

Projet n°6 : “Reconnaissance d’Image Qualité Béton”

Soutenance de Jury - Projet n°6 - Décembre 2021
Ali Naama

Sommaire



I. Description de la problématique

II. Analyse du besoin

III. Construction du Modèle

IV. Résultats et interprétations

VI. Déploiement Web en local

VII. Conclusions et recommandations

I. Description de la problématique et exploration du jeu de données

Description de la problématique



Description :

Une grande entreprise française spécialisée dans la production et le développement des matériaux de construction : Ciment, Béton et Granulat, souhaiterait mettre au point un modèle de reconnaissance d'image permettant aux équipes qualité de pouvoir analyser des images provenant de client afin de détecter des défauts de qualité plus rapidement et efficacement.

Ce client dispose d'un outil CRM Salesforce et du module de gestion de réclamation nommé Service Cloud.

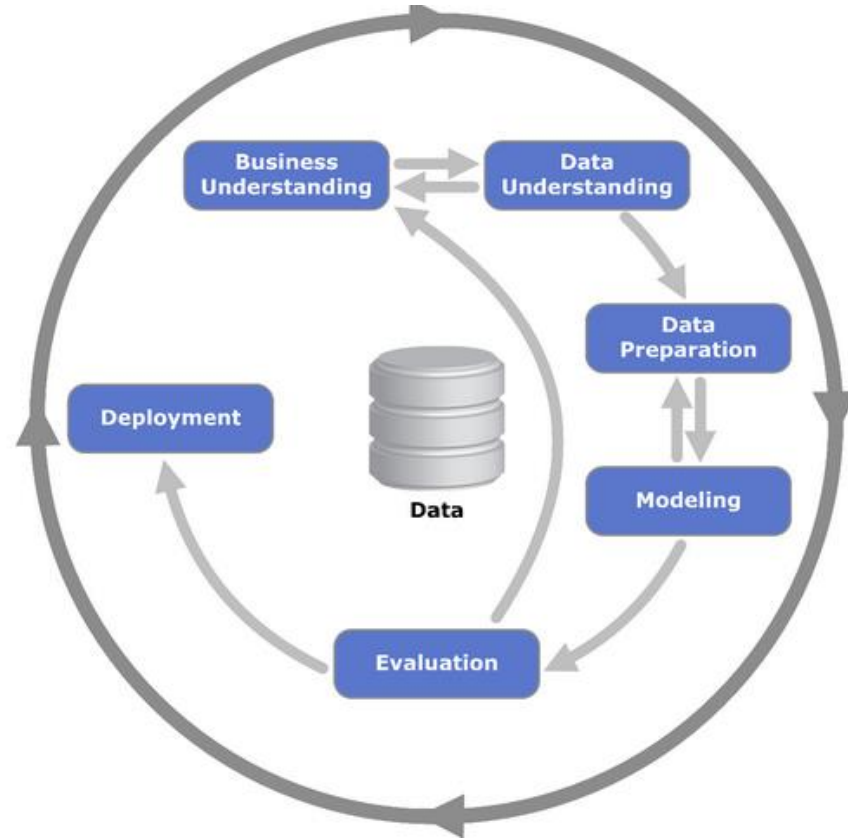
L'objectif final est de pouvoir intégrer ce module de reconnaissance d'image directement avec leur outil de gestion des réclamations qualité client.

Pour cela, le client nous transmet des photos correspondantes aux différents cas de non qualité sur les Bétons en particulier.

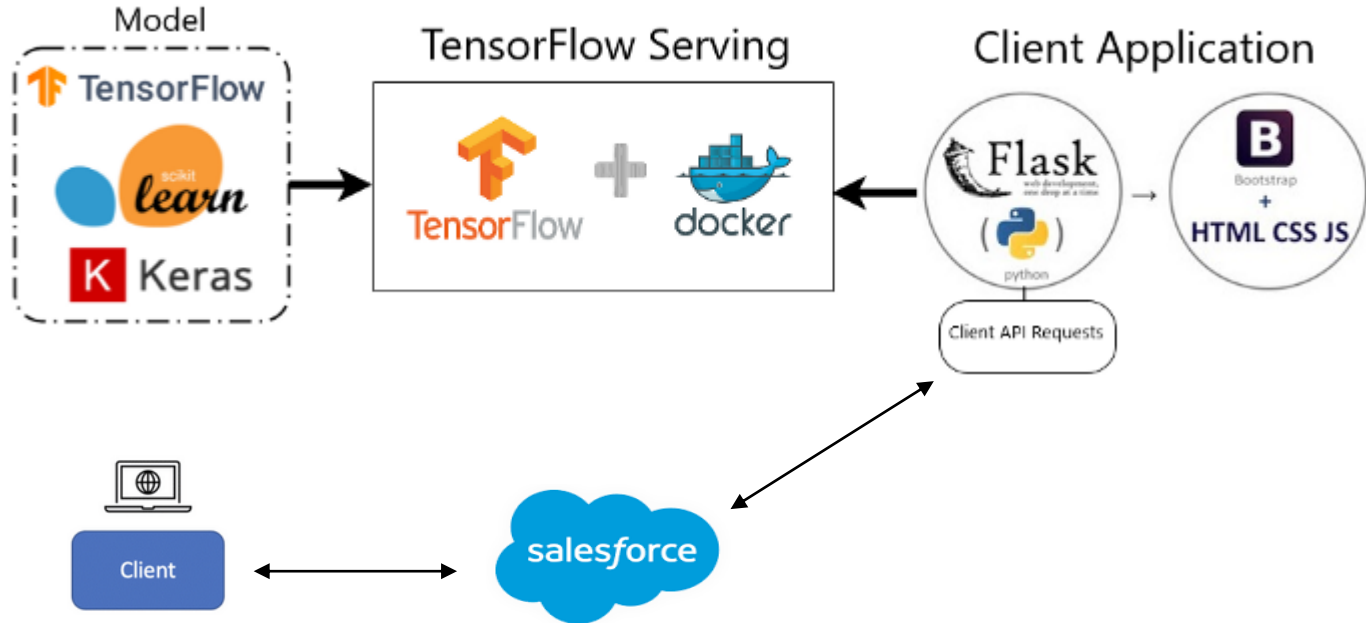
Objectifs :

