Rapport  
Modèle de détection de masques

Partie théorique

#### Convolutional Neural Network

L’architecture CNN (convolutional neural network) est un modèle de réseau de neurones utilisé pour la détection d’images dans la mesure où il est spécialement conçu pour traiter des images en entrée.

La phase d’apprentissage des ‘features’ est composé de quatre couches: la couche de convolution, la couche de pooling, la couche de correction ReLU et la couche fully-connected (image 1).

Ces couches ont pour objectif de filtrer l’image pour ne garder que les informations discriminantes.



Image 1. Schéma d’un CNN

#### 2. Architecture VGG16

VGG est un réseau de neurones convolutionnels qui utilise une architecture précise de couches de convolutions et de poolings (image 2).

Il prend en entrée des images de taille 224 x 224. Pour toutes les couches de convolution, le noyau de convolution est de taille 3x3 alors que pour les couches de pooling, le noyau est de 2x2.

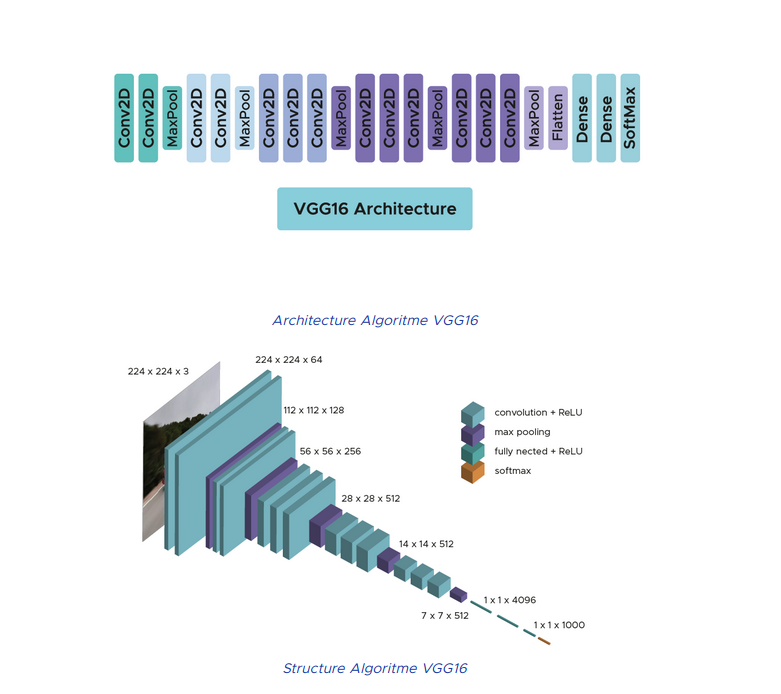


Image 2. Schéma du VGG16

#### 3. Transfer Learning

Le transfer learning ou l’apprentissage par transfert consiste à réutiliser des connaissances déjà acquises dans d’autres configurations pour la résolution d’un nouveau problème. L’objectif est donc d’utiliser des modèles pré-entrainés pour développer rapidement des modèles performants sur de nouvelles problématiques plus complexes. Il permet de gagner du temps de calcul et des ressources.

Il est principalement utilisé en reconnaissance d’image avec le CNN (Convolutional Neural Network) et le NLP (Natural Language Processing)

VGG16 ayant été entraîné sur Imagenet, une très grande base de données de 14 millions d’images labellisées sur plus de 1000 classes, il est possible de le remployer pour la détection de nouvelles images. L’intérêt est de réutiliser les connaissances acquises sur Imagenet.

#### 4. Data augmentation

Pour qu’un modèle de CNN puisse classifier les images de manière robuste, il doit pouvoir s’entraîner sur un jeu de donnée important. La data augmentation permet d’augmenter un jeu de données en y appliquant de légères transformations, telles que des translations, rotations ou retournements. L’objectif est à la fois d’accroître les données et de montrer au modèle le maximum d’images dans des configurations différentes.

#### 5. Sauvegarder le meilleur modèle

model.save(cwd + '/model\_mask.h5', save\_format = 'h5')

## Partie pratique

Le jeu de données est composé d’images de visages avec et sans masques.

L’objectif du brief est d’utiliser l’architecture VGG16 pour détecter les masques sur les visages, en utilisant une classification binaire.

