

I. Introduction :

pour réaliser le projet nous avons suivi les étapes suivant : en début, Transformer la base de données vers des images. En suit, transforme les matrices des images vers une dataSet csv pour faciliter la réutilisation des matrices. En suit, l'application du module. Après, la comparaison du module avec le module de Teachable Machine . Après, la réalisation d'un d'une application mobile kivy qui utilise le module lite. Et enfin le déploiement du module.

II. Index :

Table des matières

I.	Introduction :	1
II.	Index :	1
III.	L'application du module :	2
1.	Préparation de données sous format d'image:	2
a)	Visualisation des données :	2
b)	Equilibrage de données :	3
c)	Sauvegarder les plots des images :	3
d)	Lire la matrice d'image :	4
e)	Sauvegarder les matrices des images :	4
f)	Lire les fichiers csv :	4
g)	Equilibrage de données :	4
h)	Transformer le csv vers matrices :	5
i)	Redimensionner l'image :	5
2.	Application de module :	5
3.	Transformer vers un module Lite :	6
4.	Prédictions d'une classe :	6
5.	Comparaison de module avec le module du Teachable Machine :	9
IV.	Règles d'association :	11
1.	Data Engineering :	11
2.	Règle d'association :	12
V.	Réaliser d'une application mobile kivy qui utilise le module lite :	13
1.	Using selection :	14
2.	Using the webcam :	15

3. Déploiement du module :	17
VI. Réaliser d'une application mobile avec Android Studio qui utilise le module lite :	17
1. Formulaire des règles d'association :	18
2. Résultat :	18
a) le cas du nom maladie :	18
b) le cas du nom maladie :	19

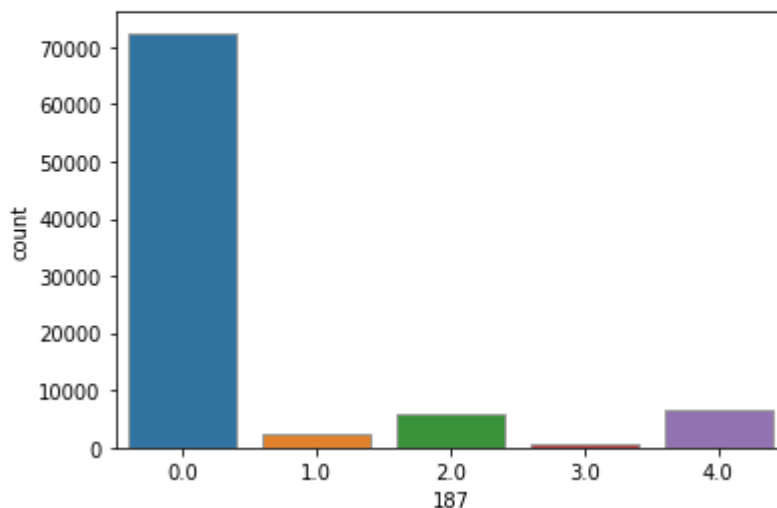
III. L'application du module :

1. Préparation de données sous format d'image:

a) Visualisation des données :

```
sb.countplot(x=187,edgecolor=".6",
             data=df)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f36ee93fb50>

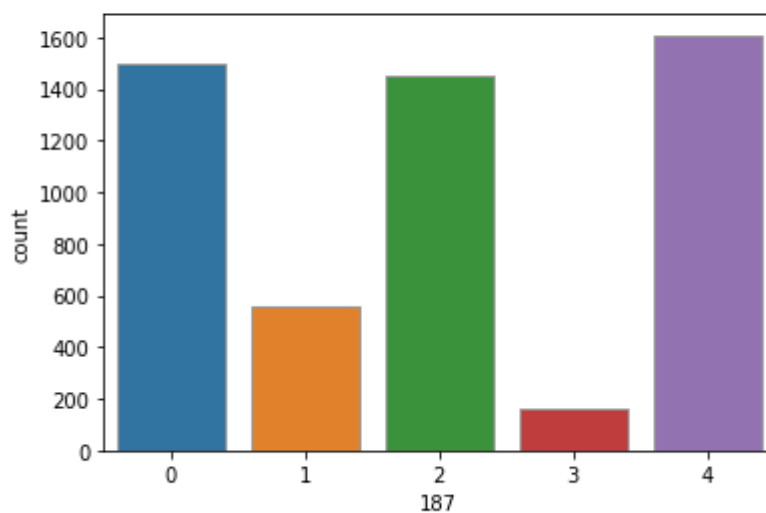


```
df[187]=df.iloc[:,187].astype(int)
c0=df.loc[df[187]==0,:].sample(1500)
c1=df.loc[df[187]==1,:]
c2=df.loc[df[187]==2,:]
c3=df.loc[df[187]==3,:]
c4=df.loc[df[187]==4,:]
ds=pd.concat([c0,c1,c2,c3,c4])
```

b) Equilibrage de données :

