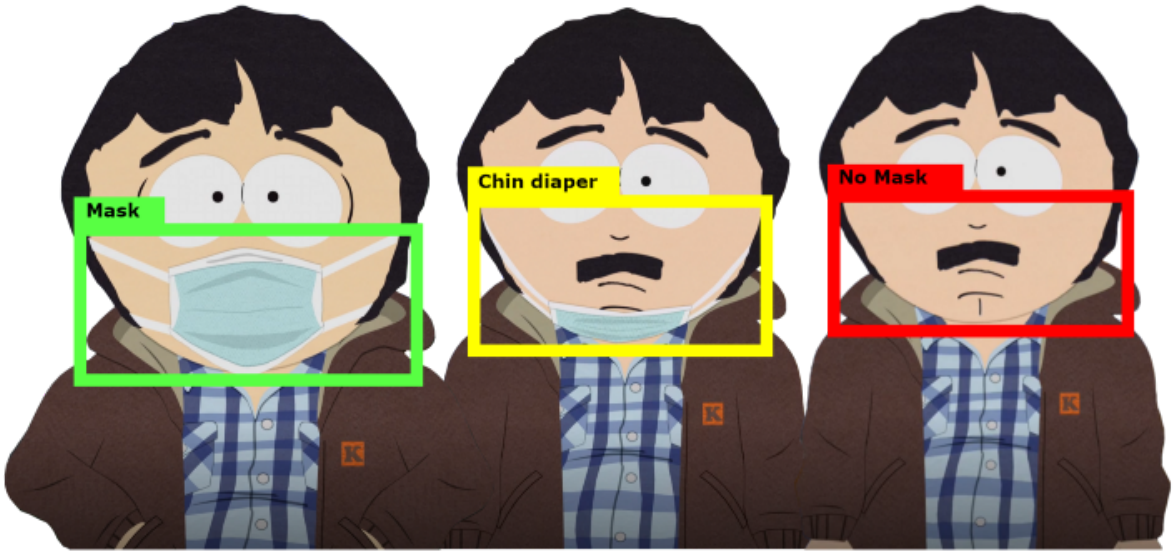
MARONE Cheikh Ahmet Tidiane Chérif
KHATER Elio

ILC - TP2

PROJET - Détecteur de Masque



Source :
https://ichi.pro/assets/images/max/724/1*QkD4Wk-B0z3VpcmviGawSw.png

TABLE DES MATIÈRES

Introduction **3**

Conception du Dataset **3**

Choix de l'algorithme ou modèle **3**

Modèle Initial **3**

Modèle Final **6**

Mise en œuvre et évaluation **8**

Conclusion **9**

Introduction

L'objectif de ce projet est de permettre aux étudiants de voir l'ensemble de la chaîne permettant de concevoir une application à base de Machine Learning (incluant les méthodes de Deep-Learning présentées lors des cours).

Conception du Dataset

Tout d'abord il fallait concevoir un dataset afin de pouvoir entraîner notre modèle. Pour cela chaque groupe d'étudiants était chargé de chercher des images respectant une certaine norme, de les charger sur le dataset si ces derniers n'étaient pas déjà présents. Pour cette vérification, on a utilisé un not développer par l'un de nos camarades, dont le but est d'éviter les doublons d'image. Une l'image validée, on passe ainsi à l'annotation de ce dernier en Pascal voc.
une fois l'ensemble des images et des annotations validées, le dataset final est composé de `2655` images annotées à l'aide de boites englobantes et se focalise sur 3 classes d'objets :

1. Label "without_mask" : Visages sans masque
2. Label "mask_weared_incorrect " : Visages avec un masque mal porté
3. Label "with_mask" : Visages avec un masque (bien porté)

