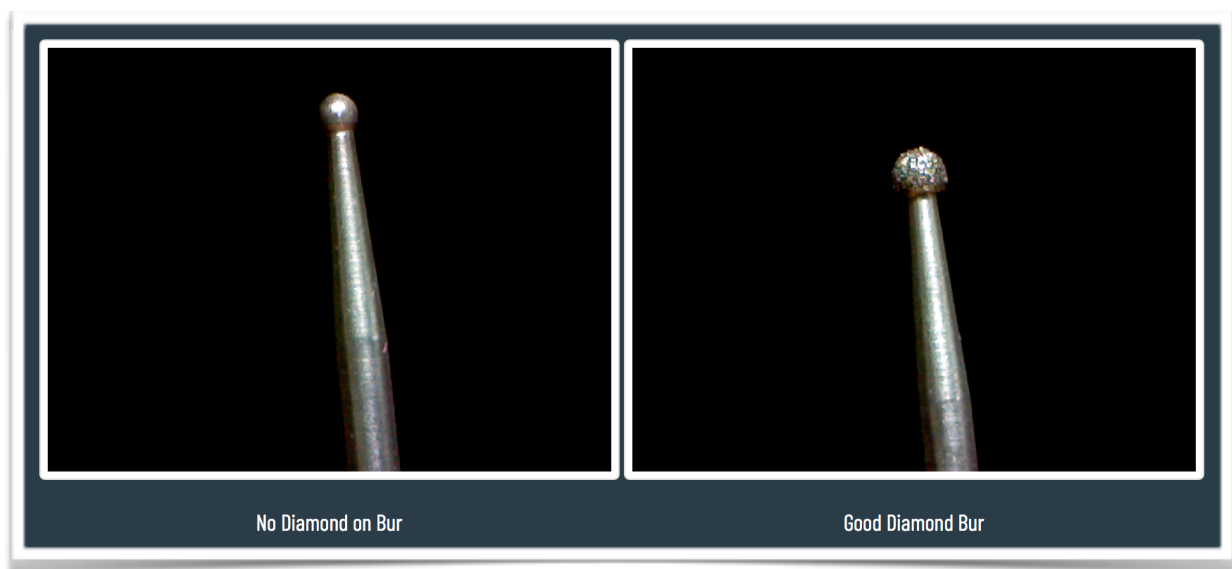
### I. Definition

#### Project Overview

During production there are always a necessity to control quality production steps to guarantee a good quality to the final product. In many cases, specially in Brazil those procedures are normally manual leading to human stress, low production and in many cases the result is not the best it could get.

The reason so far for being manual is the cost to automation. The idea of this project is to be able to create a cost effective solution.

The specific case being addressed is to classify Diamond dental burs during process. Before packing, it is necessary to verify if all burs are inside the quality range. For this project we will check if they got the diamond on head. If yes, pass, otherwise do not pass.



## Problem Statement

The burs have in average a 2 mm head where the diamonds are fixed thru an electrostatic deposition process. Being so small creates a difficulty for the human to classify, and for the project to create the dataset in a way that this dataset can be transposed to factory production.

We created a simple mechanism using python and Arduino to take pictures of burs each 120 degrees, resulting in a full 360.

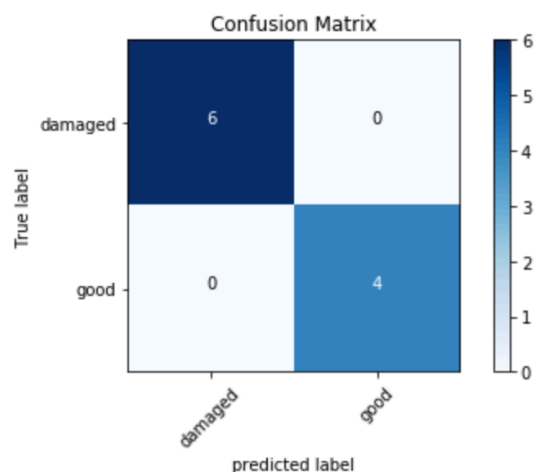
With the dataset created we will develop an CNN algorithm to classify the burs in two groups - goods ( with diamond) and damaged ( without diamonds). **We will also test in Watson Visual Recognition as a benchmark.**