```
sb.countplot(x=187,edgecolor=".6",
             data=ds)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f71594573d0>



c) Sauvegarder les plots des images :

```
class1=df[df.iloc[:,187]==0.0]
for i in range(1500):
    if(os.path.exists(f'/content/drive/MyDrive/0/{i}.png')==False):
        fig=plt.figure(figsize=(8,8))
        plt.plot(class1.iloc[i,:-1])
        plt.savefig(f'/content/drive/MyDrive/0/{i}.png')
        plt.close(fig)
```

d) Lire la matrice d'image :

```
from os import listdir
from matplotlib import image
import matplotlib.pyplot as plt
from skimage.transform import resize
from skimage import color

path='/content/drive/MyDrive/train/0'
for filename in listdir(path):
    # load image
    img = image.imread(path+'/'+ filename)
    # store loaded image
    resized_image=resize(img,(32,32,3))
    imgGray = color.rgb2gray(resized_image)
    X_train_0.append(imgGray)
    y_train_0.append(0)
```

e) Sauvegarder les matrices des images :

```
y.shape
images_1 = y.reshape( 6431, 1)
import pandas as pd
data=pd.DataFrame(images_1)
d=data.to_csv('/content/drive/MyDrive/project/y_t5.csv')
```

f) Lire les fichiers csv :

```
df_train_X=pd.read_csv('/content/drive/MyDrive/project/X_train.csv',sep=',', index_col=0)
df_train_y=pd.read_csv('/content/drive/MyDrive/project/y_train.csv',sep=',', index_col=0)
df_val_X=pd.read_csv('/content/drive/MyDrive/project/X_test.csv',sep=',', index_col=0)
df_val_y=pd.read_csv('/content/drive/MyDrive/project/y_test.csv',sep=',', index_col=0)
```

g) Equilibrage de données :

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
# Division de la bd
x_train,x_test,y_train,y_test=train_test_split(df_train_X,df_train_y,test_size=0.3,random_state=0)
# Sur-échantillonnage
from imblearn.over_sampling import RandomOverSampler
rOs = RandomOverSampler()
X_ro, y_ro = rOs.fit_resample(x_train, y_train)
x_rtest, y_rtest = rOs.fit_resample(x_test, y_test)
```

```
X_train=np.asarray(X_ro)
y_train=np.asarray(y_ro)
X_test=np.asarray(x_rtest)
y_test=np.asarray(y_rtest)
```

h) Transformer le csv vers matrices :

```
X_train=X_train.reshape(22360, 32,32,1)
```

```
y_train=y_train.reshape(22360,1)
```

```
X_test=X_test.reshape(9795, 32,32,1)
```

```
y_test=y_test.reshape(9795,1)
```

```
X,y = r0s.fit_resample(df_val_X,df_val_y)
```

i) Redimensionner l'image :

```
from skimage.transform import resize
X_train=resize(X_train,(22360,32,32,1))
X_test=resize(X_test,(9795,32,32,1))
X_val=resize(X_val,(8040,32,32,1))
```

2. Application de module :

```
[ ] module =Sequential()
```

```
[ ] module.add(Conv2D(64,3,input_shape=(32,32,1)))
module.add(Activation('relu'))
module.add(MaxPooling2D(pool_size=(2,2)))
```

```
[ ] module.add(Flatten())
```

```
▶ module.add(Dense(200))
module.add(Dense(5))
module.add(Activation('softmax'))
module.compile(loss='sparse_categorical_crossentropy',optimizer="Adam",metrics=['sparse_categorical_accuracy'])
```

```
[ ] module.fit(X_train,y_train,batch_size=32,epochs=1)
```

```
699/699 [=====] - 29s 41ms/step - loss: 0.6504 - sparse_categorical_accuracy: 0.7708
<keras.callbacks.History at 0x7fb383b30610>
```

```
[ ] test_loss,test_accuracy=module.evaluate(X_test,y_test)
```

```
307/307 [=====] - 3s 10ms/step - loss: 0.5830 - sparse_categorical_accuracy: 0.7863
```

```
[ ] test_loss,test_accuracy=module.evaluate(X_val,y_val)
```

```
252/252 [=====] - 3s 10ms/step - loss: 1.0621 - sparse_categorical_accuracy: 0.6361
```

Le choix de ce module parce que ce module est équilibrée est dans l'accuracy de 77% pour le test et de 63% en validation, ces mieux qu'un autre je trouve avec 90% en test et moins de 50% en validation.

3. Transformer vers un module Lite :

```
module.save('/content/drive/MyDrive/tfmodel4.lt')
```

```
INFO:tensorflow:Assets written to: /content/drive/MyDrive/tfmodel4.lt/assets
```

```
[ ] del module
```

```
[ ] module = load_model('/content/drive/MyDrive/tfmodel4.lt')
```

```
[ ] test_loss, test_accuracy = module.evaluate(X_val, y_val)
```

```
252/252 [=====] - 4s 15ms/step - loss: 1.0621 - sparse_categorical_accuracy: 0.6361
```

```
[ ] file = "/content/drive/MyDrive/tfmodel4.lt"
tf.saved_model.save(module, file)
converter = tf.lite.TFLiteConverter.from_saved_model(file)
tflite_model = converter.convert()
```

```
INFO:tensorflow:Assets written to: /content/drive/MyDrive/tfmodel4.lt/assets
WARNING:absl:Buffer deduplication procedure will be skipped when flatbuffer library is not properly loaded
```

```
import pathlib
tflite_model_file = pathlib.Path('/content/drive/MyDrive/model4.tflite')
tflite_model_file.write_bytes(tflite_model)
```

```
11529008
```

```
tflite_model = '/content/drive/MyDrive/model4.tflite'
interpreter = tf.lite.Interpreter(model_path=tflite_model)
interpreter.allocate_tensors()
```

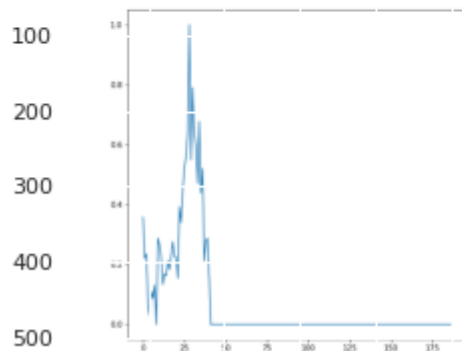
```
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

4. Prédiction d'une classe :

```
from google.colab.patches import cv2_imshow
```

```
b="/content/drive/MyDrive/test/2/31.png"  
img=plt.imread(b)  
plt.imshow(img)
```

