Détection des accidents- images-

-Machine Learning-

Réalisé par :

- MOHAMED EL BADAOUI
- SARA EL OMARY

Encadré par :

- Mme HAJAR MOUSSANIF

Les problèmes rencontrés :

- Building de dataset
- problème de performances des machines a cause du gros volume de la base de données
- Trouver le bon modèle qui a une bonne accuracy

1) Introduction

Ce n’est plus un secret pour personne, le Maroc est parmi les pays où il y’a le plus d ‘accidents de la circulation dans le monde, il a été classé premier au niveau arabe et le 6eme à l'échelle mondiale d’après le site happy awkward .

Chaque année plus de 4000 personnes trouvent la mort sur nos routes, un bilan humain très lourd qui cause à l’état une perte matérielle d’environ 14 milliards de dirhams, soit 2% du produit intérieur brut.

Pour faire face à cette situation, de nombreuses initiatives ont vue le jour (nouveau code de la route, campagnes de sensibilisations etc..) , mais aucune n’a pu éradiquer cette hémorragie mortelle qui fait du Maroc un pays à haut risque en termes d’accidents de la circulation.

Alors ce projet-là a été proposé par la ministère du transport et logistique afin de diminuer la fréquence des morts au Maroc bien à l’aide des vidéos.

2) **Architecture** :

Dans les réseaux de neurones, le réseau de neurones (ConvNets ou CNN) est l'une des principales catégories permettant la reconnaissance d'images, la classification des images. Détections d'objets, visages de reconnaissance, etc., sont quelques-uns des domaines dans lesquels les CNN sont largement utilisés**.**

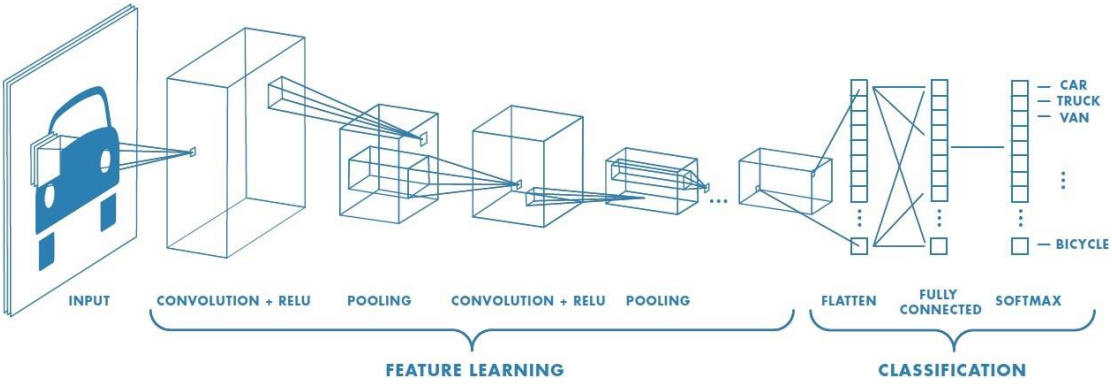
Les classifications d'images CNN prennent une image d'entrée, la traitent et la classent dans certaines catégories (par exemple, Chien, Chat, Tigre, Lion).

Puisque notre problème sert à classer les images dans notre cas on a 2 catégories image avec ACCIDENT et image SANS ACCIDENT.

Image avec accident

Image sans accident

A – Modèle CNN :



1.1 Couche de convolution

La convolution est le premier calque permettant d'extraire des entités d'une image d'entrée. La convolution préserve la relation entre les pixels en apprenant les caractéristiques de l'image à l'aide de petits carrés de données d'entrée. C'est une opération mathématique qui prend deux entrées comme une matrice d'image et un filtre ou kernal.

1.2 Padding :

Parfois, le filtre ne correspond pas parfaitement à l'image d'entrée. Nous avons deux options :

- Pad la photo avec des zéros (zéro-rembourrage) afin qu'il tienne
- Déposez la partie de l'image où le filtre ne correspondait pas. Cela s'appelle un remplissage valide qui ne conserve qu'une partie valide de l'image.

1.3 Non linéarité (ReLU)

ReLU signifie unité linéaire rectifiée pour une opération non linéaire. La sortie est ***ƒ (x) = max (0, x).***

Le but de ReLU est d'introduire la non-linéarité dans notre ConvNet. Depuis, les données du monde réel voudraient que notre ConvNet apprenne qu'il s'agirait de valeurs linéaires non négatives.