A partir des données fournis, construire un modèle de reconnaissance d'image qui pourra être interfacé avec le CRM Salesforce pour fournir une reconnaissance des images prise sur le terrain avec un niveau de précision optimale.

Méthodologie de Data Science « CRISP DM »



Architecture du Projet





Informations sur les données :

- ☐ Les données sont fournis par le client au format jpg et sont au nombre de 12
- ☐ Pour chaque image, on dispose d'une étiquette correspondante au problème qualité
- ☐ La quantité d'images étant limité on complètera avec des images provenant d'Internet et d'images provenant de processus dit d'augmentation
- ☐ Nous allons devoir mettre en place un pipeline d'extraction d'image depuis Internet et une procédure de traitement pour l'augmentation

Présentation du jeu de données



Variations de teinte



Efflorescences.

Ce sont des dépôts irréguliers de teinte claire blanchâtre à la surface des bétons.



Ressuage.

Défaut caractérisé par des zones irrégulièrement érodées et par de petites rigoles verticales où le sable apparaît délavé



Tâches noires.

Elles sont constituées par des zones très sombres de formes irrégulières. On a l'impression que la pâte de ciment y est très compacte, sans aucun pore.



Pommelages.

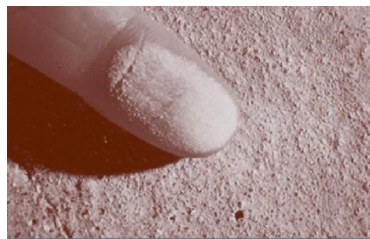
C'est un phénomène de transparence des gros granulats à la surface du béton



Traces de rouille.

ce sont des tâches d'hydroxyde ferrique provenant de la corrosion des armatures, des fils de ligature d'armature ou parfois des granulats contenant des sulfures de fer.

Présentation du jeu de données



Empoussièrement.

Il s'identifie lorsque la surface du béton se raye au contact d'un ongle et lorsque de la poussière y apparaît sous l'action de la circulation ou du balayage



Nids de cailloux.

Le manque de fines et de mortier laisse les graviers apparents.



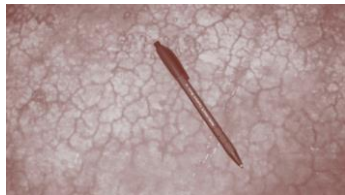
Fuites de laitances

Elles se manifestent généralement au droit des joints par des taches sombres, des nids de graviers, des coulures de laitance, évidemment à la base des pièces.



Soufflures/ bullage de surface.

Elles sont caractérisée par la présence de cavités isolées sensiblement hémisphériques (= bulles)



Faiénçage.

Il est reconnaissable à la formation sur la dalle de fissures très minces, dont la dimension moyenne varie de 10 mm à 40 mm. Ces fissures se présentent en réseaux plus ou moins hexagonaux.



Ecaillage.

L'écaillage est un phénomène de désagrégation des surfaces de béton provoqué par leur exposition au gel/dégel en présence d'humidité ou de sels déglçants.

Classe d'image à gérer



❑ On dispose de 12 classes

❑ On choisit de traduire en anglais

❑ On trouve des noms simplifié au répertoire des images

Classe	Traduction	Directory Name
Variations de teinte	Shade variations	shade
Efflorescences	efflorescence	efflorescence
Ressuage	penetrant testing	ressuage
Tâches noires	Black stains	Black
Pommelages	Pommeling	pommeling
Traces de rouille	Traces of rust	Rust
Empoussièrement	dustiness	dustiness
Nids de cailloux	Stone nests	stone
Fuites de laitances	Milt leaks	milt
Soufflures bullage de surface	Surface bubbling blowholes	bubbling
Faïençage	cracking	cracking
Ecaillage	Flaking	flaking