Le notebook est divisé en trois parties.

* Chargement de la base de données, la préparation et l’analyse
* Architecture CNN sur tensorflow
* Développement d’une application

#### Partie 1. Chargement de la base de données, la préparation et l’analyse

Dans un premier temps, nous chargeons le chemin des images dans un dictionnaire grâce à la méthode .glob() de Pathlib. Puis nous créons une boucle pour lire et redimensionner chaque image, que nous stockons dans une liste X. Nous associons le label de l’image dans une liste y (image 3).



Image 3. Code

Nous divisons ensuite le jeu de données en jeu d’entraînement et de test. A partir du jeu d’entraînement, on le redivise en jeu d’entraînement et de validation.

On normalise les images de nos jeux de données et on transforme nos labels en matrice grâce à la fonction to\_categorial().

Enfin, on définit une variable pour augmenter les données sur notre jeu d’entraînement.

#### Partie 2. Architecture CNN

Visualisation de l’architecture du modèle VGG16

On configure le modèle VGG16 en figeant les couches de classification (le but ici est de conserver les poids de l’étape des features learning) (image 4). On change les couches de classification en mettant deux outputs avec une fonction d’activation softmax (image 5).

On compile le modèle avec une erreur de Categorical cross entropy (image 5Image 3. Code Image 3. Code ).

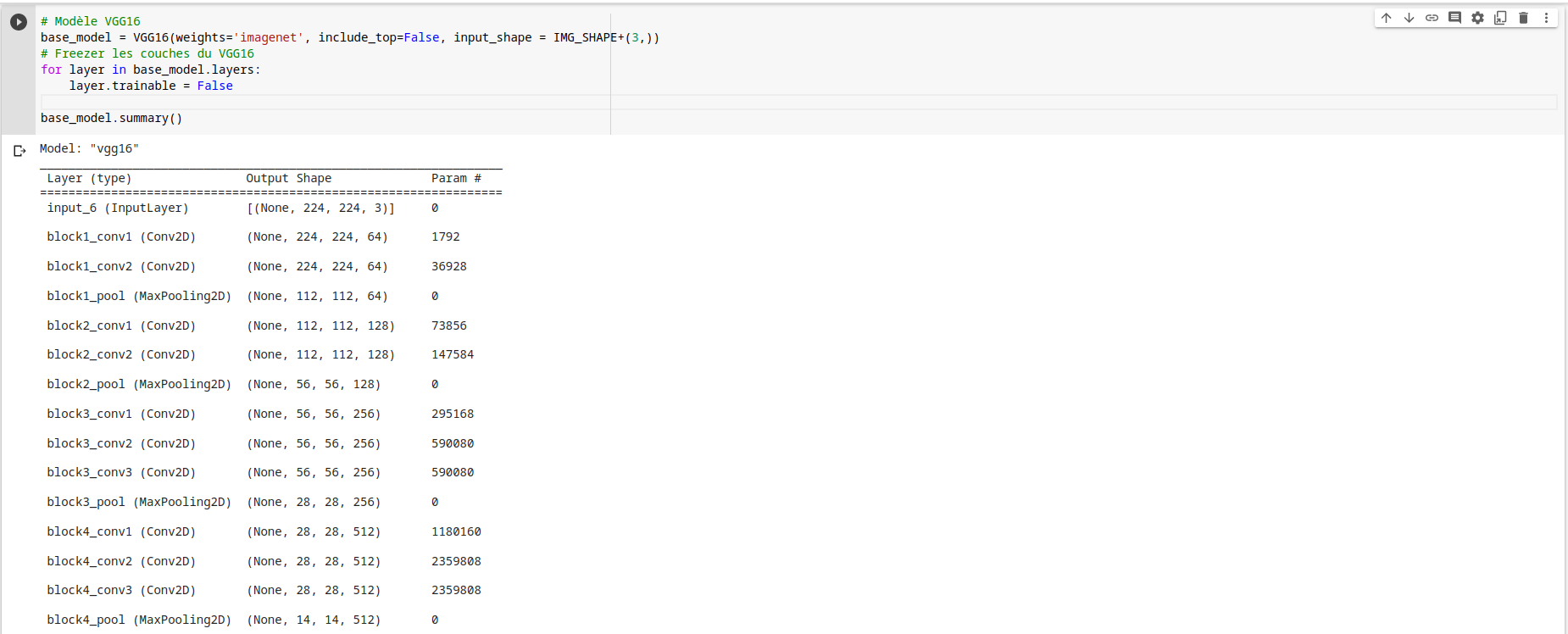


Image 4. Code



Image 5. Code

On configure le modèle pour ne garder que le meilleur modèle au fur et à mesure de chaque itération (epoch).