1.4 Layer Padding

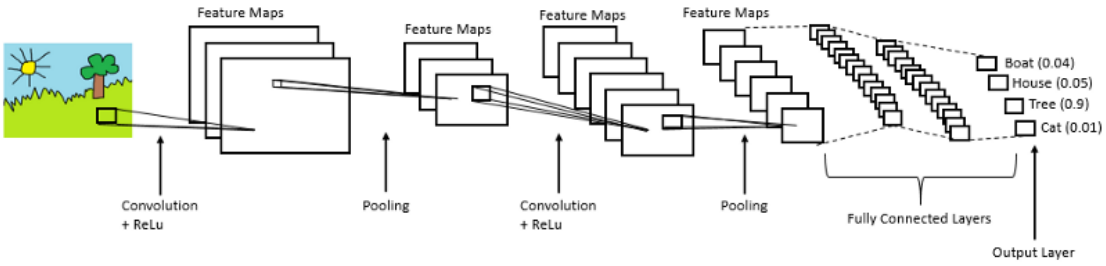
La section de regroupement des couches réduirait le nombre de paramètres lorsque les images sont trop grandes. Le regroupement spatial est également appelé sous-échantillonnage ou sous-échantillonnage, ce qui réduit la dimensionnalité de chaque carte mais conserve les informations importantes. Le pooling spatial peut être de différents types :

- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling : maximale prend l'élément le plus important de la carte d'entités rectifiée. Prendre le plus gros élément pourrait aussi prendre le pooling moyen. La somme de tous les éléments de l'appel de la carte des caractéristiques est une somme totale.

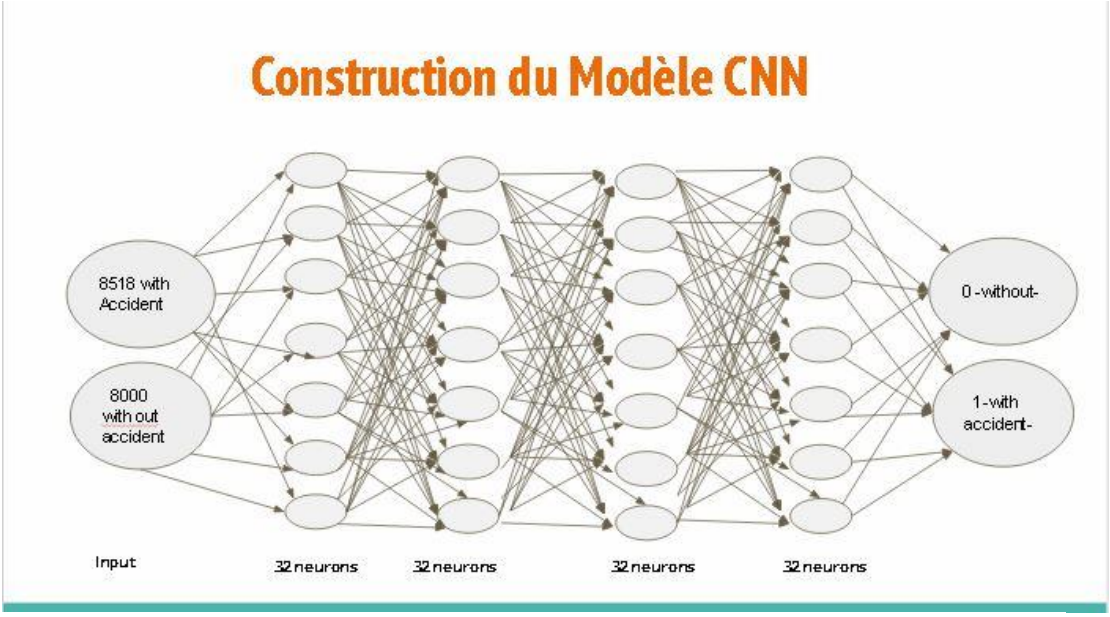
1.5 Fully Connected Layer :

La couche que nous appelons couche FC, nous avons aplati notre matrice en vecteur et l'avons insérée dans une couche entièrement connectée, telle qu'un réseau de neurones.



Architecture :

→CNN avec Keras et tensorflow



On a collecté 16 518 images, dans lesquelles on a 8518 images WITH ACCIDENT et 8000 WITHOUT ACCIDENT puis par la suite on a commencé par 1 layer vers 4 layers afin d'augmenter l'accuracy et c'était 90% pour la partie validation .

Car on a travaillé avec keras par défaut l'initialisation des neurones est 32 neurones/ layer .