Présentation du jeu de données



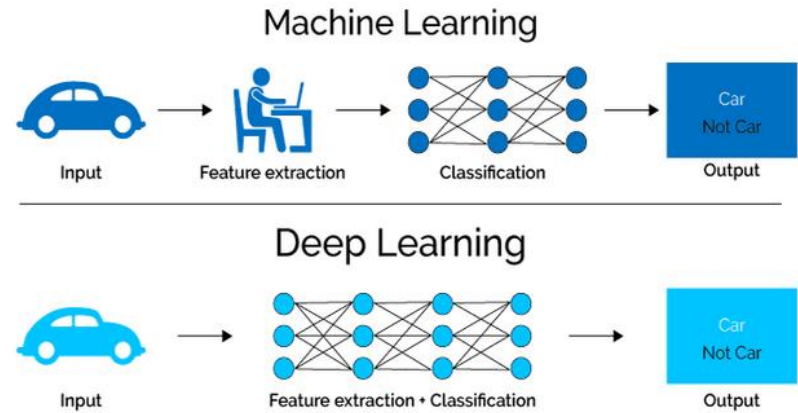
Classe	Directory Name	Train (nb Images)	Test (nb Images)
Variations de teinte	shade	1	1
Efflorescences	efflorescence	1	1
Ressuage	ressuage	3	1
Tâches noires	Black	5	2
Pommelages	pommeling	4	1
Traces de rouille	Rust	3	1
Empoussièrement	dustiness	5	1
Nids de cailloux	stone	3	1
Fuites de laitances	milt	2	1
Soufflures bullage de surface	bubbling	4	1
Faïençage	cracking	2	1
Ecaillage	flaking	3	1

❑ Le nombre d'images est limité mais nous allons remédier à cela via les techniques d'augmentation

Outils utilisés pour l'analyse

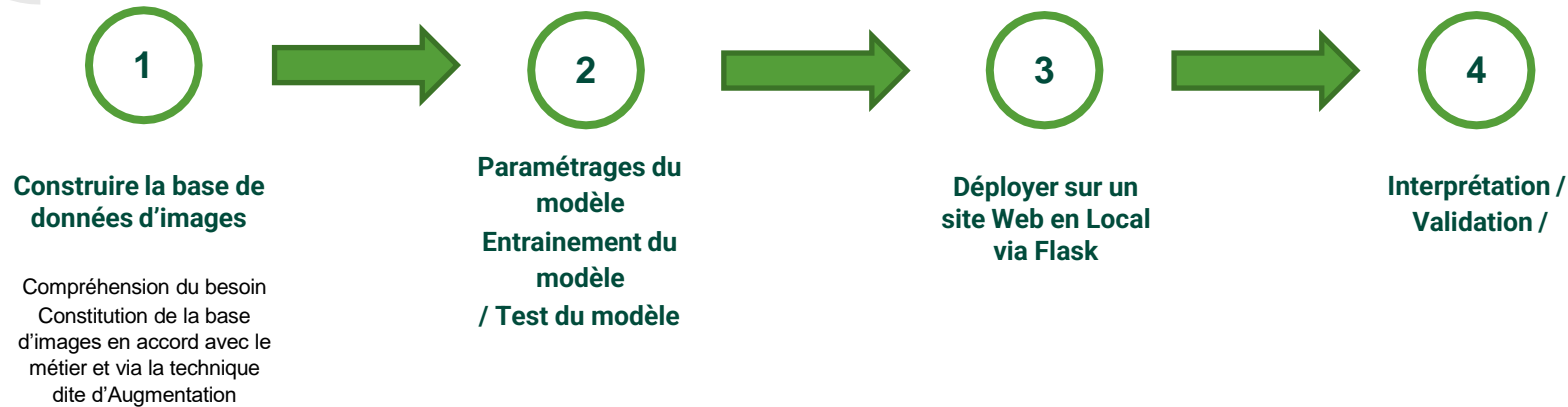


Nom	Utilisation	Fonctions spécifiques
PyCharm 2021.1	Test et développement	IDE Community Edition, Debug, Synchro Git
Python 3.9.5	Moteur Python Gestionnaire de librairies	Moteur d'exécution
Cv2 « OpenCV », PIL	OpenCV : Prend également en charge l'exécution de modèles pour Machine Learning (ML) Pillow est une bibliothèque de traitement d'image, qui est un fork et successeur du projet <u>PIL</u> (Python Imaging Library).	Elle est conçue de manière à offrir un accès rapide aux données contenues dans une image, et offre un support pour différents formats de fichiers tels que PPM, PNG, JPEG, GIF, TIFF et BMP.
Tensorflow Keras	Librairie de Machine Learning	Une plate-forme d'apprentissage automatique open source
Matplotlib 3.5.1 Numpy 1.22.0	Génération de graphiques Gestion des densités de probabilité	Barplot, Scatterplot, lineplot, distplot, heatmap Calcul statistique