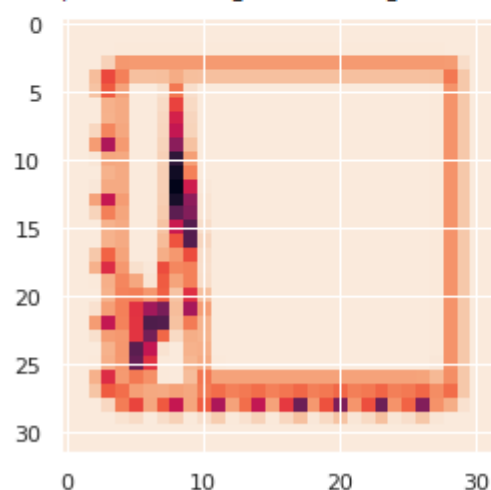
```
<matplotlib.image.AxesImage at 0x7fb382c3f950>  
0
```



```
from skimage import color  
from skimage.transform import resize  
resized_image = resize(img,(32,32,3))  
imgGray = color.rgb2gray(resized_image)
```

```
plt.imshow(imgGray)
```

```
<matplotlib.image.AxesImage at 0x7fb383349190>
```



```
X_train.shape
```

```
(22360, 32, 32, 1)
```

```
imgGray.shape
```

```
(32, 32)
```

```
g=imgGray.reshape((1,32,32,1))
```

```
to_predict = np.array(np.array(g),dtype='float32')
interpreter.set_tensor(input_details[0]['index'], to_predict)
interpreter.invoke()
tflite_results = interpreter.get_tensor(output_details[0]['index'])
print(tflite_results)
```

```
[[7.8095468e-03 3.2197237e-02 6.8997997e-01 5.8053754e-04 2.6943266e-01]]
```

```
list=[0,1,2,3,4]
X=tflite_results
for i in range(5):
    for j in range(5):
        if X[0][list[i]] > X[0][list[j]]:
            temp=list[i]
            list[i]=list[j]
            list[j]=temp
```

```
list
```

```
[2, 4, 1, 0, 3]
```

```
class_list=["Class 0","Class 1","Class 2","Class 3","Class 4"]
```

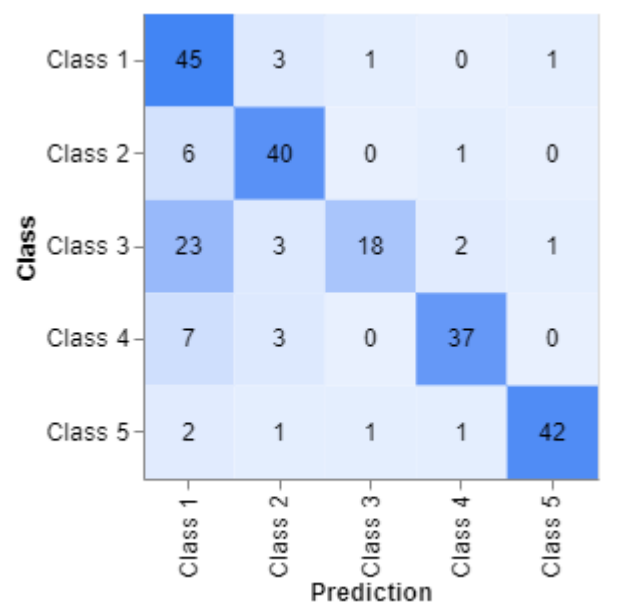
```
for i in range(5):
    print(class_list[list[i]], " sa predictions ",round(tflite_results[0][list[i]]*100 , 2),"%")
```

```
Class 2 sa predictions 69.0 %
Class 4 sa predictions 26.94 %
Class 1 sa predictions 3.22 %
Class 0 sa predictions 0.78 %
Class 3 sa predictions 0.06 %
```

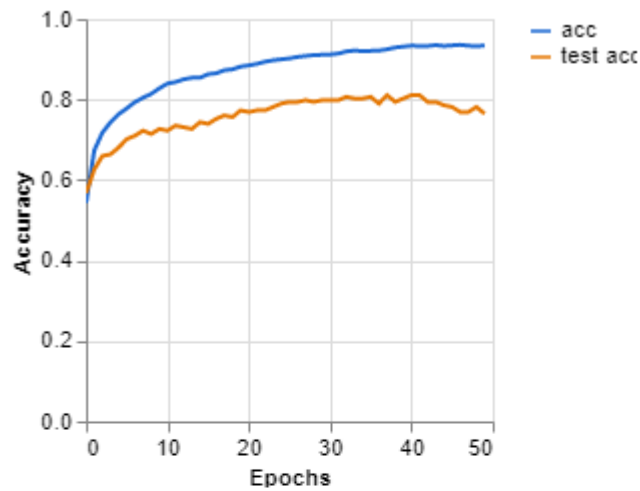

5. Comparaison de module avec le module du Teachable Machine : Accuracy per class

CLASS	ACCURACY	# SAMPLES
Class 1	0.90	50
Class 2	0.85	47
Class 3	0.38	47
Class 4	0.79	47
Class 5	0.89	47

