



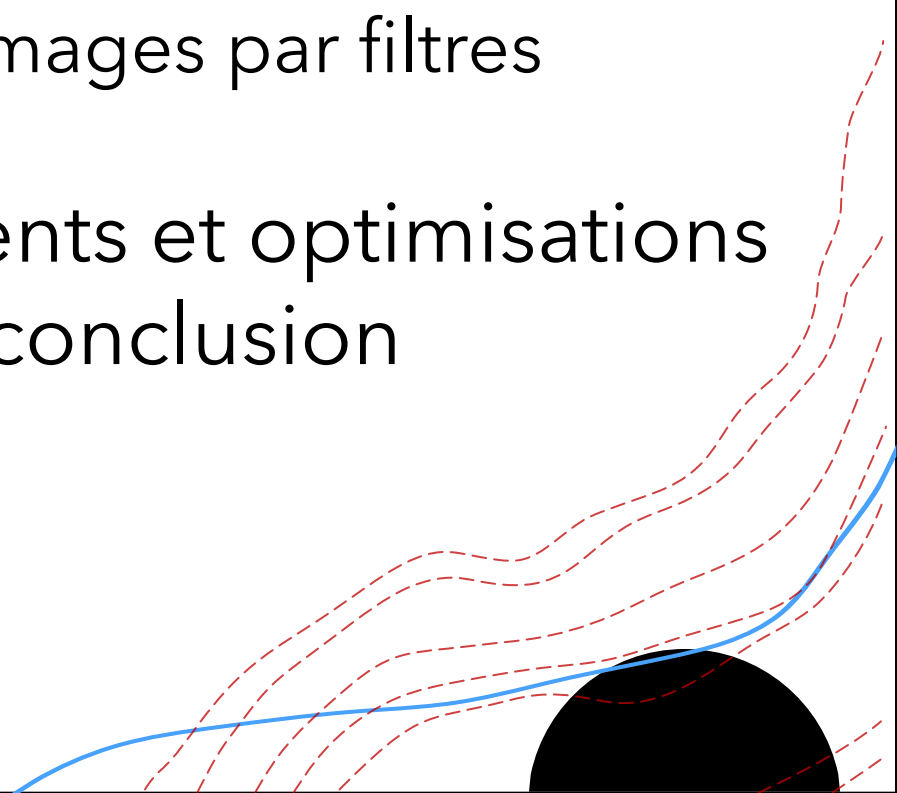
Simulation de l'interprétation des images médicales par l'intelligence artificielle