## Metrics

To evaluate if our system is working as expected, we need to know how many burs were classified correctly over the total number of predictions. So we will use accuracy, which is exactly what we need.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

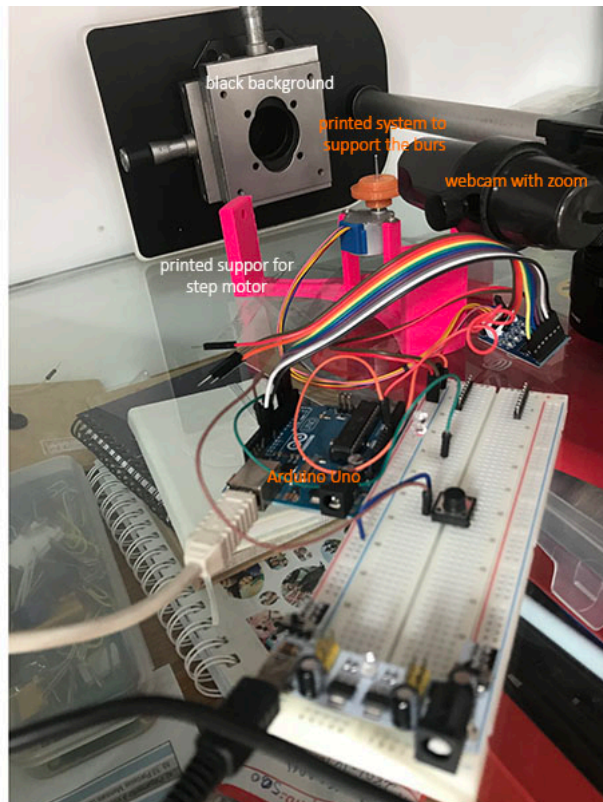
We will also plot a confusion matrix to better visualize our results.



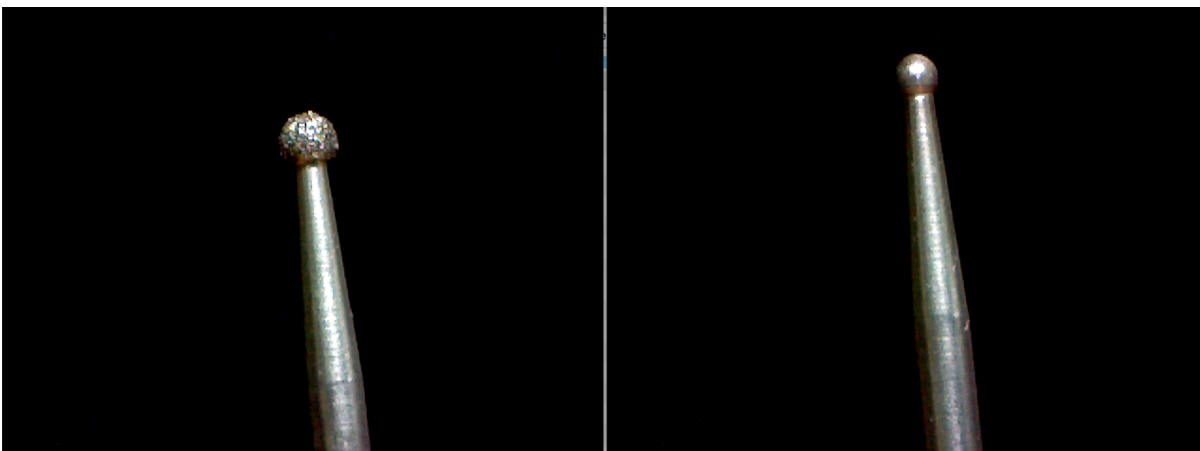
## II. Analysis

### Data Exploration

One of the main challenges of this project was to create the dataset. The burs are tiny and difficult to photograph. The idea is to develop the system in a way we can put this system on production with low cost and good performance.