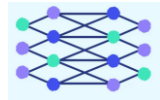


II. Analyse du besoin

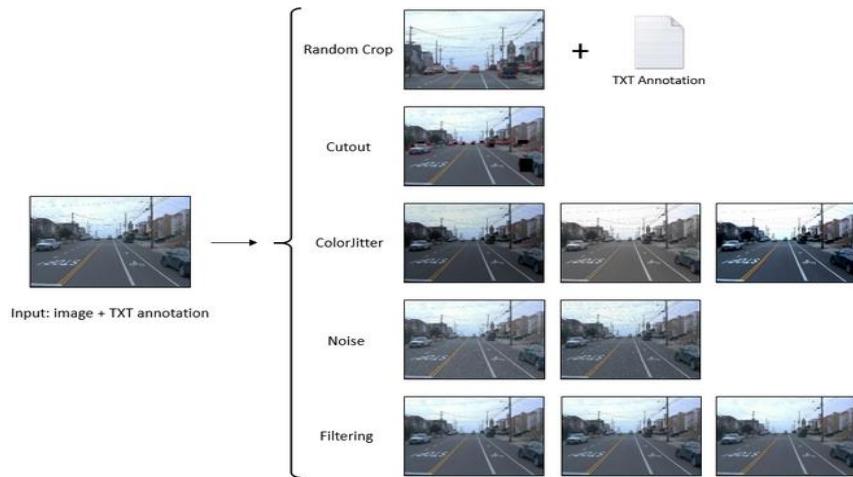
Stratégie d'analyse



Augmentation de Données



- L'augmentation des données est une technique permettant d'augmenter la diversité des ensembles de données (dans mon cas des données de type images) pour collecter davantage de données réelles, pour aider à améliorer la précision du modèle et à empêcher le surajustement du modèle (« overfitting ») ou au contraire le sous-ajustement « underfitting ».
- Les procédures d'augmentation de données les plus efficaces pour la tâche de détection d'objets en utilisant Python et la librairie OpenCV sont les suivantes :
 - *Random Crop (Recadrage aléatoire)*
 - *Cutout (Découpage)*
 - *ColorJitter*
 - *Adding Noise (Ajouter du bruit)*
 - *Filtering*



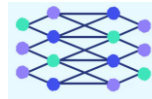
Augmentation de Données – Mise en œuvre

Pour chaque image, je vais appliquer ces 5 transformations pour augmenter le jeu de données :

- Random Crop, Cutout
- ColorJitter : Brightness, Saturation, Contrast : 3 modes
- Noise bruit : Gaussien et Sp : 2 modes
- Filtering : Blur, Gaussien, Median : 3 modes

Classe	Directory Name	Train (nb Images)	Test (nb Images)	Random Crop (nb Images)	Cutout (nb Images)	ColorJitter (3) (nb Images)	Adding Noise (2) (nb Images)	Filtering (3) (nb Images)	Total	Facteur de Gain
Variations de teinte	shade	1	1	2	2	6	4	6	20	10
Efflorescences	efflorescence	1	1	2	2	6	4	6	20	10
Ressuage	ressuage	3	1	4	4	12	8	12	40	10
Tâches noires	Black	5	2	7	7	21	14	21	70	10
Pommelages	pommeling	4	1	5	5	15	10	15	50	10
Traces de rouille	Rust	3	1	4	4	12	8	12	40	10
Empoussièrement	dustiness	5	1	6	6	18	12	18	60	10
Nids de cailloux	stone	3	1	4	4	12	8	12	40	10
Fuites de laitances	milt	2	1	3	3	9	6	9	30	10
Soufflures bullage de surface	bubbling	4	1	5	5	15	10	15	40	10
Faïençage	cracking	2	1	3	3	9	6	9	30	10
Ecaillage	flaking	3	1	4	4	12	8	12	40	10

Augmentation de Données – Mise en œuvre



- ❑ L'augmentation des données est réalisé via un script Python utilisant la librairie CV2
- ❑ Le principe est de lire une image et de la transformer selon les différents types de transformations :
 - ❑ Cutout
 - ❑ ColorJitter
 - ❑ Adding Noise
 - ❑ Filtering
 - ❑ Random Crop
- ❑ Les libraires utilisés pour ce traitement :

```
import os
import cv2
import numpy as np
import random
import time
```

```
# Cutout
cutout = cutout(image, gt_boxes, amount=0.5)
cv2.imwrite(f"{dir_path}/{img_name}_cutout.jpg", cutout)

# ColorJitter
b_img = colorjitter(image, cj_type="b")
s_img = colorjitter(image, cj_type="s")
c_img = colorjitter(image, cj_type="c")
cv2.imwrite(f"{dir_path}/{img_name}_brightness.jpg", b_img)
cv2.imwrite(f"{dir_path}/{img_name}_saturation.jpg", s_img)
cv2.imwrite(f"{dir_path}/{img_name}_contrast.jpg", c_img)

# Adding Noise
gaussn_img = noisy(image, noise_type="gauss")
sp_img = noisy(image, noise_type="sp")
cv2.imwrite(f"{dir_path}/{img_name}_gaussnoise.jpg", gaussn_img)
cv2.imwrite(f"{dir_path}/{img_name}_spnoise.jpg", sp_img)

# Filtering
blur_img = filters(image, f_type="blur")
gaussf_img = filters(image, f_type="gaussian")
median_img = filters(image, f_type="median")
cv2.imwrite(f"{dir_path}/{img_name}_blur.jpg", blur_img)
cv2.imwrite(f"{dir_path}/{img_name}_gaussblur.jpg", gaussf_img)
cv2.imwrite(f"{dir_path}/{img_name}_median.jpg", median_img)

# Random Crop
rancrop, new_boxes = randomcrop(image, gt_boxes, scale=0.5)
cv2.imwrite(f"{dir_path}/{img_name}_rancrop.jpg", rancrop)
filepath = f"{dir_path}/{img_name}_rancrop.txt"
write_anno_to_txt(new_boxes, filepath)
```