IMANE BENRAZZOUK





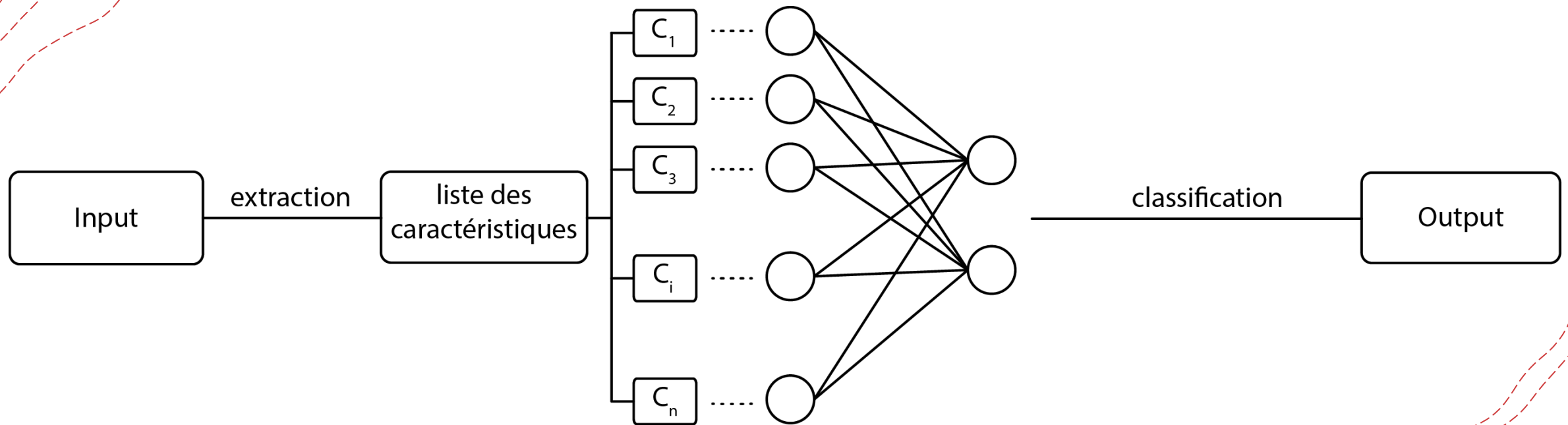
PLAN

- Introduction
 - ✓ Classification traditionnelle des images
 - ✓ Présentation de la base de données
 - ✓ Traitement des images par filtres
 - ✓ Problématique
 - Modèles intelligents et optimisations
 - Comparaison et conclusion
 - Annexes
- 

The image features a minimalist design with a white background. In the top-left corner, there is a solid black circle. In the bottom-right corner, there is another solid black circle. Several red dashed lines are scattered across the page, primarily in the corners, creating a sense of movement and depth. The word "INTRODUCTION" is centered in a large, bold, black, sans-serif font.

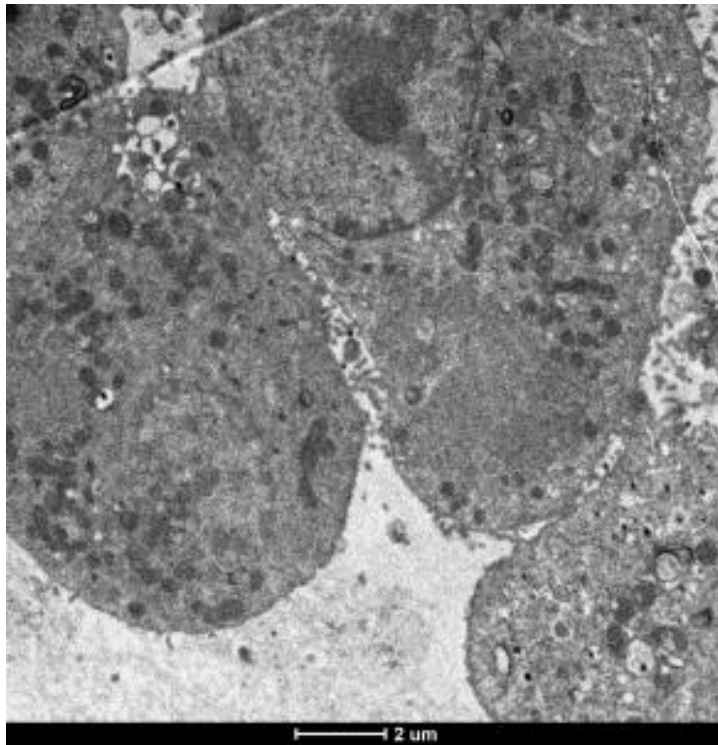
INTRODUCTION

Classification traditionnelle des images



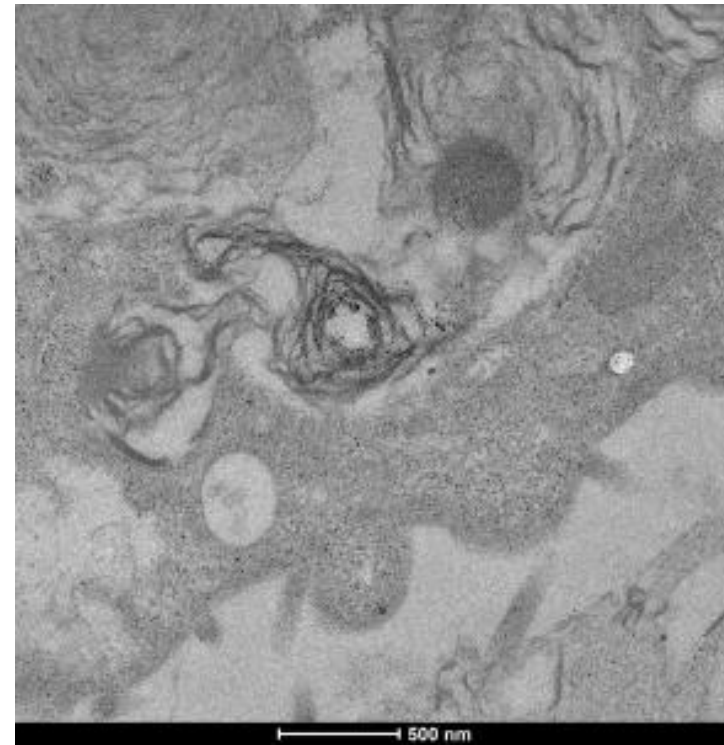
Présentation de la base de données

Exemple de cellule atteinte « infected »