The photos were taken under two classes “Good” and “Damaged”

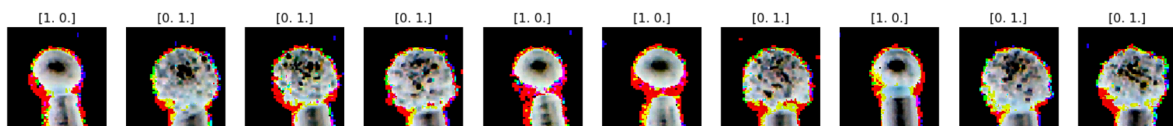


It was taken a total of 524 bur images and separated them in three datasets (train, validation and test)

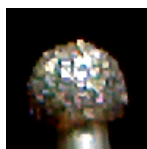
Found 431 images belonging to 2 classes. (train)  
Found 51 images belonging to 2 classes. (validation)  
Found 42 images belonging to 2 classes. (test)

The images were loaded in batches even this technique is commonly used for larger datasets or when memory constrain are faced. There are over 1 million burs check per month, I believe it is a good idea to be prepared.

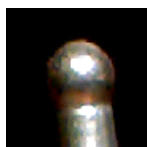
Since we are using categorical classes, the "damaged": 0 indicates that the damage images are on hot encoding index 0 ([1,0]); and "good": 1 indicates it is on the second position ([0,1]). It will be more clear on the predictions answers. In resume [1. 0.] is damaged and [0. 1.] is good.



The bur head is the main focus of interest, the algorithm will do its prediction as:



[0. 1.] - There is diamond fixed on the head



[1. 0.] - There is no diamond fixed on the head

## Algorithms and Techniques

Since every single image is different from each other, it makes impossible to do a simple comparison. For example if we get a "good" bur image and compare to our "good" bur reference, the computer will not classify as "good" since the images are actually different from each other.

To be able to classify we will create an algorithm that will analyse every pixel on the image and find patterns that are unique on each class.

In order to make it work we will build a [Convolutional Neural Network](#), which is the state-of-the-art algorithm for most image processing tasks, including classification.

In our case, it will classify under two classes “good” or “damaged”.

Some of the parameters choose on the CNN:

- ~ Optimizer: RMSprop (Tested with Adam and SGD)
- ~ Loss: categorical\_crossentropy (even there was only 2 categories we could use a binary, but we got better results with the categorical\_crossentropy).
- ~ Metrics: accuracy

## Benchmark

I have used Watson as benchmark. It is a undisclosed architecture probably for more complex problems. I got good results with less confidence than the CNN developed in this project. The results proved we could achieve better results with a local developed algorithm.

The front end to test is a benchmark from Stefani Mazon from a IBM course. You can find the NodeRed code at: <https://github.com/smazon/visual-recognition>

To test with Watson I set an instance of a custom visual recognition:

Cloud Foundry Applications					
Name ^	Region	CF Org	CF Space	Memory (MB)	Status
BurClasses	US South	leogold.br@gmail...	dev	256	● Running (1/1)

Cloud Foundry Services					
Name ^	Region	CF Org	CF Space	Plan	Service Offering
BurClasses-cloudantNoSQLDB →	US South	leogold.br@gmail...	dev	Lite	cloudantNoSQLDB

Services					
Name ^	Location	Resource Group	Plan	Details	Service Offering
Cloudant-es	US South	Default	Lite	Provisioned	Cloudant
Visual Recognition-sn	US South	Default	Lite	Provisioned	Visual Recognition
cloud-object-storage-dsx	global	Default	Lite	Provisioned	Cloud Object Stor...
data-science-experience-vx	US South	Default	Lite	Provisioned	Watson Studio

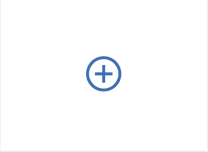
### Default Custom Model

Associated Service : Visual Recognition-sn

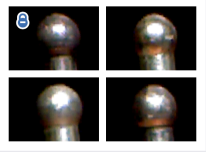
[My classes \(3\)](#) [All images \(524\)](#) [Train Model](#) [Trained](#)

Drag and drop files from your project.