Lorsque l’on fit le modèle, on augmente les données d’entraînement et on utilise le jeu de validation pour chaque itération. On le stocke dans une variable ‘history’ (image 6).

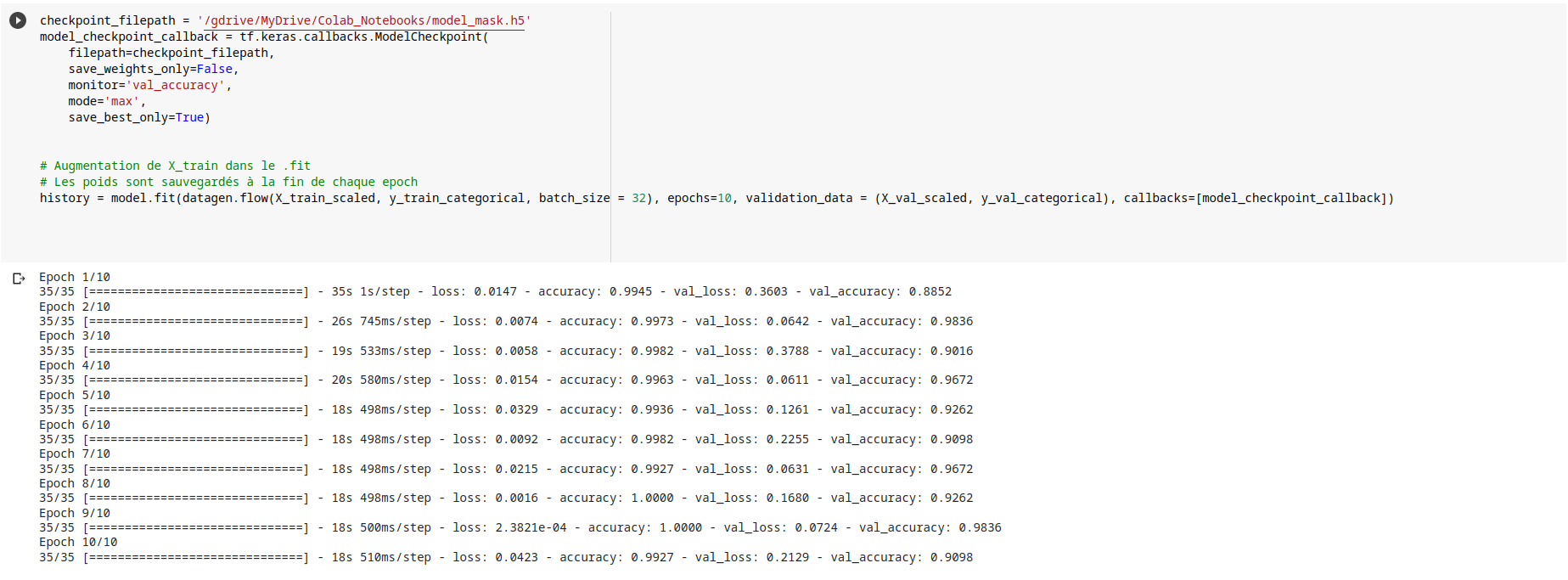


Image 6. Code

On affiche la courbe d’erreur (image 7) et de précision (image 8) pour les données d’entraînement et de validation.