Étiquette = 1

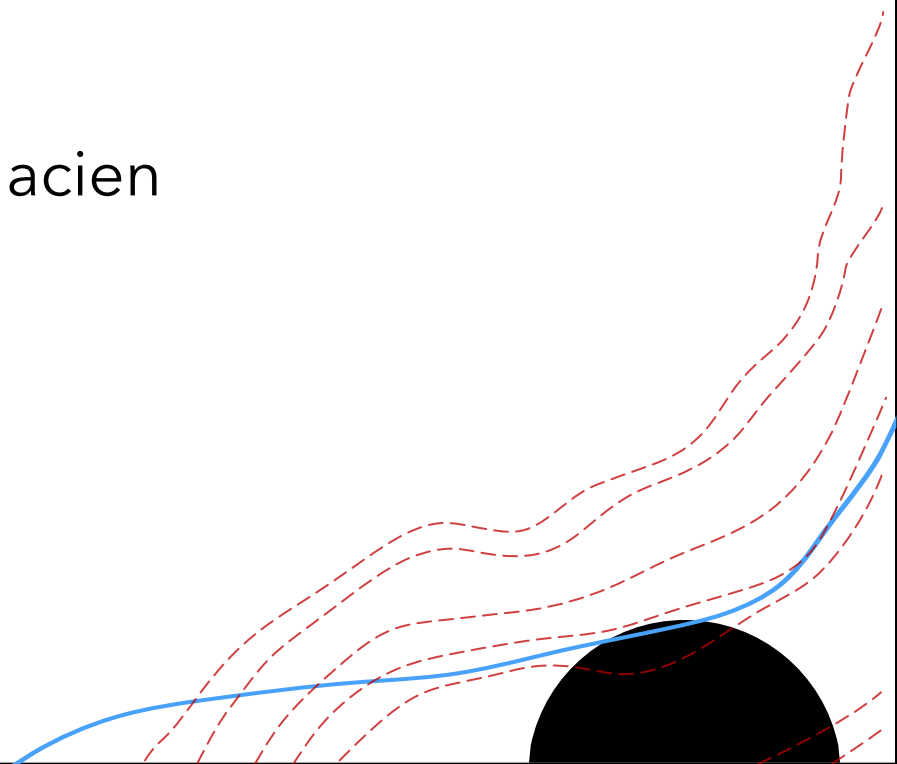
Exemple de cellule saine « control »



Étiquette = 0



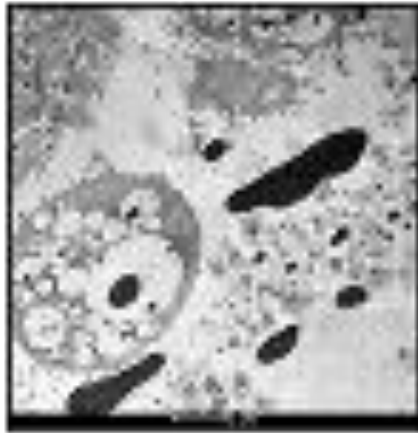
Traitement des images

- Détection du contour par filtres de Sobel et Laplacien
 - Détection des zones sombres (Thresholding)
 - Lissage des images par le filtre Gaussien
- 