3) les outils et langages :

4) Code :

```
[ ]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import matplotlib.image as mpimg
     import seaborn as sns
     %matplotlib inline

     from sklearn.model_selection import train_test_split
     from sklearn.metrics import confusion_matrix
     import itertools
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
     from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization
     from tensorflow.keras.losses import categorical_crossentropy
     from tensorflow.keras.regularizers import l2
     from tensorflow.keras.callbacks import ReduceLROnPlateau, TensorBoard, EarlyStopping, ModelCheckpoint
     from tensorflow.keras.optimizers import Adam
     from keras.preprocessing.image import ImageDataGenerator
     from keras.callbacks import ReduceLROnPlateau
     from keras.models import load_model
```

Figure 1 : Importation des bibliothèques

```
[ ]  import numpy as np # linear algebra
     from PIL import Image
     import glob

     def buildData(path, label):
       labels=[]
       images=[]
       for filename in glob.glob(path):
           im=Image.open(filename)
           images.append(np.array(im))
           labels.append(label)
       return images, labels

     x, y = buildData("/content/gdrive/My Drive/FILESS/training_set/withAccident/*", 1)
     a, b = buildData("/content/gdrive/My Drive/FILESS/training_set/withoutAccident/*", 0)
     x.extend(a)
     y.extend(b)
     x = np.expand_dims(np.array(x), -1)
     print(x.shape)
     print(np.array(y).shape)
     #get labels
     y = pd.get_dummies(np.array(y))
```

Figure 2 : Mettre les images dans un tableau et les donner un Label

```
[ ]  x = np.array(x)
     #split training and valdiation set
     random_seed = 2
     x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=random_seed)
     print(x_train.shape)
```

Figure 3 : Split training et validation test

```
[ ]  # Initialising the CNN
     classifier = Sequential()

     # Step 1 - Convolution
     classifier.add(Conv2D(32, (3, 3), input_shape=(256, 256, 1), activation = 'relu'))
     classifier.add(BatchNormalization())

     # Step 2 - Pooling
     classifier.add(MaxPooling2D(pool_size = (2, 2)))
     classifier.add(Dropout(0.2))

     # Adding a second convolutional layer
     classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
     classifier.add(BatchNormalization())
     classifier.add(MaxPooling2D(pool_size = (2, 2)))
     classifier.add(Dropout(0.2))


     # Adding a tree convolutional layer
     classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
     classifier.add(BatchNormalization())
     classifier.add(MaxPooling2D(pool_size = (2, 2)))
     classifier.add(Dropout(0.2))

     # Adding a 4 convolutional layer
     classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
     classifier.add(BatchNormalization())
     classifier.add(MaxPooling2D(pool_size = (2, 2)))
     classifier.add(Dropout(0.2))

     # Step 3 - Flattening
     classifier.add(Flatten())

     # Step 4 - Full connection
     classifier.add(Dense(units = 128, activation = 'relu'))
     classifier.add(Dense(units = 128, activation = 'relu'))
     classifier.add(Dense(units = 128, activation = 'relu'))
     classifier.add(Dense(units = 2, activation = 'sigmoid'))
```

```
# Define the optimizer
optimizer = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-7)
# Compile the model
classifier.compile(loss=categorical_crossentropy,
              optimizer = optimizer,
              metrics=['accuracy'])
# Set a learning rate annealer
early_stopper = EarlyStopping(monitor='val_acc', min_delta=0, patience=8, verbose=1, mode='auto')
checkpointer = ModelCheckpoint('/content/gdrive/My Drive/model2.h3', monitor='val_acc', verbose=1, save_best_only=True)

# Compiling the CNN
classifier.compile(optimizer = optimizer, loss = 'binary_crossentropy', metrics = ['accuracy'])

# Part 2 - Fitting the CNN to the images

from keras.preprocessing.image import ImageDataGenerator

test_datagen = ImageDataGenerator(rescale = 1./255)

classifier.fit(x_train, y_train, batch_size=50,
          validation_data = (x_test, y_test),
          epochs=400,
          verbose=1,
          callbacks=[early_stopper, checkpointer]
              )

classifier.save('save.h3')
```

Figure 4 : Création du Modèle et enregistrement des weights

```
Epoch 00031: val_acc improved from 0.89885 to 0.90036, saving model to /content/gdrive/My Drive/model2.h3
13264/13264 [==============================] - 66s 5ms/sample - loss: 0.0169 - acc: 0.9934 - val_loss: 0.5870 - val_acc: 0.9004
```