Augmentation de Données – Mise en œuvre

- ❑ A partir d'une image fournie en entrée du traitement, j'obtiens une démultiplication d'un nombre d'image avec pour chacune une transformation associée :

DatasetP6 > Train > black



black



black0



black0_blur



black0_brightness



black0_contrast



black0_cutout



black0_enhanced



black0_gaussblur



black0_gaussiannoise



black0_median



black0_rancrop



black0_rancrop



black0_saturation



black0_snoise



black001



black1



black1_blur



black1_brightness



black1_contrast



black1_cutout



black1_enhanced



black1_gaussblur



black1_gaussiannoise



black1_median



black1_rancrop



black1_rancrop



black1_saturation



black1_snoise



black2



black2_blur



black2_brightness



black2_contrast



black2_cutout



black2_enhanced



black2_gaussblur



black2_gaussiannoise



black2_median



black2_rancrop



black2_rancrop



black2_saturation



black2_snoise



black3



black3_blur



black3_brightness



black3_contrast



black3_cutout



black3_enhanced



black3_gaussblur



black3_gaussiannoise



black3_median



black3_rancrop



black3_rancrop



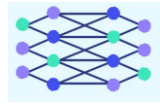
black3_saturation



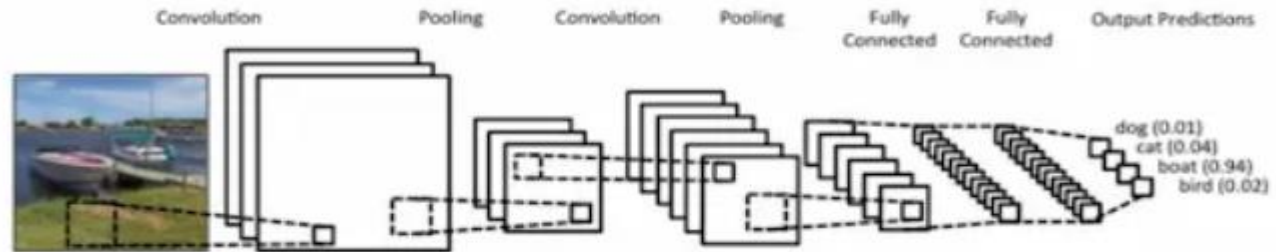
black3_snoise

III. Construction du Modèle

Modèle de type CNN « Deep Learning »

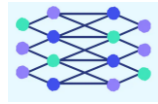


- ❑ Les étapes de réalisations du modèle sont les suivantes :
 - ❑ A partir d'un exemple de projet similaire qui fonctionne, j'ai testé et adapter ce modèle afin de le faire correspondre à notre modèle.
 - ❑ L'inconvénient majeur demeure le peu de données disponible à date qui est compensé par le procédé d'augmentation.
 - ❑ Le principe d'un réseaux de neurone convolutionnel est le suivant : Analyser les images, Réduction de la taille, Remplacement des valeurs négatives par des 0, Réduction au nombre de classe du vecteurs en sortie avec une probabilité d'appartenance à la classe.
 - ❑ Pour rappel nous avons 12 classes correspondante à notre catégorisation de défaut de qualité.



Exemple d'architecture d'un CNN

Modèle de type CNN « Deep Learning »

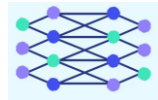


❑ Utilisation des libraires de Machine Learning (Tensor flow / keras) et de gestion d'image (cv2) :

```
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.optimizers import RMSprop
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
import matplotlib.pyplot as plt
import os, sys

# Description des classes du modèle
CLASS_NAMES = ['shade', 'efflorescence', 'ressuage', 'Black', 'pommeling', 'Rust', 'dustiness', 'stone', 'milt', 'bubbling', 'cracking', 'flaking']
```

Modèle de type CNN « Deep Learning »



❑ Construction du modèle :

```
# Description des classes du modèle
CLASS_NAMES = ['shade', 'efflorescence', 'ressuage', 'Black', 'pommeling', 'Rust', 'dustiness', 'stone', 'milt', 'bubbling', 'cracking', 'flaking']

train_datagen = ImageDataGenerator(rescale = 1./255,
                                    shear_range = 0.2,
                                    zoom_range = 0.2,
                                    horizontal_flip = True)

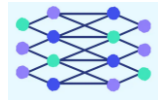
test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory(r'DatasetP6\train',
                                                target_size = (64, 64),
                                                batch_size = 32,
                                                classes = CLASS_NAMES ,
                                                class_mode = 'categorical')

test_set = test_datagen.flow_from_directory(r'DatasetP6\test',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            classes = CLASS_NAMES ,
                                            class_mode = 'categorical')

# Adaptation des paramètres : Nombre de classes 12
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (12,12), activation='relu', input_shape=(64, 64, 3)),
    tf.keras.layers.MaxPooling2D(8, 8),
    tf.keras.layers.Conv2D(32, (4,4), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(12, activation='softmax')
])
```