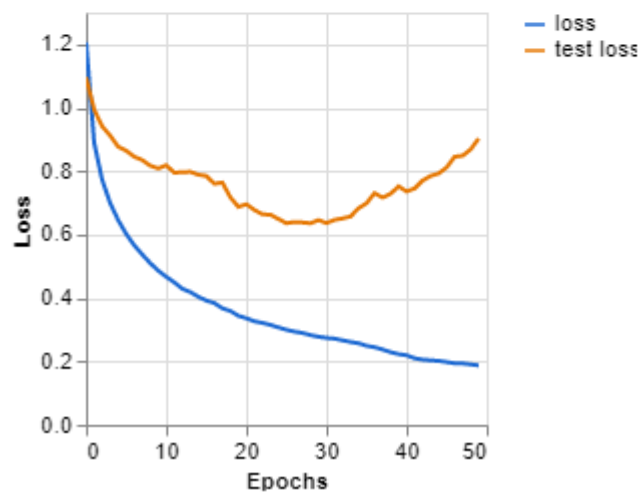
Confusion Matrix



Accuracy per epoch



Loss per epoch



```
test_loss,test_accuracy=model.evaluate(X_val,y_val)
```

4/4 [=====] - 3s 222ms/step - loss: 6.3860 - sparse_categorical_accuracy: 0.0100

Accuracy(Teachablemachine_test)=0.7647=76.47%

Accuracy(Teachablemachine_val)=0.01=1%

Accuracy(Module_test)=77%

Accuracy(Module_val)=77%

Teachablemachine_test il souffre de overfitting.

IV. Règles d'association :

1. Data Engineering :

```
# Read in data and display first 5 rows
data = pd.read_csv("probleme card.csv")
data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

→

association rule

data														
	AgeGrp_1.0	AgeGrp_2.0	AgeGrp_3.0	AgeGrp_4.0	AgeGrp_5.0	Sex_0	Sex_1	Restecg_0	Restecg_1	Restecg_2	...	ThalachGrp_5.0	OldpeakGrp_1.0	C
0	0	0	0	1	0	0	1	1	0	0	...	0	0	
1	1	0	0	0	0	0	1	0	1	0	...	1	0	
2	0	1	0	0	0	1	0	1	0	0	...	0	1	
3	0	0	1	0	0	0	1	0	1	0	...	1	1	
4	0	0	1	0	0	1	0	0	1	0	...	0	1	
...	
298	0	0	1	0	0	1	0	0	1	0	...	0	1	
299	0	1	0	0	0	0	1	0	1	0	...	0	1	
300	0	0	0	0	1	0	1	0	1	0	...	0	0	
301	0	0	1	0	0	0	1	0	1	0	...	0	1	
302	0	0	1	0	0	1	0	1	0	0	...	0	1	

303 rows × 46 columns

data.columns

```
Index(['AgeGrp_1.0', 'AgeGrp_2.0', 'AgeGrp_3.0', 'AgeGrp_4.0', 'AgeGrp_5.0',
      'Sex_0', 'Sex_1', 'Restecg_0', 'Restecg_1', 'Restecg_2', 'CP_0', 'CP_1',
      'CP_2', 'CP_3', 'Slope_0', 'Slope_1', 'Slope_2', 'Ca_0', 'Ca_1', 'Ca_2',
      'Ca_3', 'Ca_4', 'Thal_0', 'Thal_1', 'Thal_2', 'Thal_3',
      'TrestbpsGrp_1.0', 'TrestbpsGrp_2.0', 'CholGrp_1.0', 'CholGrp_2.0',
      'CholGrp_3.0', 'CholGrp_4.0', 'ThalachGrp_1.0', 'ThalachGrp_2.0',
      'ThalachGrp_3.0', 'ThalachGrp_4.0', 'ThalachGrp_5.0', 'OldpeakGrp_1.0',
      'OldpeakGrp_2.0', 'OldpeakGrp_3.0', 'Fbs_0', 'Fbs_1', 'Exang_0',
      'Exang_1', 'Target_0', 'Target_1'],
      dtype='object')
```

2. Règle d'association

```
: import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, fpmax, fpgrowth

#frequent_itemsets = fpgrowth(df, min_support=0.6, use_colnames=True)
### alternatively:
frequent_itemsets = apriori(data, min_support=0.3, use_colnames=True)
#frequent_itemsets = fpmax(df, min_support=0.6, use_colnames=True)

frequent_itemsets
```

```
:
```

	support	itemsets
0	0.320132	(AgeGrp_3.0)
1	0.349835	(AgeGrp_4.0)
2	0.316832	(Sex_0)
3	0.683168	(Sex_1)
4	0.485149	(Restecg_0)
...

```
: from mlxtend.frequent_patterns import association_rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.2)
rules
```

```
:
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Sex_1)	(Thal_3)	0.683168	0.386139	0.336634	0.492754	1.276106	0.072836	1.210184
1	(Thal_3)	(Sex_1)	0.386139	0.683168	0.336634	0.871795	1.276106	0.072836	2.471287
2	(Target_0)	(Sex_1)	0.455446	0.683168	0.376238	0.826087	1.209200	0.065092	1.821782
3	(Sex_1)	(Target_0)	0.683168	0.455446	0.376238	0.550725	1.209200	0.065092	1.212073
4	(Target_0)	(CP_0)	0.455446	0.471947	0.343234	0.753623	1.596838	0.128288	2.143273
...
669	(OldpeakGrp_1.0, Fbs_0)	(TrestbpsGrp_1.0, Target_1, Exang_0)	0.712871	0.399340	0.346535	0.486111	1.217287	0.061857	1.168852
670	(Target_1, OldpeakGrp_1.0)	(TrestbpsGrp_1.0, Fbs_0, Exang_0)	0.521452	0.485149	0.346535	0.664557	1.369801	0.093553	1.534840
671	(Target_1, Fbs_0)	(TrestbpsGrp_1.0, OldpeakGrp_1.0, Exang_0)	0.468647	0.511551	0.346535	0.739437	1.445479	0.106798	1.874587
672	(Exang_0)	(TrestbpsGrp_1.0, OldpeakGrp_1.0, Target_1, Fb...	0.673267	0.409241	0.346535	0.514706	1.257709	0.071006	1.217322

```

: rules["antecedent_len"] = rules["antecedents"].apply(lambda x: len(x))
rules["consequents_len"] = rules["consequents"].apply(lambda x: len(x))
rules