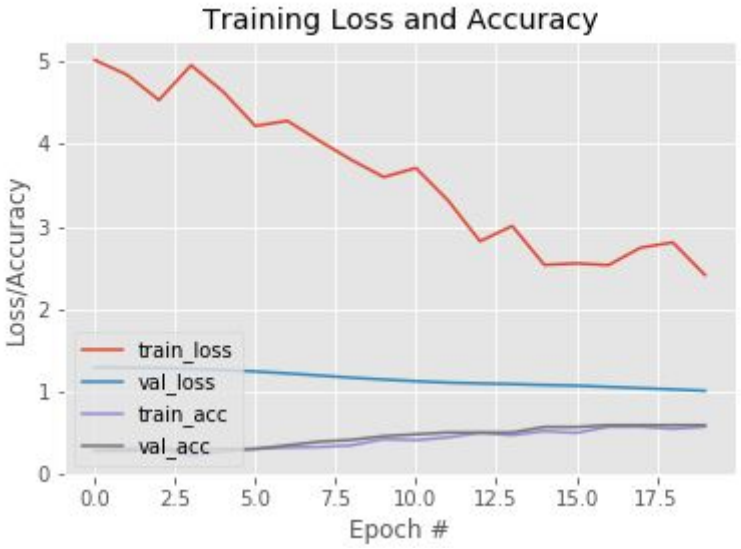
Choix de l'algorithme ou modèle

Modèle Initial

Au départ, nous avions choisi le modèle https://www.kaggle.com/mirzamujtaba/face-mask-detection qui utilise MobilenetV2 enlevant la couche de sortie pour l'adapter à la détection de masque.
Nous l'avions testé avec « nos images » soit 137 (avant la finalisation du dataset), les résultats étaient prometteurs. En effet, le flux vidéo détectait les 3 labels, le val_loss qui descend et le val_accuracy qui monte, ce qui atteste que le modèle apprend. Au final, on a testé avec deux optimiseur RAdam puis avec Adam, Adam donnant de meilleurs résultats .

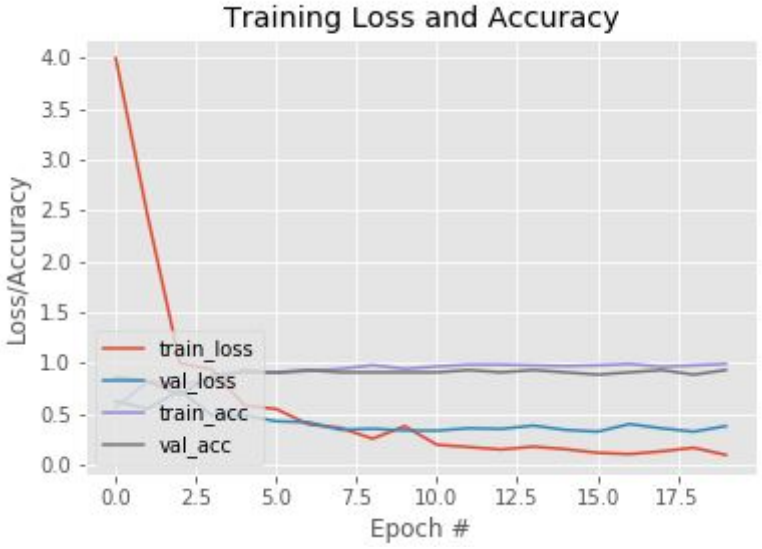
Lien GitHub le fichier d'origine avec MobilenetV2 :
https://github.com/bruaba/mask_detection/blob/main/OldTrainVersion/main.py