Modèle de type CNN « Deep Learning »



□ Entraînement du modèle : Utilisation du jeu de données d'entraînement

```
# Adaptation des paramètres : Nombre de classes 12
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (12,12), activation='relu', input_shape=(64, 64, 3)),
    tf.keras.layers.MaxPooling2D(8, 8),
    tf.keras.layers.Conv2D(32, (4,4), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(12, activation='softmax')
])

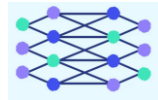
model.summary()

opt = RMSprop(lr=0.001)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

r = model.fit_generator(training_set,
                        steps_per_epoch = 120,
                        epochs = 25,
                        validation_steps = 36)
```

V. Résultats et interprétations

Résultats et interprétations



- ❑ Il est intéressant de visualiser les précisions de notre réseau de neurones :
- ❑ Tester une image qui appartient à la classe : Black, soit le quatrième élément du vecteur de restitution.

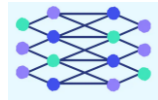
```
CLASS_NAMES = ['shade', 'efflorescence', 'ressuage', 'Black', 'pommeling', 'Rust', 'dustiness', 'stone', 'milt', 'bubbling', 'cracking', 'flaking']
```



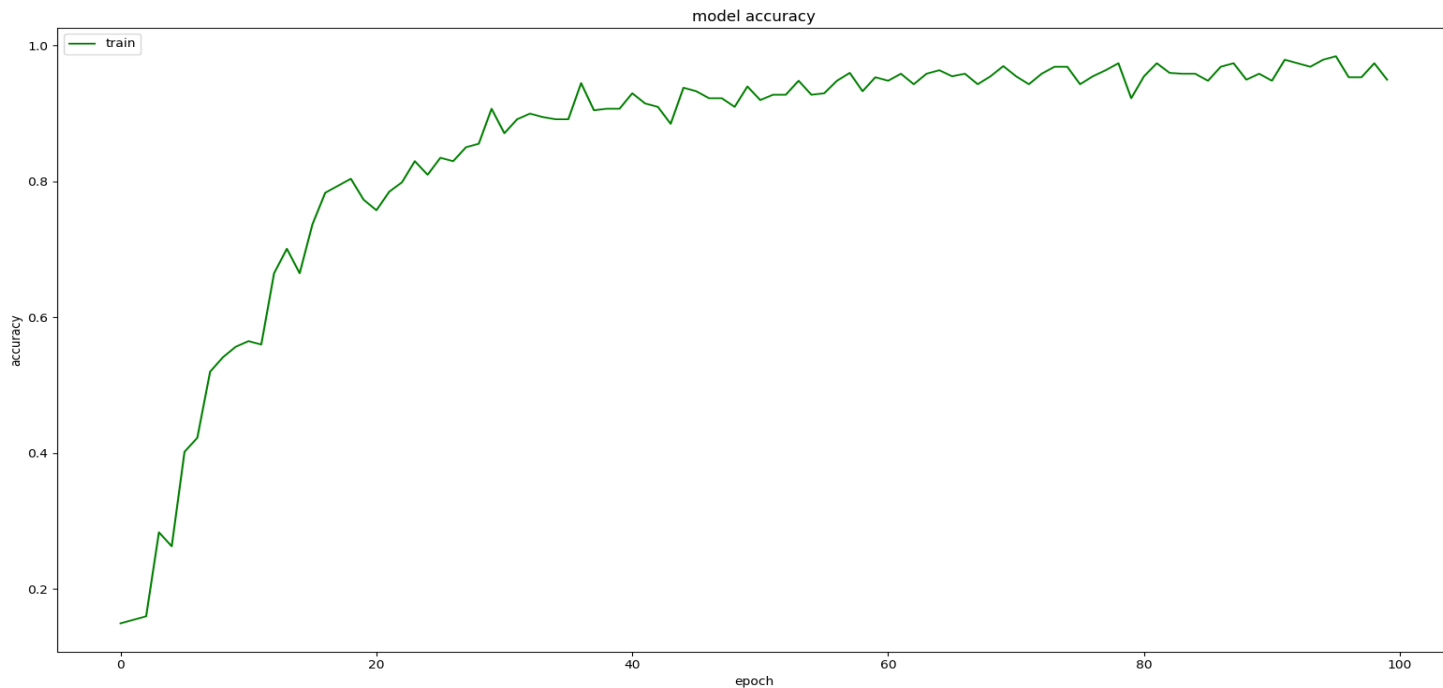
- ❑ L'image a été formellement reconnu avec une probabilité de 100 % ce qui est excellent !

```
[[0.00000000e+00 0.00000000e+00 0.00000000e+00 1.00000000e+00 1.0401200e-32  
5.4674009e-29 1.7949565e-36 4.8002928e-28 0.00000000e+00 3.6283435e-08  
0.00000000e+00 0.00000000e+00]]
```