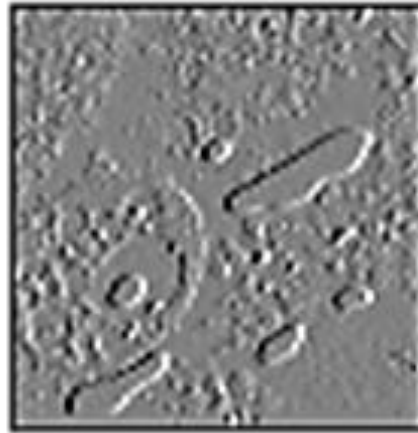
Détection du contour

Filtre de Sobel

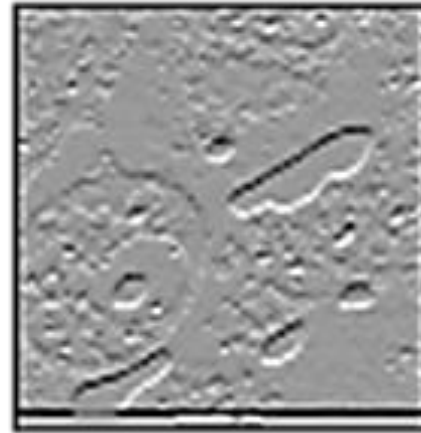
Original



Sobel X



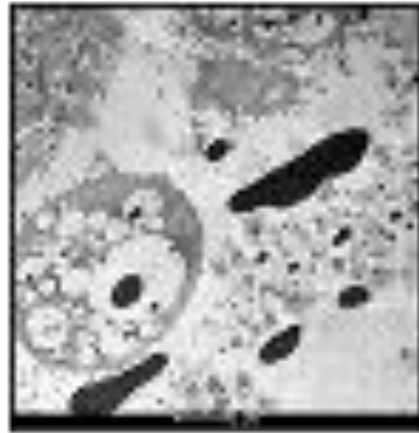
Sobel Y



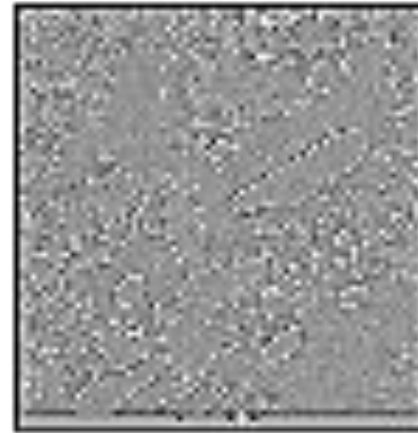
Détection du contour

Filtre Laplacien

Original

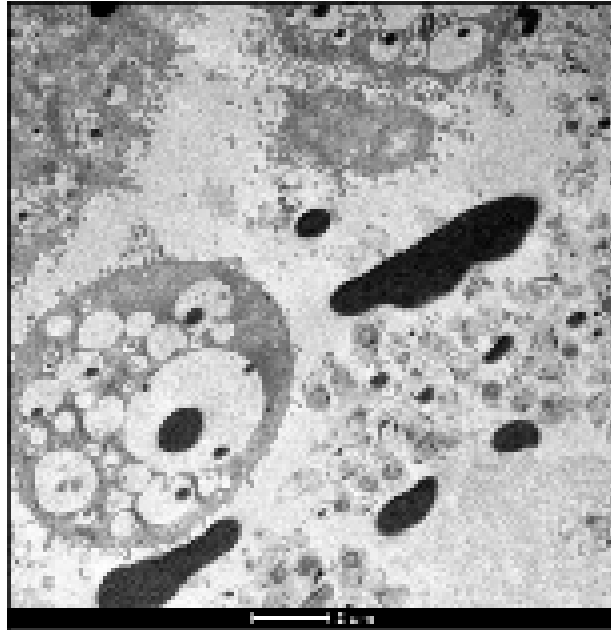


Laplacian

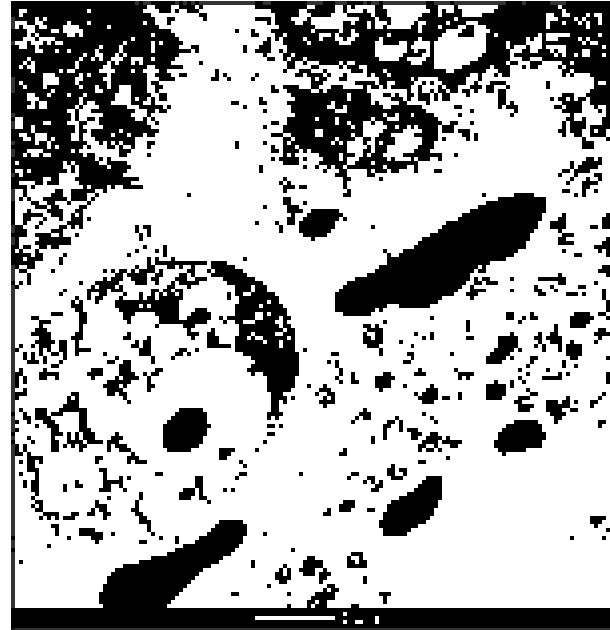


Détection des zones sombres (Thresholding)