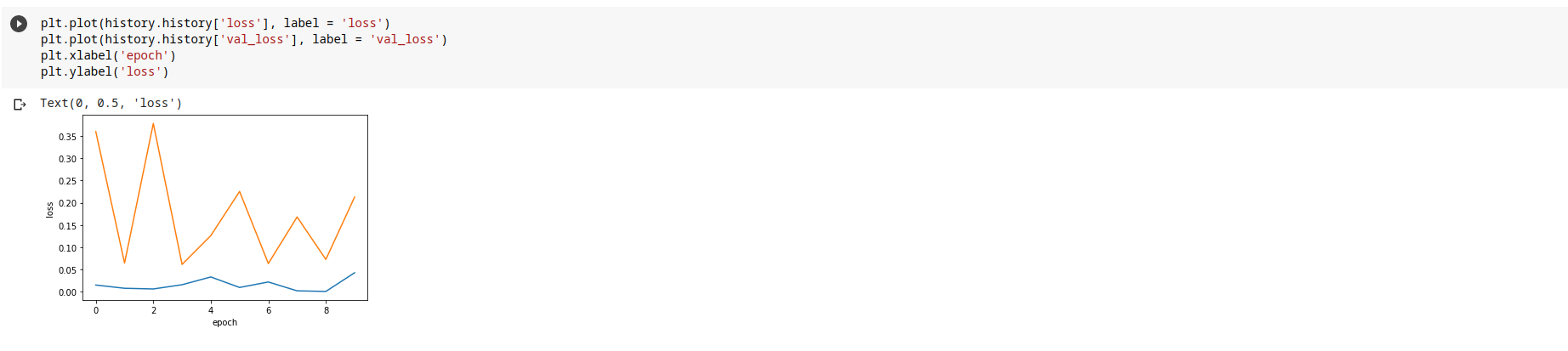


Image 7. Code



Image 8. Code

On prédit les données sur le jeu de test (image 9).

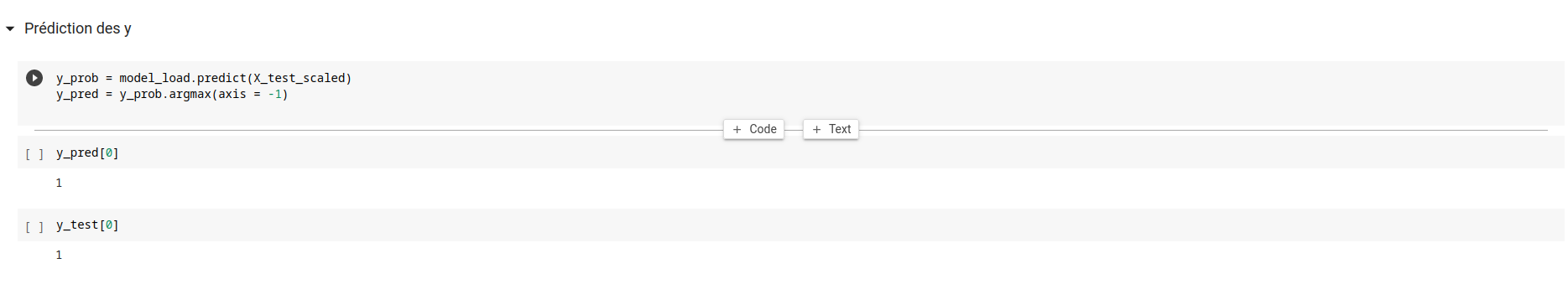


Image 9. Code

On réalise une matrice de confusion (image 10).