Figure 5 : Accuracy du Modele

```
[ ]  from tensorflow.keras.models import load_model
     MODELPATH = 'save.h3'
     model = load_model(MODELPATH)
     scores = model.evaluate(np.array(x_test), np.array(y_test))
     print(scores[0])
     print(scores[1])
```

Figure 6 : Load du Modèle

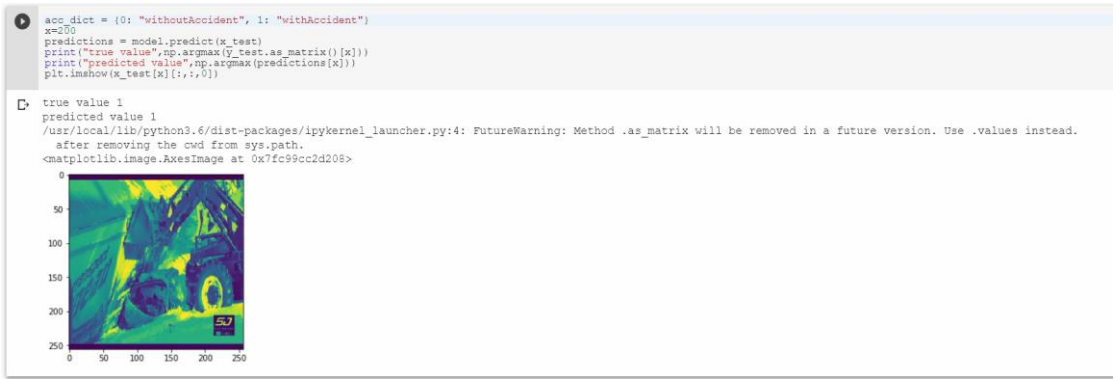


Figure 7 : Affichage De la valeur réelle et la valeur prédite

5) Simulation

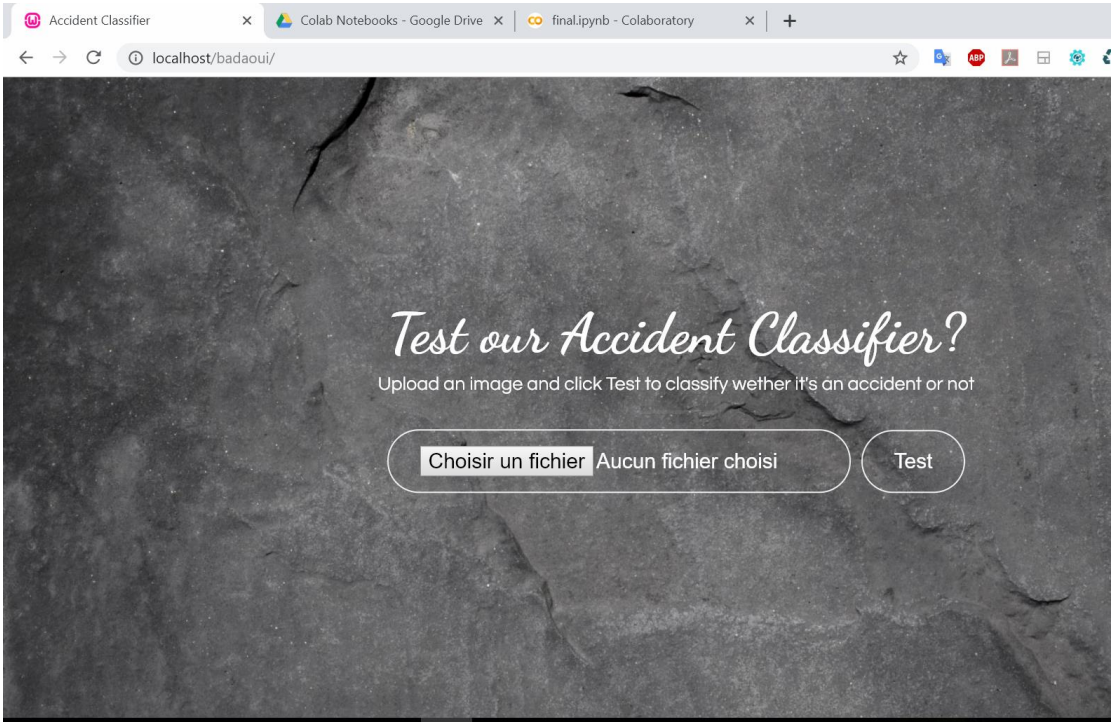


Figure 1 :Site Web pour le test

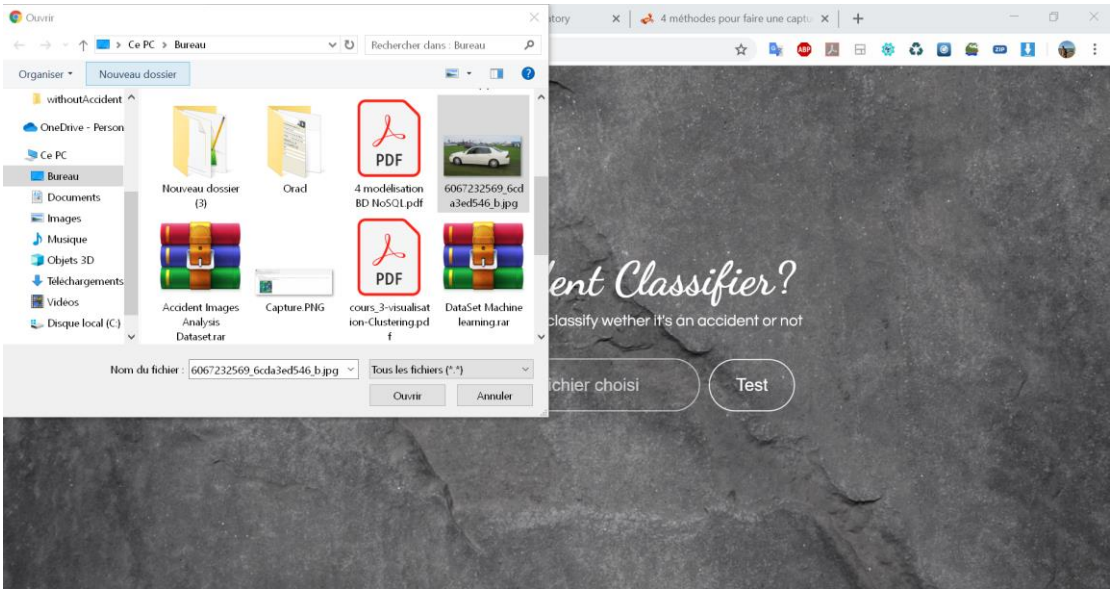