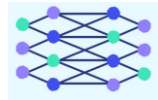
Résultats et interprétations



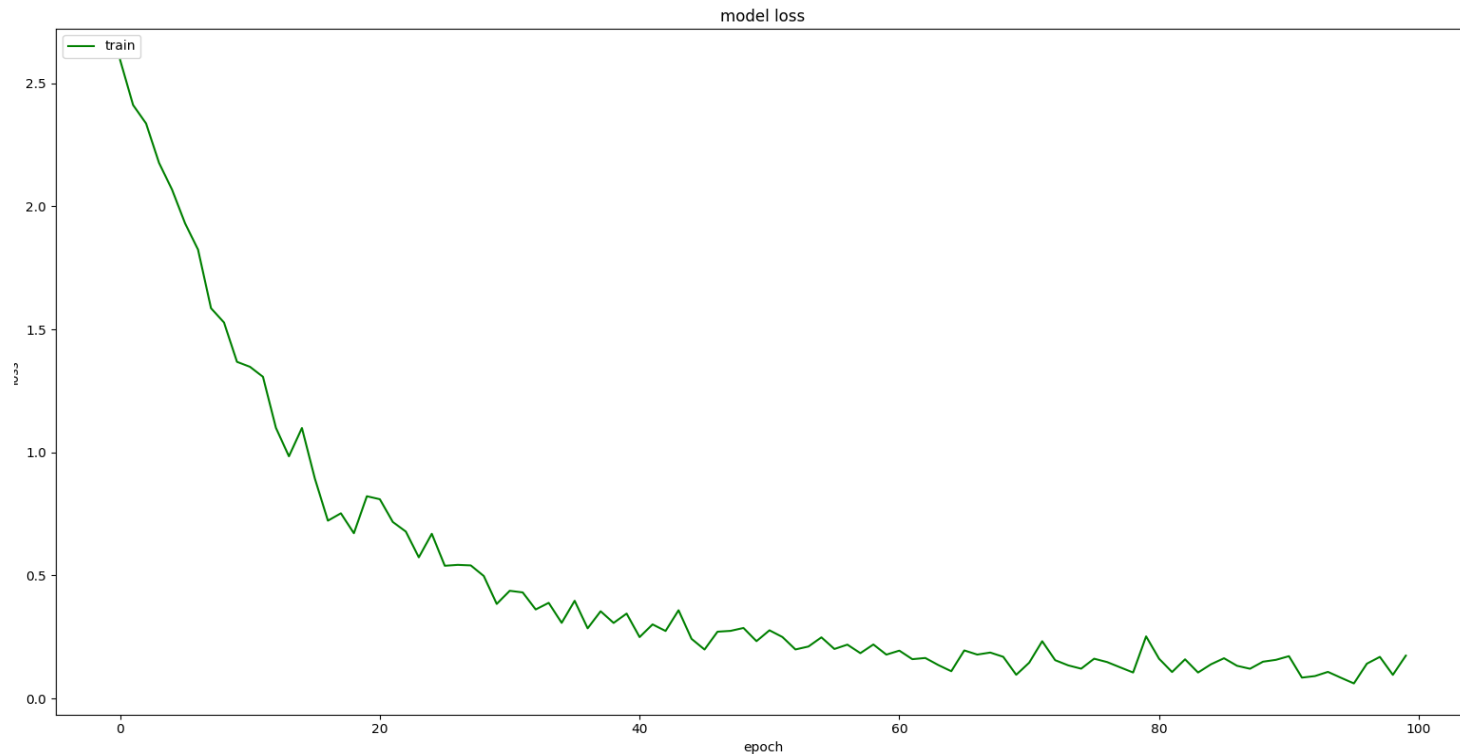
- ❑ Il est intéressant de visualiser les précisions de notre réseau de neurones :
- ❑ La courbe suivante donne une vision de la précision du modèle en fonction du nombre d'itération



Résultats et interprétations

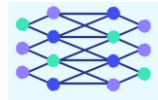


- Il est intéressant de visualiser les précisions de notre réseau de neurones à travers la courbe des erreurs de précisions en fonction du nombre d'itération :



V. Déploiement Web Flask


Déploiement Web en Local



- ❑ Une fois que notre modèle est compilé et sauvegardé je vais le publier via une application Web en local en utilisant la librairie Flask :

```
Projet6LF.py x P6App.py x
1 import os
2 import uuid
3 import flask
4 import urllib
5 import numpy as np
6 from PIL import Image
7 from tensorflow.keras.models import load_model
8 from keras.preprocessing import image
9 from flask import Flask, render_template, request, send_file
10 from tensorflow.keras.preprocessing.image import load_img, img_to_array
11
12 app = Flask(__name__)
13 BASE_DIR = os.path.dirname(os.path.abspath(__file__))
14 pathmdl = str(BASE_DIR) + '\p6.model'
15 print(BASE_DIR)
16 print(pathmdl)
17 model = load_model(pathmdl)
```

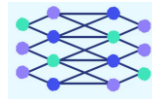
Uploaded Image



Model Prediction

Rank	Class	Probability
1st	shade	0.0 %
2nd	efflorescence	0.0 %
3rd	ressuage	0.0 %
4rd	Black	0.0 %
5th	pommeling	0.0 %
6th	Rust	0.0 %
7th	dustiness	0.0 %
8th	stone	0.0 %
9th	milt	0.0 %
10th	bubbling	0.0 %
11th	cracking	0.0 %
12th	flaking	100.0 %