AVec RAdam

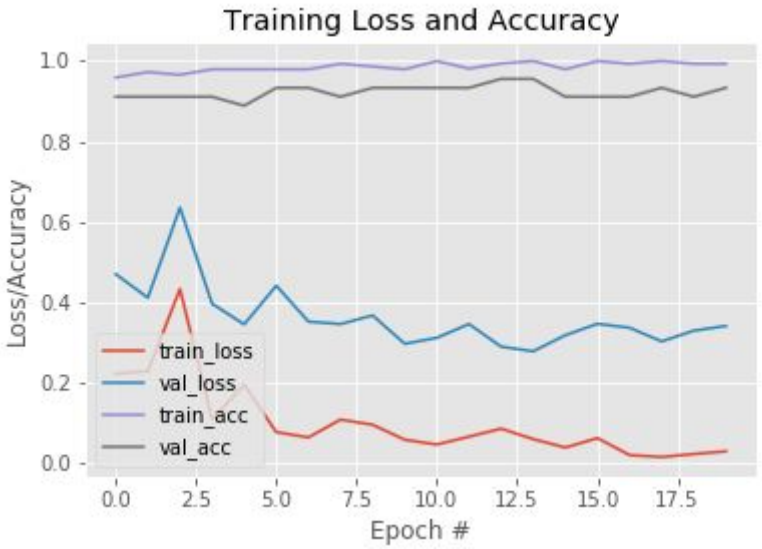


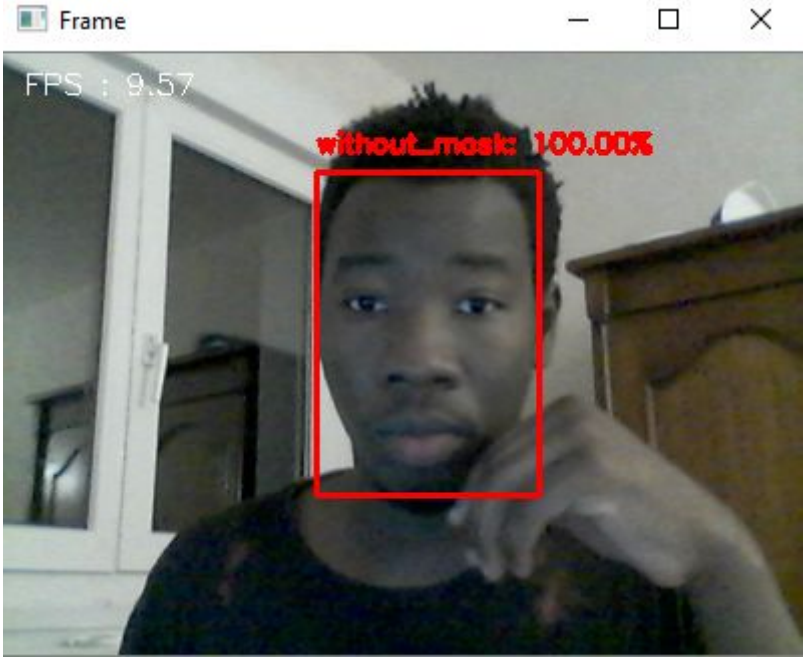
1er entrainement

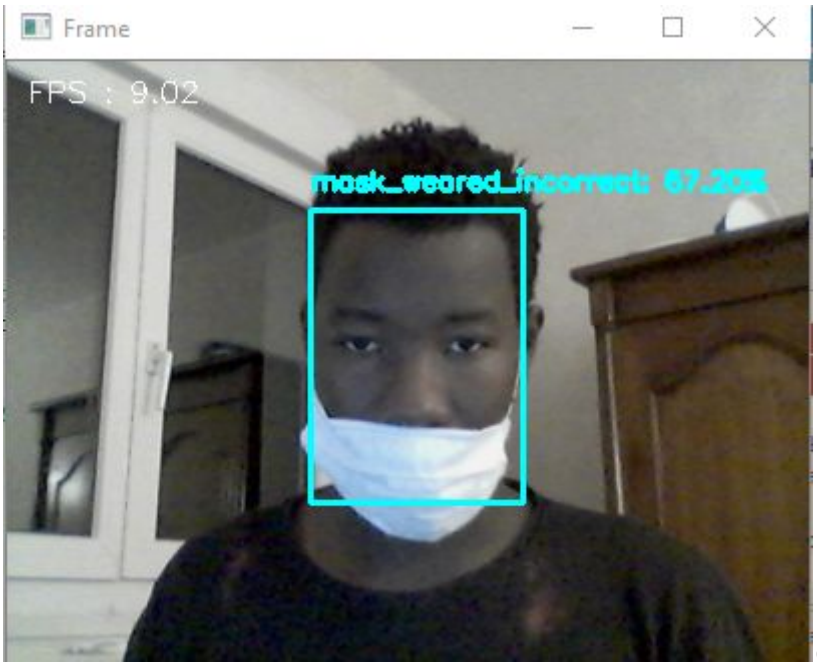
avec Adam



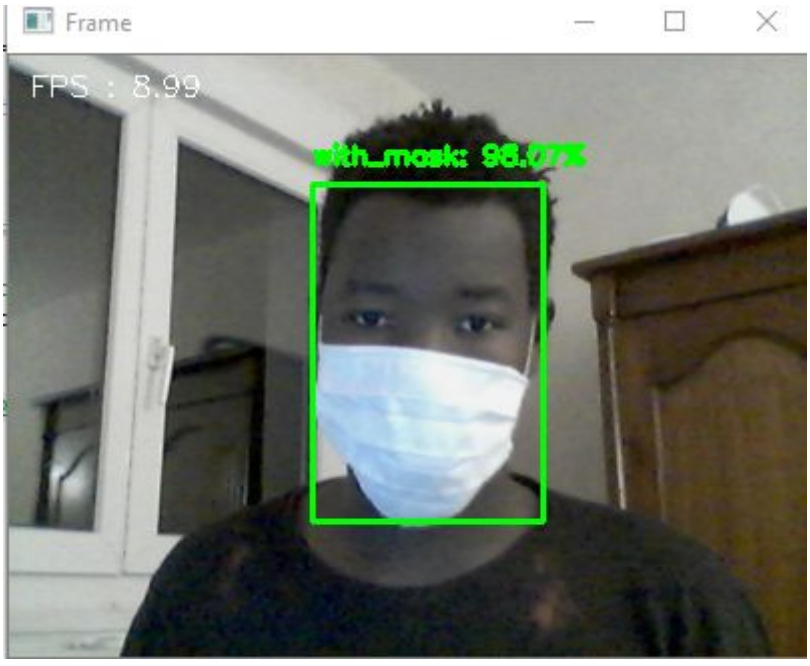
En faisant un deuxième entraînement, on constate que le modèle est surentraîné.
2e entraînement (sur entraîner)


Training Loss and Accuracy
Loss/Accuracy
Epoch #
train_loss
val_loss
train_acc
val_acc
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Frame
FPS : 9.57
without_mask: 100.00%



q



Modèle Final

Une fois le dataset finalisé, on l'a intégré à nos entraînements. Et là, à notre grande surprise, le val_loss se met à augmenter comme le val_accuracy .
Ainsi nous avons entrepris des recherches pour connaître l'origine de ce phénomène. On a trouver l'explication suivante :

https://stackoverflow.com/questions/51704808/what-is-the-difference-between-loss-accuracy-validation-loss-validation-accur
et
https://datascience.stackexchange.com/questions/43191/validation-loss-is-not-decreasing

Ainsi, on a joué sur différent facteur (modifications du learning rate, du batch size, le nombre époque, ajout d'une couche drop out , augmentation de cette dernière, diminution ou suppression de couche dense, …) notre histoire des commits peut en attester, au final aucune de ces modifications n'a pu aboutir à endiguer ce phénomène. Cela peut être dû aussi au fait qu'on a modifier plusieurs variables en même temps.

Ainsi, nous avons décidé de faire volte face et d'essayer avec d'autres modèles.
Tout d'abord avec inceptionV3 : elle prend plus de temps que mobilenetV2 mais il nous a pas donné de résultats satisfaisants.
On a aussi essayé l'algorithme suivant :

```
model = Sequential([
                    Conv2D(filters=32, kernel_size=(3,3),activation='relu',padding='same',input_shape=(70,70,3)),
                    MaxPool2D(pool_size=(2,2), strides=2),
                    Conv2D(filters=32, kernel_size=(3,3), activation='relu', padding= 'same'),
                    MaxPool2D(pool_size=(2,2), strides =2),
                    Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding= 'same'),
                    MaxPool2D(pool_size=(2,2), strides =2),
                    Flatten(),
                    Dense(units=64, activation= 'relu'),
                    #means the output is 0,1 (the labels) and the P(c=0) +P(c=1) = 1
                    Dense(units=1, activation='sigmoid'),

])
```