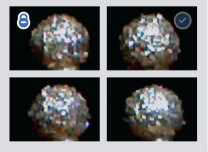
3 classes | 0 incomplete classes | 0 unclassified images New training data size: 0.0/250 MB



Create a class



damaged  
216 images



good  
308 images

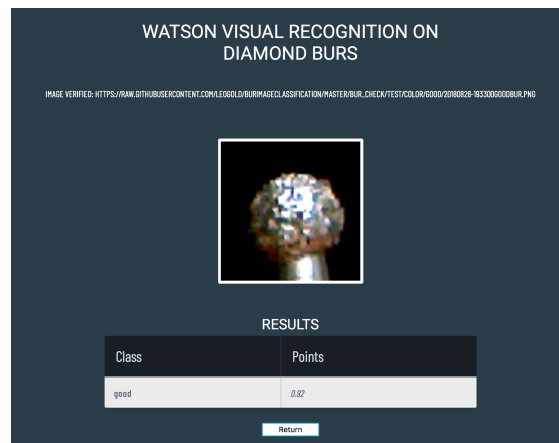
After training we can test the results at:

~ [https://burclasses.mybluemix.net/bur\\_classification](https://burclasses.mybluemix.net/bur_classification)

Links of images to try:

~ [https://raw.githubusercontent.com/leogold/BurImageClassification/master/Bur\\_check/test/color/damaged/20180826-193018DamagedBur.png](https://raw.githubusercontent.com/leogold/BurImageClassification/master/Bur_check/test/color/damaged/20180826-193018DamagedBur.png)

~ [https://raw.githubusercontent.com/leogold/BurImageClassification/master/Bur\\_check/test/color/good/20180826-193325GoodBur.png](https://raw.githubusercontent.com/leogold/BurImageClassification/master/Bur_check/test/color/good/20180826-193325GoodBur.png)



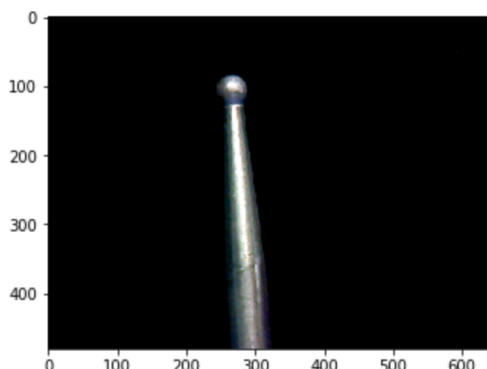
### III. Methodology

It took a while to get the algorithm working, after several attempts working on data and algorithm model I got the following:

#### Data Preprocessing

The initial idea was to take the images in a way to use them directly on the CNN.

```
plt.imshow(imagem_estudo)
plt.show()
```



It was taken images of 640 x 480 in color and gray depth. Always using the black background.

After some attempts with bad results and long training times, I decided to reduce the size of image doing some cropping and resizing.

```

from keras.preprocessing import image
from tqdm import tqdm

def save_image(img_path):
    ab = 0
    cd = 0
    img = cv2.imread(img_path)
    ab , cd = (find_pixel(img, 1, 1, 1))
    nova_img = separa_head(img, ab, cd)

    # Save images to check
    new_name = 'Bur_check' + img_path[10:]
    cv2.imwrite(new_name,nova_img,[cv2.IMWRITE_PNG_COMPRESSION, 9])

def save_images(img_paths):
    [save_image(img_path) for img_path in tqdm(img_paths)]

```

The optimal results were taken after implementing some code to extract only the bur head.

```

def find_pixel(img, r_query, g_query, b_query):
    # rgb = img.convert('RGB')
    for y in range(img.shape[0]):
        for x in range(img.shape[1]):
            r, g, b, = img[y,x]
            if r > r_query and g > g_query and b > b_query:
                return (y,x)

ab , cd = (find_pixel(image_estudo, 3, 3, 3))

```

First we try to locate the first pixel not black on the image going from the top to bottom. Avoiding some noise, I set up that all 3 channels should have a value bigger than 3.

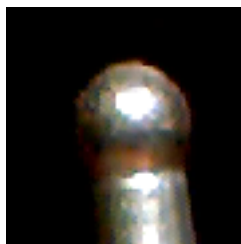
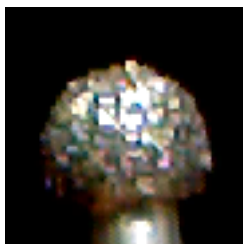
```

def separa_head(img, ab, cd):
    a = 0
    b = 0
    c = 0
    d = 0
    a = ab - 20
    b = ab + 70
    c = cd - 45
    d = cd + 45
    crop_img = img[a:b, c:d]
    return crop_img

```

After finding the pixel, it is time to crop and save the new image, creating a new dataset.