Figure 2 : Choix de l'image

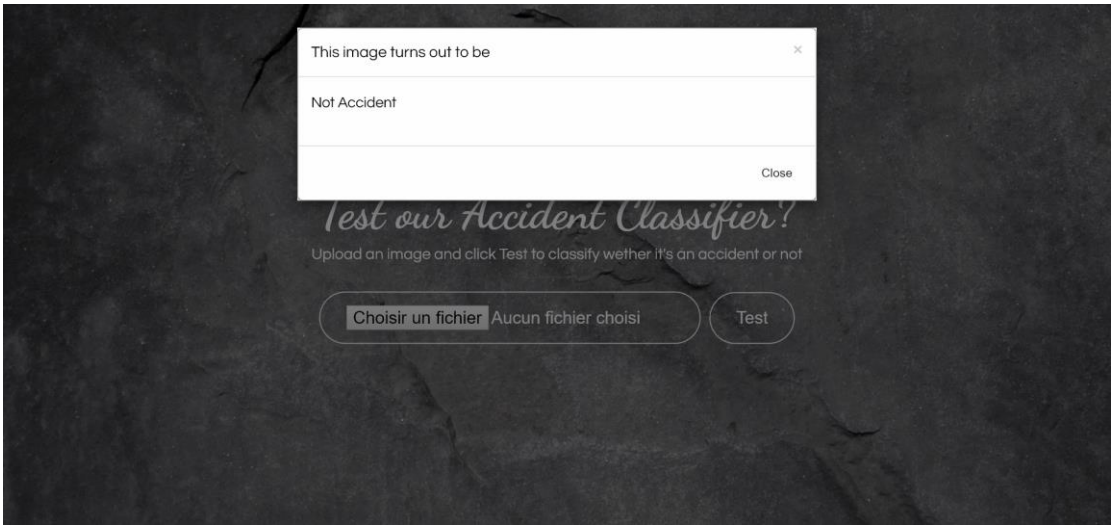


Figure 3 : Résultat de la classification

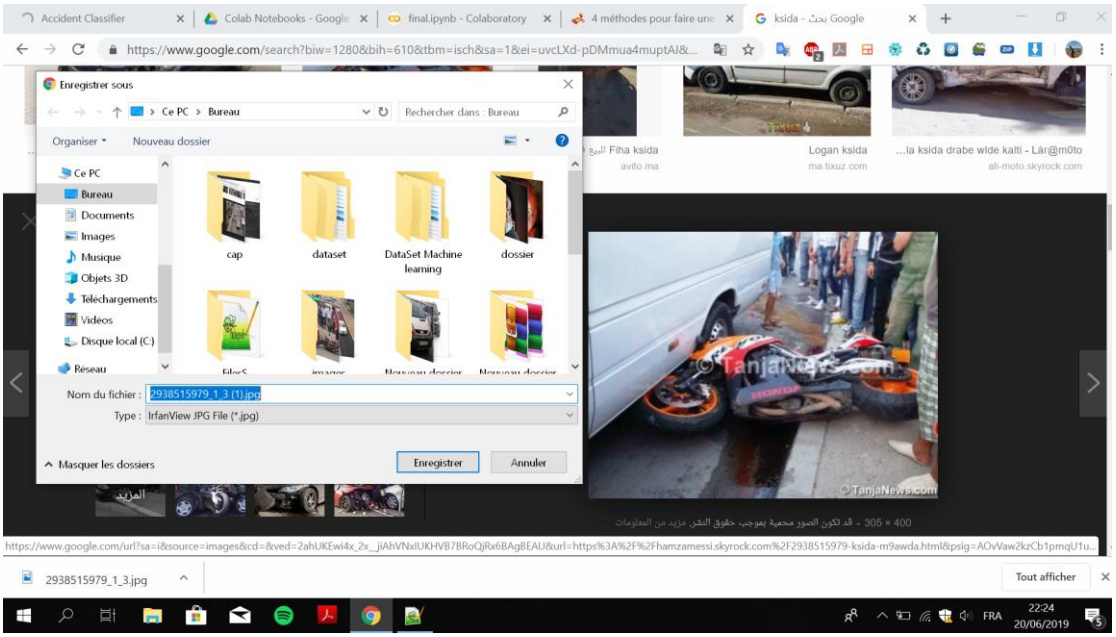


Figure 4 : Téléchargement de l'image

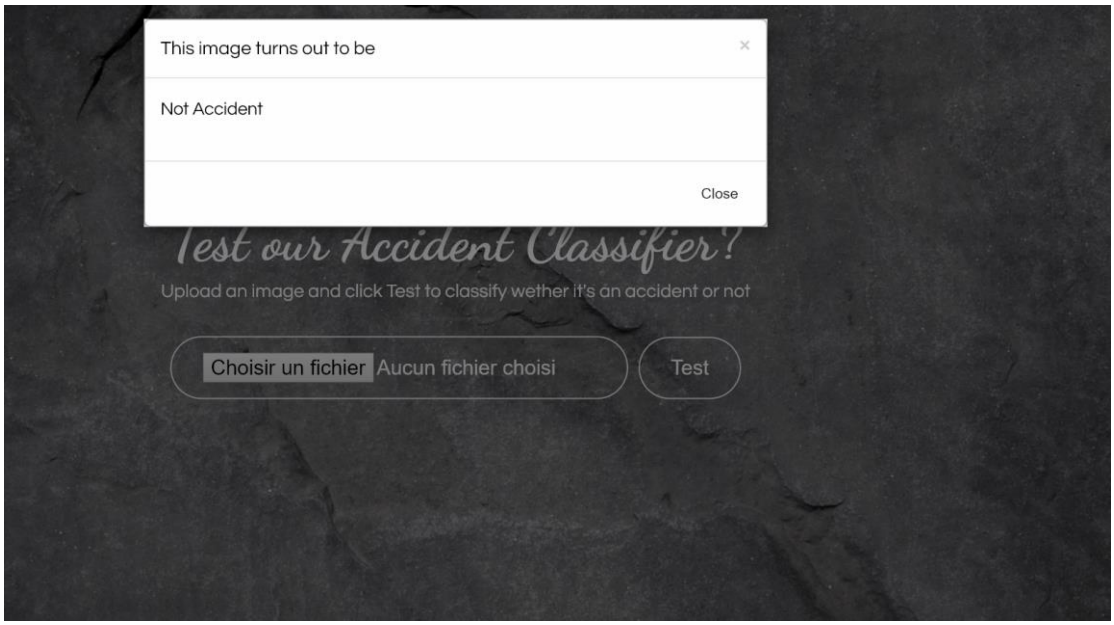


Figure 5 : Image with Accident