https://github.com/The-Assembly/Build-A-Face-Mask-Detector-With-TensorFlow/blob/main/FaceMask%20detection/FaceMaskDectection2.py
Mais des résultats identiques.

```
[ ]
H = model.fit(trainX, trainY,
                    epochs=20,
                    callbacks=[checkpoint],
                    validation_data=(testX, testY))
```

```
Epoch 1/20
163/163 [==============================] - 511s 3s/step - loss: 2.1876 - acc: 0.6170 - val_loss: 0.8082 - val_acc: 0.6938
Epoch 2/20
163/163 [==============================] - 508s 3s/step - loss: 0.7976 - acc: 0.6990 - val_loss: 0.7975 - val_acc: 0.6938
Epoch 3/20
163/163 [==============================] - 506s 3s/step - loss: 0.7455 - acc: 0.7058 - val_loss: 0.8014 - val_acc: 0.6938
Epoch 4/20
163/163 [==============================] - 505s 3s/step - loss: 0.6988 - acc: 0.7042 - val_loss: 0.8265 - val_acc: 0.6920
Epoch 5/20
163/163 [==============================] - 505s 3s/step - loss: 0.6661 - acc: 0.7108 - val_loss: 0.8489 - val_acc: 0.6920
Epoch 6/20
163/163 [==============================] - 504s 3s/step - loss: 0.6046 - acc: 0.7279 - val_loss: 0.8842 - val_acc: 0.6972
Epoch 7/20
163/163 [==============================] - 505s 3s/step - loss: 0.5908 - acc: 0.7263 - val_loss: 0.9668 - val_acc: 0.6938
Epoch 8/20
163/163 [==============================] - 506s 3s/step - loss: 0.5892 - acc: 0.7274 - val_loss: 0.8883 - val_acc: 0.6903
Epoch 9/20
163/163 [==============================] - 506s 3s/step - loss: 0.5444 - acc: 0.7444 - val_loss: 1.0244 - val_acc: 0.6972
Epoch 10/20
153/163 [===========================>..] - ETA: 28s - loss: 0.5164 - acc: 0.7431
```

avec lr = 0.001.

Au final c’est avec Resnet50V2 qu’on obtient un modèle qui apprend.

```
[INFO] training head...
Epoch 1/30
35/35 [==============================] - 47s 1s/step - loss: 0.5385 - accuracy: 0.6673 - val_loss: 0.4994 - val_accuracy: 0.6922
Epoch 2/30
35/35 [==============================] - 47s 1s/step - loss: 0.5201 - accuracy: 0.6735 - val_loss: 0.4936 - val_accuracy: 0.6922
Epoch 3/30
35/35 [==============================] - 47s 1s/step - loss: 0.5114 - accuracy: 0.6776 - val_loss: 0.4912 - val_accuracy: 0.6922
Epoch 4/30
35/35 [==============================] - 47s 1s/step - loss: 0.5049 - accuracy: 0.6814 - val_loss: 0.4910 - val_accuracy: 0.6931
Epoch 5/30
35/35 [==============================] - 47s 1s/step - loss: 0.4956 - accuracy: 0.6836 - val_loss: 0.4896 - val_accuracy: 0.6914
Epoch 6/30
35/35 [==============================] - 47s 1s/step - loss: 0.4978 - accuracy: 0.6836 - val_loss: 0.4890 - val_accuracy: 0.6922
Epoch 7/30
35/35 [==============================] - 47s 1s/step - loss: 0.4972 - accuracy: 0.6834 - val_loss: 0.4891 - val_accuracy: 0.6931
Epoch 8/30
35/35 [==============================] - 47s 1s/step - loss: 0.4923 - accuracy: 0.6906 - val_loss: 0.4886 - val_accuracy: 0.6931
Epoch 9/30
35/35 [==============================] - 47s 1s/step - loss: 0.4902 - accuracy: 0.6892 - val_loss: 0.4888 - val_accuracy: 0.6931
Epoch 10/30
35/35 [==============================] - 47s 1s/step - loss: 0.4891 - accuracy: 0.6892 - val_loss: 0.4874 - val_accuracy: 0.6931
Epoch 11/30
```