Adjusting to get 90 x 90 pixels.



## Implementation

The initial attempts did not get any result, the expectation was to easily get results, but not at all. First there were no learning at all with the algorithm guessing always 1 or always 0. After while I started to get excellent results on training (overfitting) but not at validation, getting results no better than before.

## Refinement

After testing several layouts, testing different kernel sizes (2x2, 3x3 and 5x5), different

Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 90, 90, 32)	2432
max_pooling2d_7 (MaxPooling2)	(None, 45, 45, 32)	0
conv2d_11 (Conv2D)	(None, 45, 45, 32)	25632
max_pooling2d_8 (MaxPooling2)	(None, 22, 22, 32)	0
dropout_9 (Dropout)	(None, 22, 22, 32)	0
conv2d_12 (Conv2D)	(None, 22, 22, 128)	102528
max_pooling2d_9 (MaxPooling2)	(None, 11, 11, 128)	0
dropout_10 (Dropout)	(None, 11, 11, 128)	0
dense_7 (Dense)	(None, 11, 11, 32)	4128
flatten_3 (Flatten)	(None, 3872)	0
dense_8 (Dense)	(None, 16)	61968
dropout_11 (Dropout)	(None, 16)	0
dense_9 (Dense)	(None, 2)	34
Total params: 196,722		
Trainable params: 196,722		
Non-trainable params: 0		

number of filters and sequences (16, 32, 64, 128), adding dropouts to avoid the previous overfitting, different number of convolutional layers (1, 2 and 3).

The add of a learning rate of 0.0001 lead me to starting get good results. The add of an EarlyStop helped also to prevent the overfitting, and to get a better control during training stopping after some epochs with no improvement. (tried several numbers, and also re-trained the model)

Even it is an small dataset, it was chosen not to use data augmentation since the the shape and size may be important in future improvement of this algorithm.



## IV. Results

### Model Evaluation and Validation

The model is performing good enough taking in account the small dataset and no use of data augmentation. It is performing well under the 3 datasets. (train, validation and test).

```
score = model.evaluate_generator(test_batches)
print("accuracy: ", score[1])

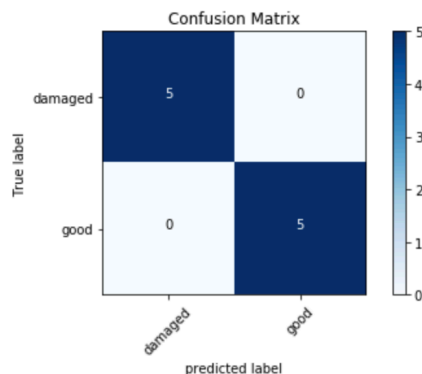
('accuracy: ', 0.9523809410276867)
```

Following there is a plot of a confusion matrix, where the batch performed on 100% accuracy.

```
cm_plot_labels = ['damaged', 'good']
plot_confusion_matrix(cm, cm_plot_labels, title= 'Confusion Matrix')
```

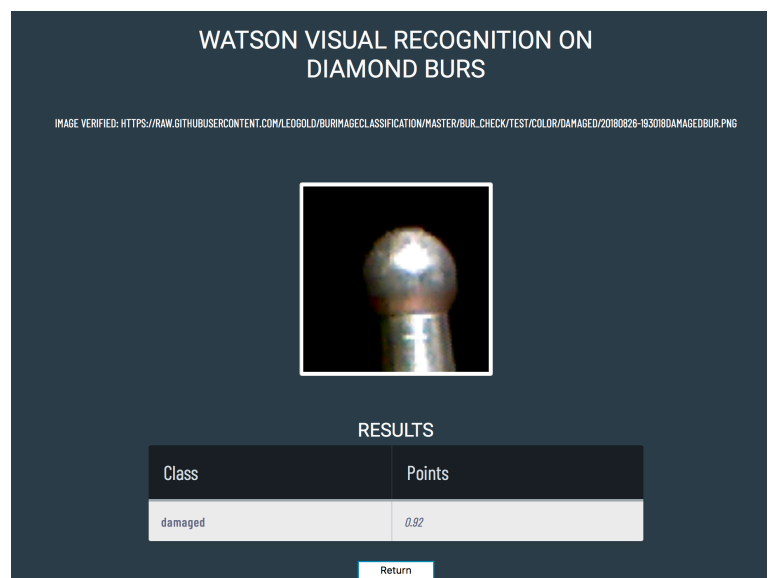
Confusion matrix, without normalization

```
[[5 0]
 [0 5]]
```