```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecedent_len	consequents_len
0	(Sex_1)	(Thal_3)	0.683168	0.386139	0.336634	0.492754	1.276106	0.072836	1.210184	1	1
1	(Thal_3)	(Sex_1)	0.386139	0.683168	0.336634	0.871795	1.276106	0.072836	2.471287	1	1
2	(Target_0)	(Sex_1)	0.455446	0.683168	0.376238	0.826087	1.209200	0.065092	1.821782	1	1
3	(Sex_1)	(Target_0)	0.683168	0.455446	0.376238	0.550725	1.209200	0.065092	1.212073	1	1
4	(Target_0)	(CP_0)	0.455446	0.471947	0.343234	0.753623	1.596838	0.128288	2.143273	1	1
...
669	(OldpeakGrp_1.0, Fbs_0)	(TrestbpsGrp_1.0, Target_1, Exang_0)	0.712871	0.399340	0.346535	0.486111	1.217287	0.061857	1.168852	2	3
670	(Target_1, OldpeakGrp_1.0)	(TrestbpsGrp_1.0, Fbs_0, Exang_0)	0.521452	0.485149	0.346535	0.664557	1.369801	0.093553	1.534840	2	3
671	(Target_1, Fbs_0)	(TrestbpsGrp_1.0, OldpeakGrp_1.0, Exang_0)	0.468647	0.511551	0.346535	0.739437	1.445479	0.106798	1.874587	2	3

```

: rules=rules[ (rules['antecedent_len'] < 4) &
               (rules['consequents_len'] == 1)&
               (rules['confidence']>0.9)&
               (rules['lift'] > 1.2)]
rules