Lien GitHub le fichier resnet:
https://github.com/bruaba/mask_detection/blob/main/OldTrainVersion/Resnet_Projet_Mask.ipynb

Au même mon binôme à continuer le l’entrainement avec le MobileNetV2 et il a eu de meilleur résultat :
https://github.com/bruaba/mask_detection/blob/main/facemaskrecognitionv2.ipynb


```
Epoch 1/20
84/84 [==============================] - 141s 2s/step - loss: 0.8070 - accuracy: 0.8644 - val_loss: 0.4866 - val_accuracy: 0.8065
Epoch 2/20
84/84 [==============================] - 137s 2s/step - loss: 0.7490 - accuracy: 0.8328 - val_loss: 0.4323 - val_accuracy: 0.8257
Epoch 3/20
84/84 [==============================] - 137s 2s/step - loss: 0.7375 - accuracy: 0.8539 - val_loss: 0.4875 - val_accuracy: 0.8124
Epoch 4/20
84/84 [==============================] - 137s 2s/step - loss: 0.8237 - accuracy: 0.8504 - val_loss: 0.5159 - val_accuracy: 0.7976
Epoch 5/20
84/84 [==============================] - 138s 2s/step - loss: 0.7360 - accuracy: 0.8505 - val_loss: 0.3770 - val_accuracy: 0.8597
Epoch 6/20
84/84 [==============================] - 137s 2s/step - loss: 0.7363 - accuracy: 0.8504 - val_loss: 0.4850 - val_accuracy: 0.8050
Epoch 7/20
84/84 [==============================] - 137s 2s/step - loss: 0.7432 - accuracy: 0.8522 - val_loss: 0.4345 - val_accuracy: 0.8287
Epoch 8/20
84/84 [==============================] - 137s 2s/step - loss: 0.7439 - accuracy: 0.8613 - val_loss: 0.4200 - val_accuracy: 0.8419
Epoch 9/20
84/84 [==============================] - 137s 2s/step - loss: 0.7556 - accuracy: 0.8585 - val_loss: 0.4187 - val_accuracy: 0.8331
Epoch 10/20
84/84 [==============================] - 137s 2s/step - loss: 0.6994 - accuracy: 0.8660 - val_loss: 0.3568 - val_accuracy: 0.8685
Epoch 11/20
84/84 [==============================] - 136s 2s/step - loss: 0.6686 - accuracy: 0.8740 - val_loss: 0.3806 - val_accuracy: 0.8552
Epoch 12/20
84/84 [==============================] - 136s 2s/step - loss: 0.6331 - accuracy: 0.8952 - val_loss: 0.4434 - val_accuracy: 0.8272
Epoch 13/20
84/84 [==============================] - 135s 2s/step - loss: 0.5782 - accuracy: 0.8742 - val_loss: 0.4090 - val_accuracy: 0.8464
Epoch 14/20
84/84 [==============================] - 136s 2s/step - loss: 0.5702 - accuracy: 0.8849 - val_loss: 0.3750 - val_accuracy: 0.8612
Epoch 15/20
84/84 [==============================] - 134s 2s/step - loss: 0.5460 - accuracy: 0.8992 - val_loss: 0.4519 - val_accuracy: 0.8331
Epoch 16/20
84/84 [==============================] - 135s 2s/step - loss: 0.5924 - accuracy: 0.8941 - val_loss: 0.4132 - val_accuracy: 0.8464
Epoch 17/20
84/84 [==============================] - 136s 2s/step - loss: 0.5662 - accuracy: 0.8935 - val_loss: 0.3647 - val_accuracy: 0.8700
Epoch 18/20
84/84 [==============================] - 137s 2s/step - loss: 0.5345 - accuracy: 0.8990 - val_loss: 0.4542 - val_accuracy: 0.8287
Epoch 19/20
84/84 [==============================] - 138s 2s/step - loss: 0.5397 - accuracy: 0.8782 - val_loss: 0.3596 - val_accuracy: 0.8730
Epoch 20/20
84/84 [==============================] - 137s 2s/step - loss: 0.6267 - accuracy: 0.8943 - val_loss: 0.3565 - val_accuracy: 0.8774
```

Mise en œuvre et évaluation

Comme vu précédemment, on fait des inférences et on affiche les résultats en temps réel avec une couleur par classe, les incrustations du labels et le FPS calculé.