Déploiement Api Flask



- ❑ Une fois que mon modèle est compilé et sauvegardé je vais le publier via une API en local en utilisant la librairie Flask : Ici un exemple d'appel api et la définition de l'Api dans le code de l'Application :

```
P6App.py x P6ReqApi.py x
1 import requests
2 import numpy as np
3 from PIL import Image
4
5 KERAS_REST_API_URL = "http://localhost:5000/predictapi"
6 IMAGE_PATH = "DatasetP6/Test/black/Tachesnoires3.jpg"
7
8 # load the input image
9 image = open(IMAGE_PATH, "rb").read()
10
11
12 # construct the payload for the request
13 payload = {"image": image}
14
15 # submit the request
16 req = requests.post(KERAS_REST_API_URL, files=payload).json()
17 print(req)

```

```
@app.route("/predictapi", methods=["POST"])
def predictapi():
    # ensure an image was properly uploaded to our endpoint
    if flask.request.method == "POST":
        if flask.request.files.get("image"):
            img = flask.request.files["image"].read()
            img = Image.open(io.BytesIO(img))
            #image = prepare_image(image, target=(64, 64))
            if img.mode != 'RGB':
                img = img.convert('RGB')
            img = img.resize((64, 64))
            img = image.img_to_array(img)
            img = np.expand_dims(img, axis=0)
            inputs = preprocess_input(img)
            #predictions = model.predict(inputs)
            predictions = model.predict(inputs)
            print(predictions)
            b = predictions.tolist() # nested lists with same data, indices
            json_predictions = json.dumps(b)

            # return the results as a JSON response
            return flask.jsonify(json_predictions)

```

```
C:\Users\Utilisateur\PycharmProjects\pythonProject\venv\Scripts\python.exe "C:/Users/Utilisateur/PycharmProjects/pythonProject/venv/Scripts/Data Science VAE/P6ReqApi.py"
[[1.0595426476616964e-30, 0.0, 1.1707454711944005e-31, 0.999998927116394, 4.363134419732573e-12, 1.1166297326781205e-06, 8.667215663687365e-17, 4.445335254829244e-21, 1.608934015459005e-33, 2.882206929876573e-19, 7.349906786872
Process finished with exit code 0

```

- ❑ Résultat du modèle de prédiction au format json, lisible par n'importe quelle application, Salesforce en particulier


Augmentation de Données – Test



- ❑ Pour chaque image, je vais vérifier le résultat de reconnaissance du modèle avec Augmentation et le comparer au modèle sans Augmentation afin d'évaluer le gain en performance :
- ❑ Je constate qu'avec l'augmentation, j'obtiens un taux de reconnaissance de **75 %** (9 images / 12 reconnues) contre 2 images reconnues sur 12 sans augmentation (16,66 %)

Classe	Directory Name	Test avec Modèle sans Augmentation	Test Modèle avec Augmentation
Variations de teinte	shade	KO	KO
Efflorescences	efflorescence	KO	OK
Ressuage	ressuage	KO	OK
Tâches noires	Black	OK	OK
Pommelages	pommeling	KO	OK
Traces de rouille	Rust	OK	OK
Empoussièrement	dustiness	KO	KO
Nids de cailloux	stone	KO	OK
Fuites de laitances	milt	KO	KO
Soufflures bullage de surface	bubbling	KO	OK
	cracking	KO	OK
	flaking	KO	OK

Uploaded Image



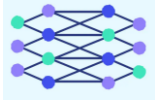
Model Prediction

Rank	Class	Probability
1st	shade	0.0 %
2nd	efflorescence	0.0 %
3rd	ressuage	0.0 %
4th	Black	0.0 %
5th	pommeling	0.0 %
6th	Rust	0.0 %
7th	dustiness	0.0 %
8th	stone	0.0 %
9th	milt	0.0 %
10th	bubbling	0.0 %
11th	cracking	0.0 %
12th	flaking	100.0 %

Reconnaissance d'une image illustrant un phénomène d'écaillage

VI. Conclusion et recommandations

Conclusion



- ❑ Cette étude / POC (Proof of Concept) a permis l'utilisation de méthodes de reconnaissance d'images basée sur l'utilisation des librairies Python : Keras, tensorflow et Flask.
- ❑ J'ai pu générer un modèle de prédiction et réaliser une amélioration de la précision grâce à la méthode d'Augmentation.
- ❑ Avec un taux de reconnaissance de 75%, il est possible de l'améliorer encore plus en disposant de photos plus précises pour les 3 catégories non reconnues : Shade, Dustiness, Milt.
- ❑ Les courbes de précisions du modèle : Précision et Erreur sont assez régulières ce qui traduit un bon niveau de prédiction
- ❑ Le modèle a pu être déployé en local pour des premiers tests en utilisant une application Web en local et via une API.
- ❑ En terme d'amélioration du modèle, il faudra faire évoluer ce modèle de classification afin de le rendre plus précis en travaillant en collaboration étroite avec les équipes qualité pour obtenir plus d'images et également optimiser les paramètres de génération du modèle via la méthode Keras-Tuner équivalent du Grid Search.





Merci de votre attention