```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecedent_len	consequents_len
228	(TrestbpsGrp_1.0, Ca_0, Thal_2)	(Target_1)	0.323432	0.544554	0.300330	0.928571	1.705195	0.124204	6.376238	3	1
262	(OldpeakGrp_1.0, Ca_0, Thal_2)	(Target_1)	0.353135	0.544554	0.323432	0.915888	1.681903	0.131131	5.414741	3	1

```

: antecedent_sele = rules['antecedents'] == frozenset({'AgeGrp_1.0', 'AgeGrp_2.0', 'AgeGrp_3.0', 'AgeGrp_4.0', 'AgeGrp_5.0',
'Sex_0', 'Sex_1', 'Restecg_0', 'Restecg_1', 'Restecg_2', 'CP_0', 'CP_1',
'CP_2', 'CP_3', 'Slope_0', 'Slope_1', 'Slope_2', 'Ca_0', 'Ca_1', 'Ca_2',
'Ca_3', 'Ca_4', 'Thal_0', 'Thal_1', 'Thal_2', 'Thal_3',
'TrestbpsGrp_1.0', 'TrestbpsGrp_2.0', 'CholGrp_1.0', 'CholGrp_2.0',
'CholGrp_3.0', 'CholGrp_4.0', 'ThalachGrp_1.0', 'ThalachGrp_2.0',