Pour l’évaluation de la solution du dataset on utilise le bout de code suivant:

```
# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

print(predIdxs)

# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs))

"""
from tensorflow.python.keras.metrics import Metric
k = 4

#avg_prec = Metric.average_precision_at_k(labels, predIdxs, k)

#print(avg_prec)

"""
import seaborn as sns

eval = model.evaluate(testX, testY)

print(eval)

test_Y = np.argmax(testY, axis=1)
from sklearn.metrics import confusion_matrix

sns.heatmap(confusion_matrix(test_Y, predIdxs), annot=True, fmt='g', cmap=plt.cm.Blues)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()
```

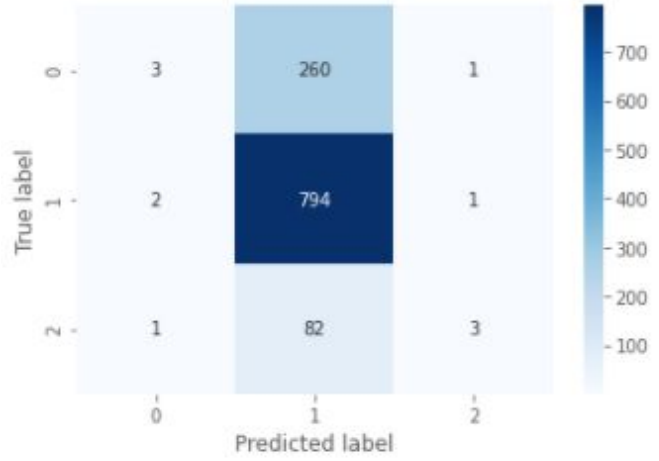
Exemple pour le ResNet50:
Lien GitHub le fichier resnet:
https://github.com/bruaba/mask_detection/blob/main/Resnet_Projet_Mask.ipynb

```
              precision    recall  f1-score   support

           0       0.50      0.01      0.02       264
           1       0.70      1.00      0.82       797
           2       0.60      0.03      0.07        86

    accuracy                           0.70      1147
   macro avg       0.60      0.35      0.30      1147
weighted avg       0.65      0.70      0.58      1147
```

```
36/36 [==============================] - 3s 85ms/step - loss: 0.4891 - accuracy: 0.6975
[0.4890788495540619, 0.6974716782569885]
```



Exemple pour le MobileNetV2 :
https://github.com/bruaba/mask_detection/blob/main/facemaskrecognitionv2.ipynb

```
[INFO] evaluating network...
              precision    recall  f1-score   support

           0       0.82      0.89      0.85       158
           1       0.95      0.90      0.92       472
           2       0.49      0.62      0.55        47

    accuracy                           0.88       677
   macro avg       0.75      0.80      0.77       677
weighted avg       0.89      0.88      0.88       677
```

Conclusion

En somme, ce projet a été très instructif sur différents facteurs et notamment les 3 principaux objectifs: conception d'une base d'images annotées destinée à l'entraînement des modèles/algorithmes de Machine Learning, sélection et entraînement d'un algorithme/modèle adapté à l'application choisie, mise en œuvre temps réel de cet algorithme pour présentation des résultats obtenus. Ainsi, il nous a permis de connaître la difficulté de créer un dataset, le travail colossal que cela représente, l'impact que le dataset peut avoir sur la solution. En effet avec nos 137 images , on avait des résultats très satisfaisants alors qu'en augmentant le nombre d'images les paramètres associés étaient catastrophiques . Donc il a fallu changer d'algorithmes faute de changement des résultats et ainsi devoir en chercher de nouveaux , les comparer. Grâce à ce projet aussi, on a pu tester différents optimiseurs, différentes fonctions de loss, étudié chaque recoin de l'algorithme pour savoir ce qu'il fait et son utilité.
Les résultats du modèle final sont pas si mal (val loss < 0.4 et val accuracy > 0.8).