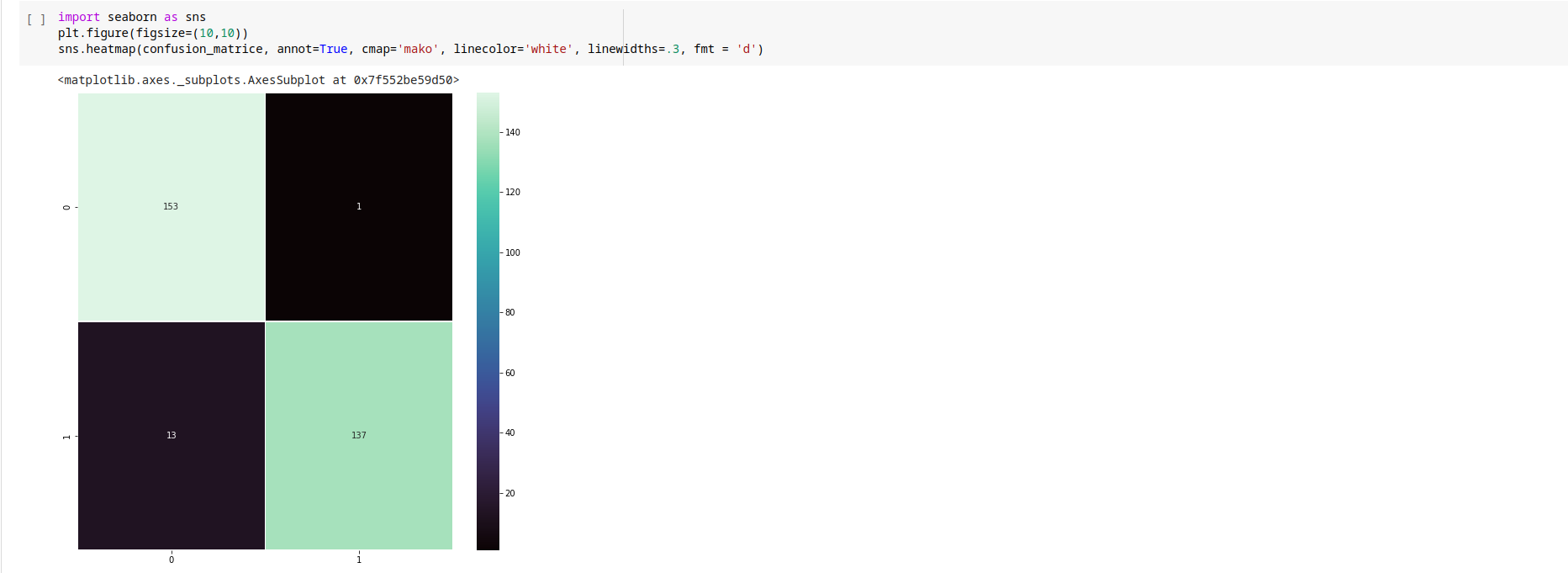


Image 10. Code

Enfin, on teste notre modèle sur de nouvelles photos.

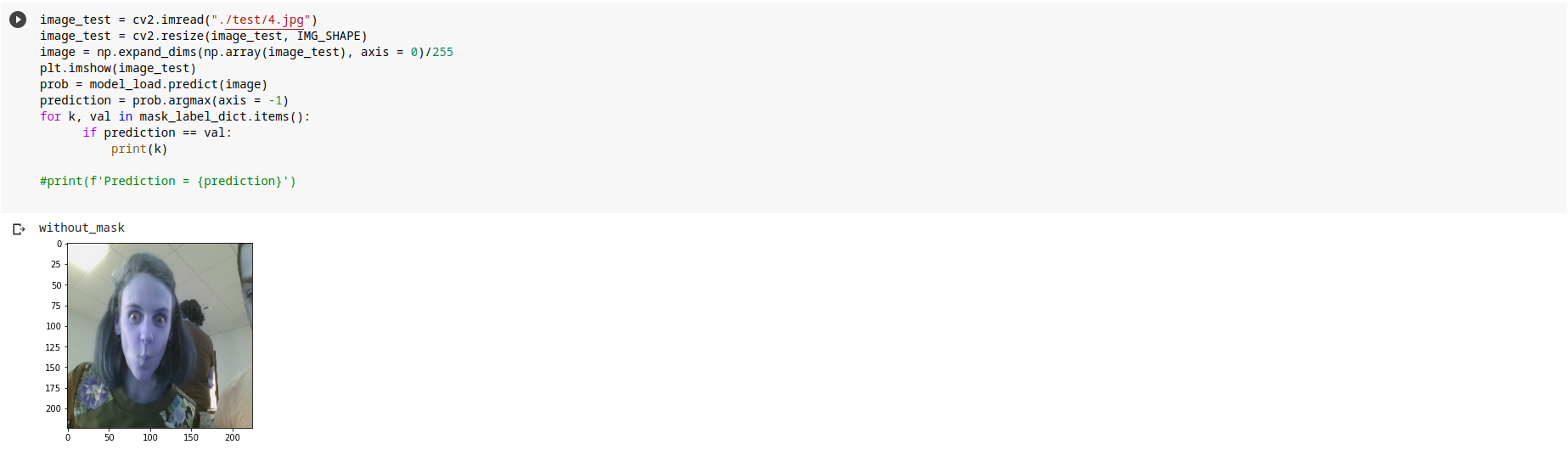


Image 11. Code

#### Partie 3. Développement d’une application