'ThalachGrp_3.0', 'ThalachGrp_4.0', 'ThalachGrp_5.0', 'OldpeakGrp_1.0',
'OldpeakGrp_2.0', 'OldpeakGrp_3.0', 'Fbs_0', 'Fbs_1', 'Exang_0',
'Exang_1'}) # or frozenset({'Kidney Beans', 'Onion'})
consequent_sele = rules['consequents'] == frozenset({'Target_1'})
#final_sele = (consequent_sele)

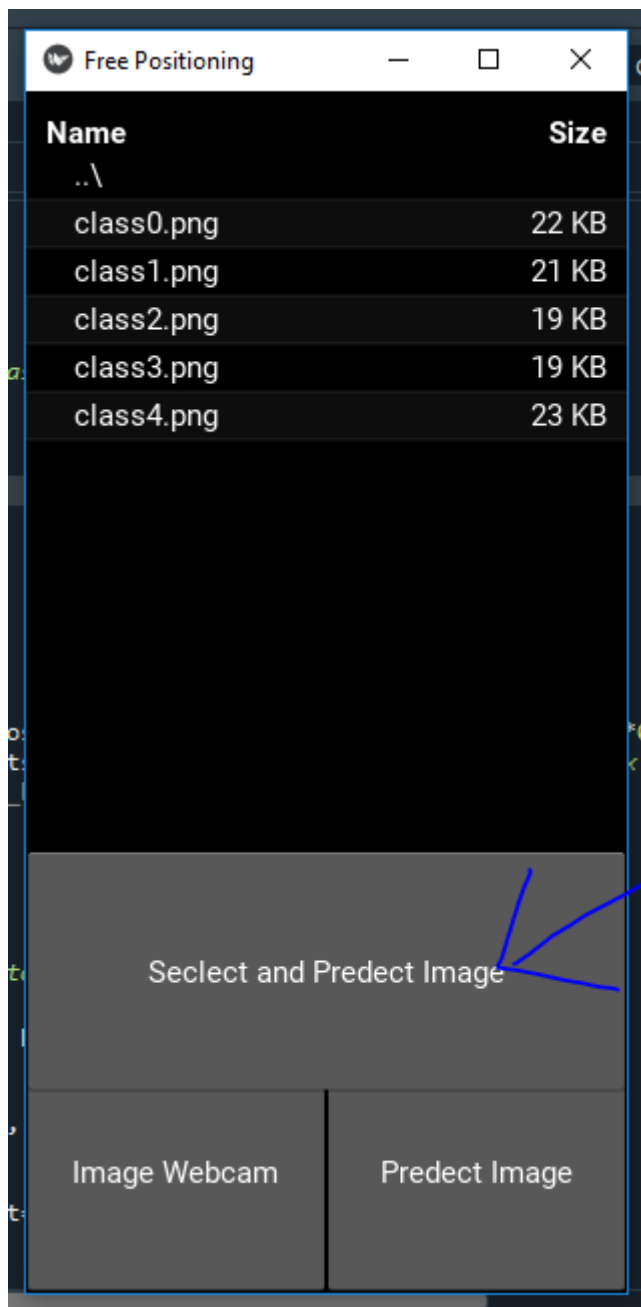
f=rules.loc[consequent_sele]
f

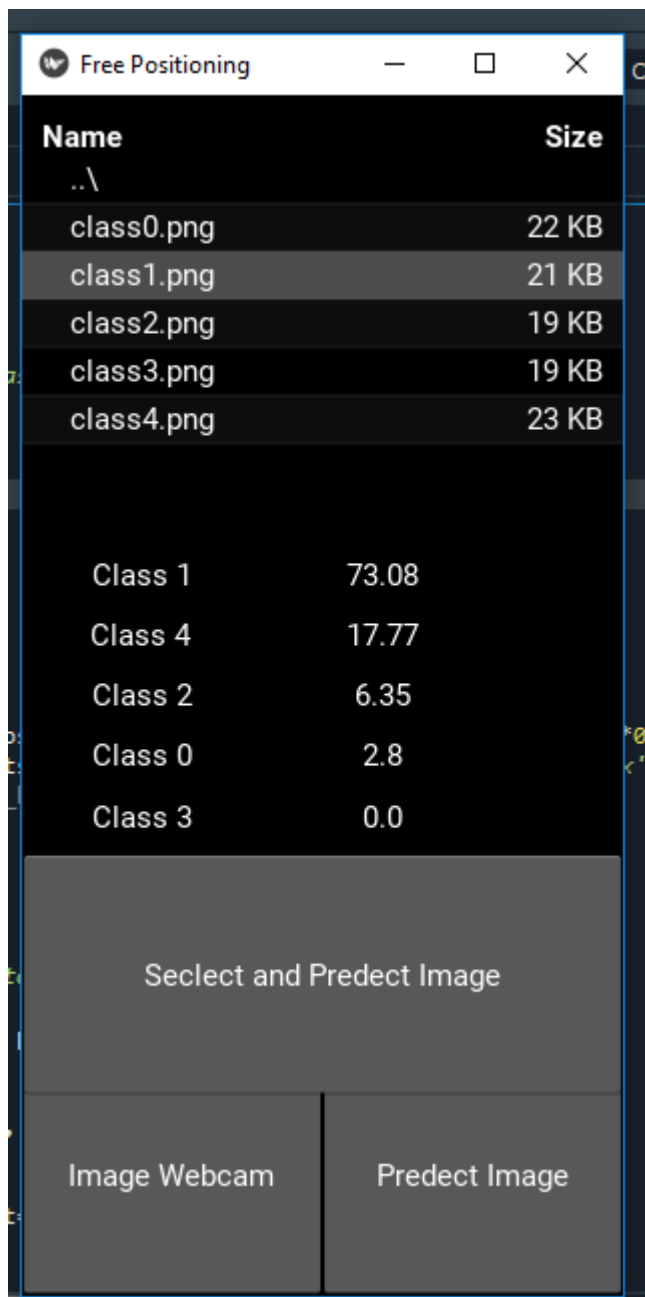
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecedent_len	consequents_len
228	(TrestbpsGrp_1.0, Ca_0, Thal_2)	(Target_1)	0.323432	0.544554	0.300330	0.928571	1.705195	0.124204	6.376238	3	1
262	(OldpeakGrp_1.0, Ca_0, Thal_2)	(Target_1)	0.353135	0.544554	0.323432	0.915888	1.681903	0.131131	5.414741	3	1

V. Réaliser d'une application mobile kivy qui utilise le module lite :

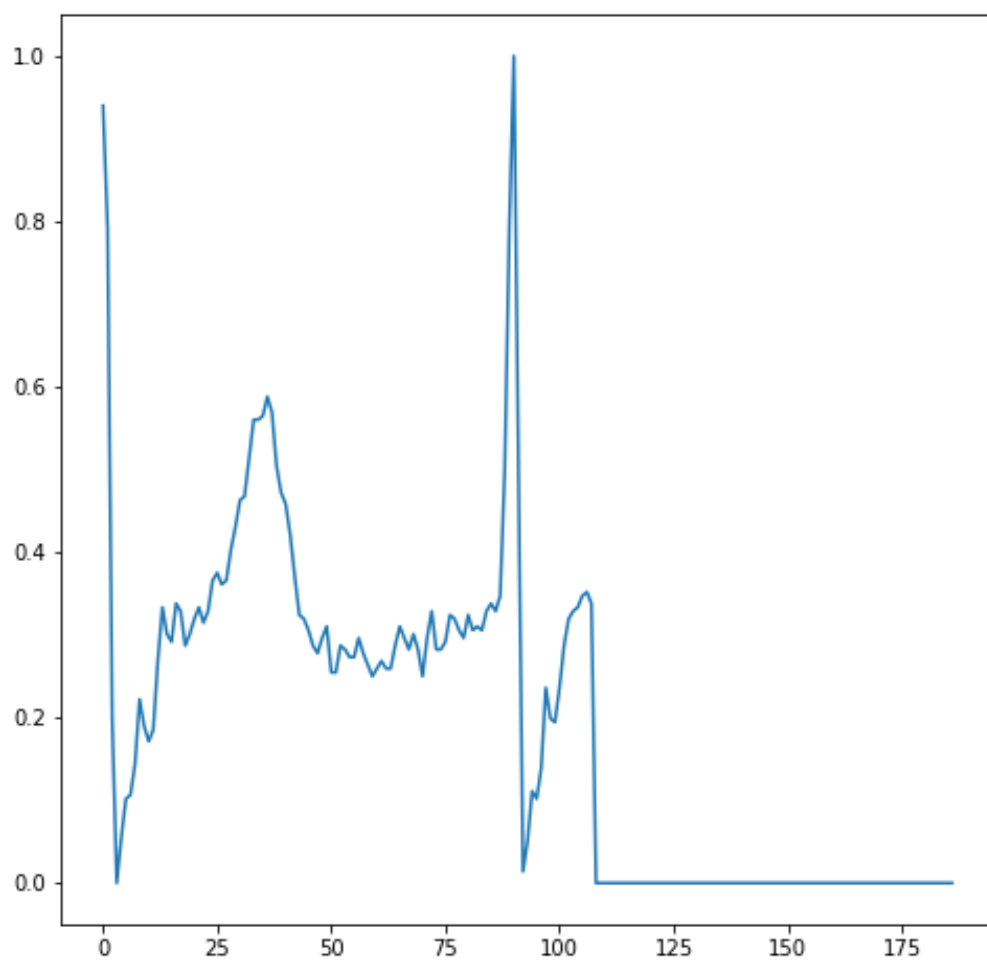