### Justification

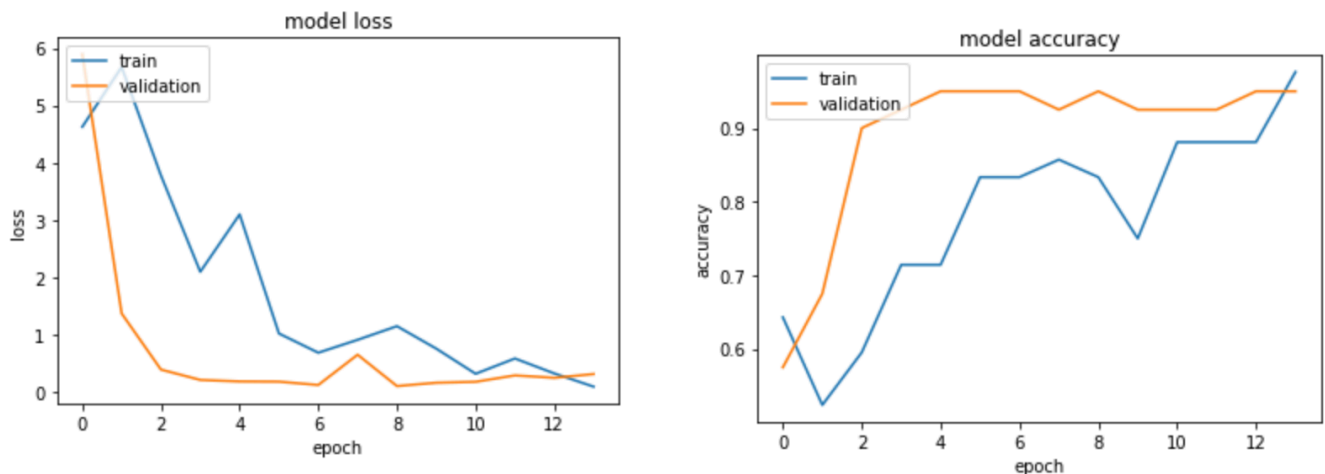
Comparing to Watson that is showing 92% confidence on the analysed images, the developed algorithm is delivering with 90 - 95% accuracy. Both are close, but more important is to have a local, lighter, cheaper solution for this problem.



## V. Conclusion

### Free-Form Visualisation

On the plots below we can see that there is a short training cycle to avoid past experience (overfit). It probably had happened due to the small dataset created for the project. I have run several times, and the graph is consistent as show below.



During the process of creating the dataset, I took several burs that were misclassified by the human actual system. If we create more classes and increase the number of images to train, we will achieve a much better quality control that is currently running on the plant. There is room for improve classification accuracy, processing time, process efficiency and cost reduction.

### Reflection

It was quite challenging to start getting usable results from the algorithm. There was a lot of guessing on building the model. I could not find a natural or mechanical way to build the model. It was based on others previous experiences and a lot of testing. Maybe after more practice it became more clear and easy to structure the model necessity.

After getting some good results, it became clear that there is a huge number of possibliteis to improve process on industry with affordable Visual Recognition systems.

### Improvement

During the project I came up with several challenges and new ideas of improvements. Instead manually finding the bur head, it would be a good improvement to the algorithm find

it and also classify the type of head, there are many other shapes and sizes (cylindrical, conical, wheel, etc.).

After this classification there are possibilities to not only classify the burs as “good” or “damaged” but also with the type of diamond (fine, medium or coarse) and also the type of damage (no diamond, too much diamond, distribution, other parts contamination, etc.)

There is plenty of room for improvement that will be for next project.

-----

#### Acknowledge

Thank you to Ricardo Ricci Lopes for giving valuable insights to address the challenges during the process.

Thank you to Stefany Mazon for the crash course and for sharing the code to build the front end to use Watson.