1. Using selection :

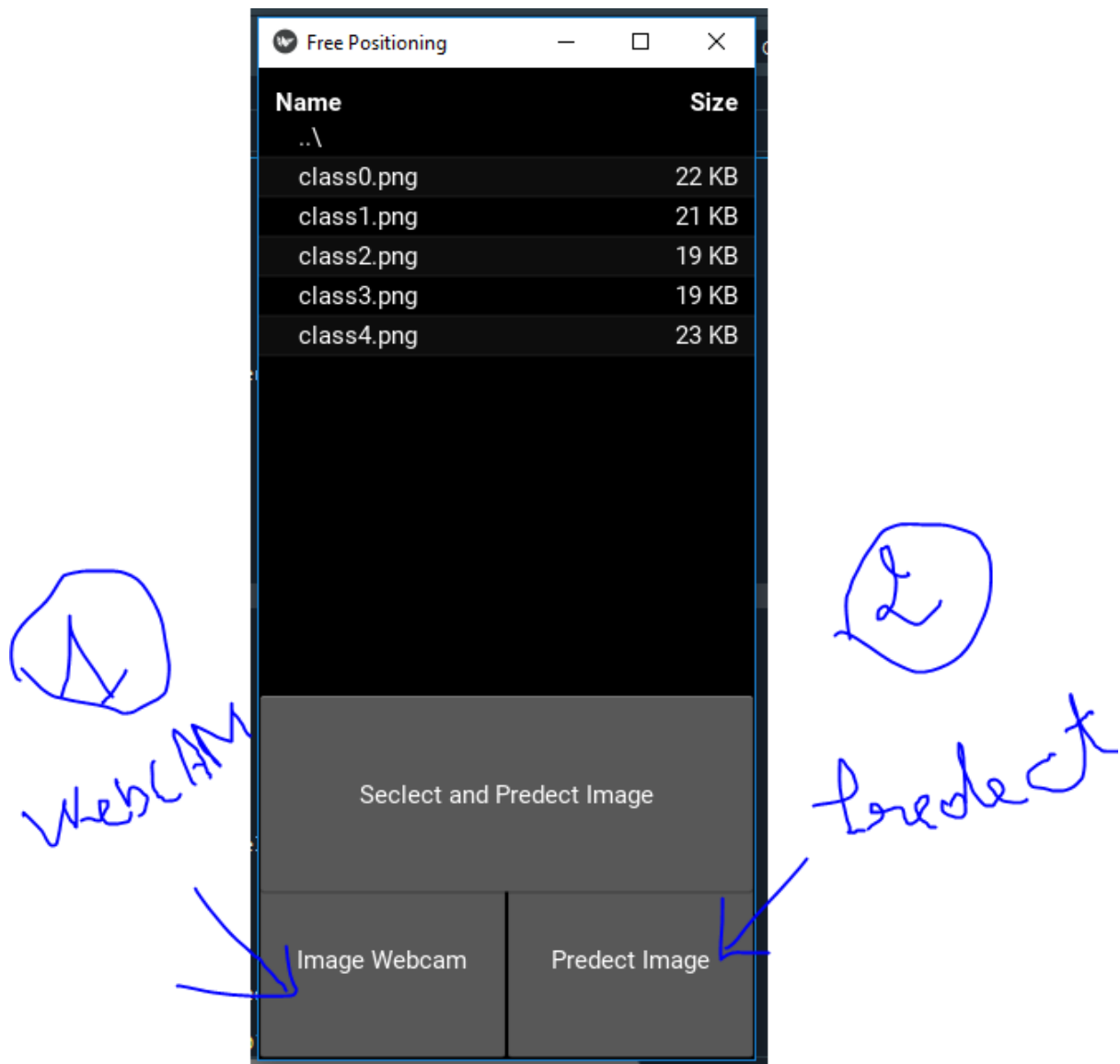




2. Using the webcam :

Pour Cette image de class1 :



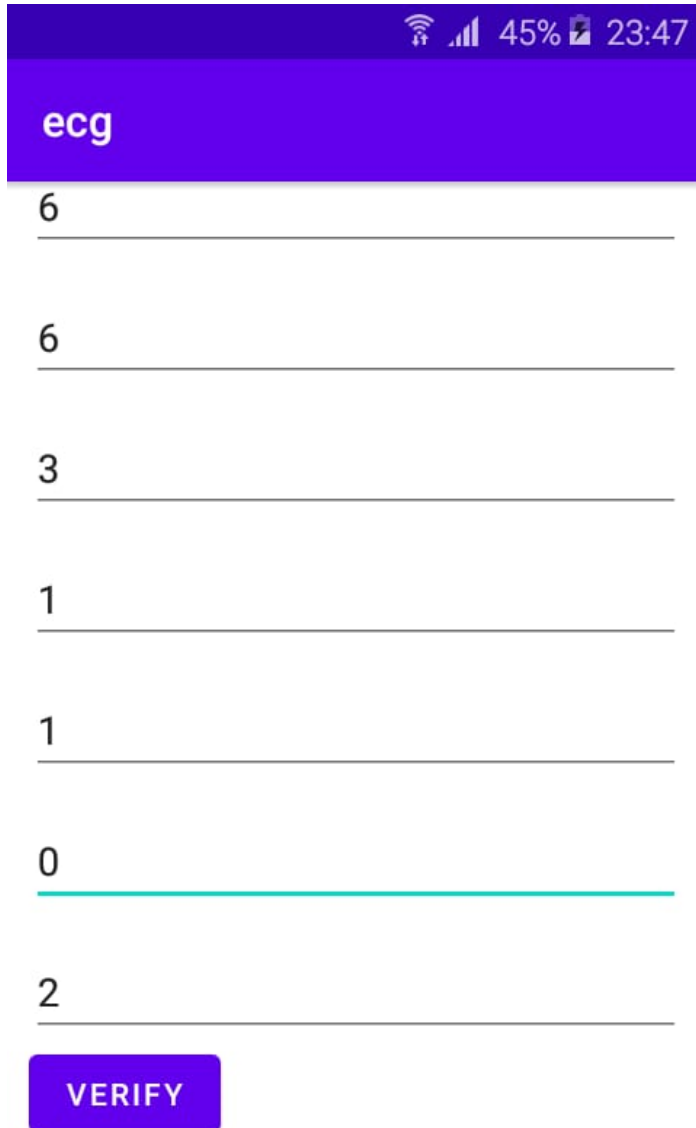


3. Déploiement du module :

En utilisant les codes qui sont dans le fichier jpybn « KivyApp to APK.ipynb ».

VI. Réaliser d'une application mobile avec Android Studio qui utilise le module lite :

1. Formulaire des règles d'association :



ecg

6

6

3

1

1


0

2

VERIFY

2. Résultat :

a) le cas du nom maladie :



48%

23:54

ecg

6

6

3

1

1

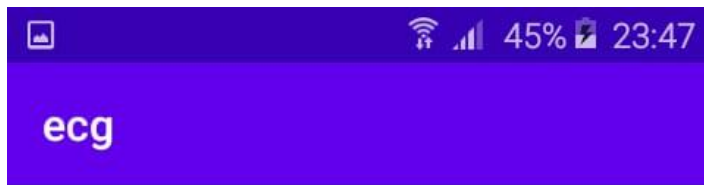
1

2

C'est bon

VERIFY

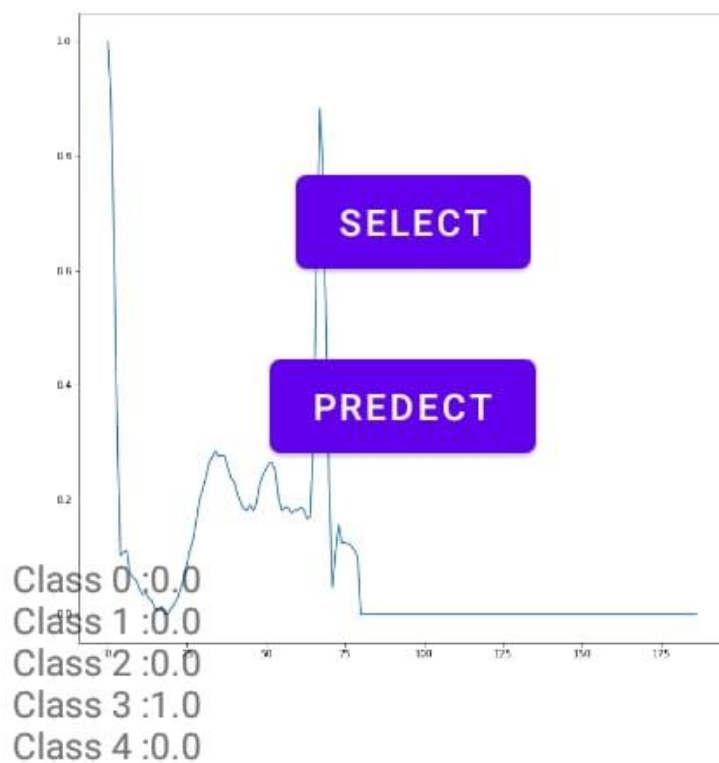
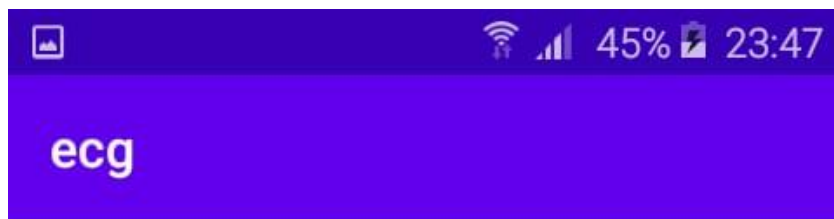
b) le cas du nom maladie :
Pour le cas de la maladie en affiche ECG.



SELECT

PREDECT

TextView



C'est Vais