Original

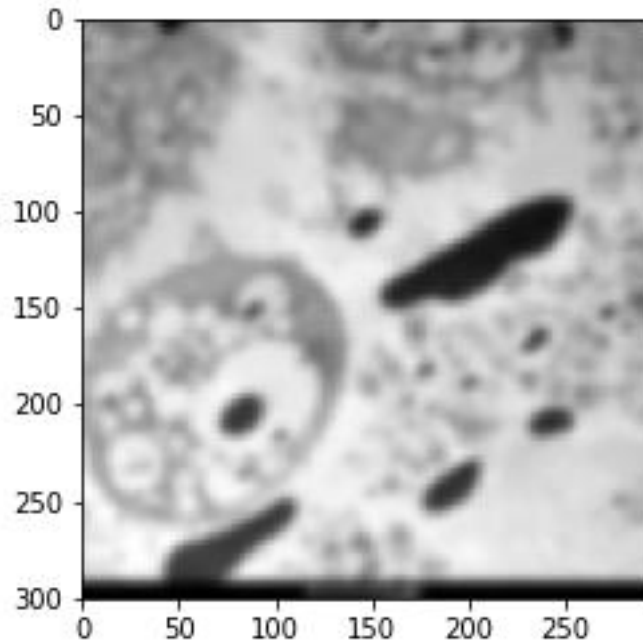


Threshold

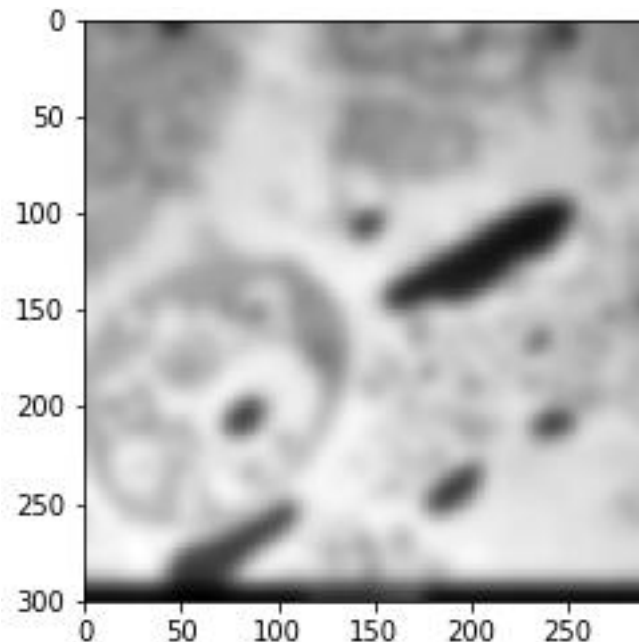


Lissage des images

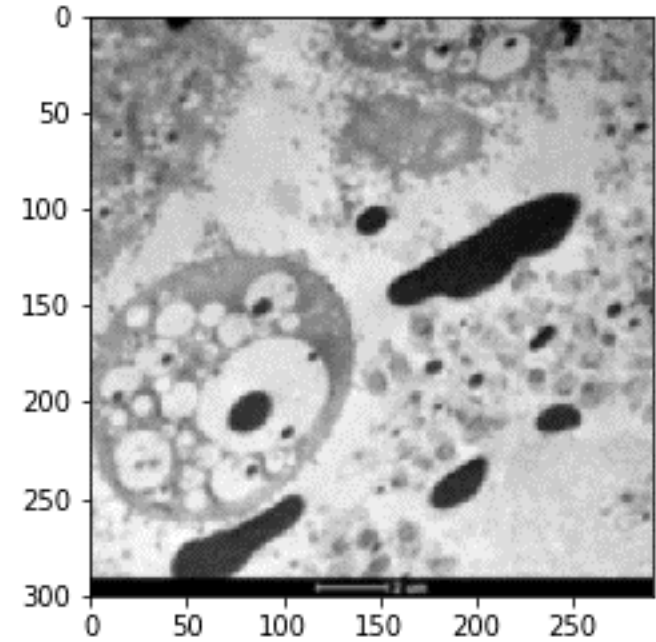
Filtre Gaussien



sigma=3.0



sigma=5.0

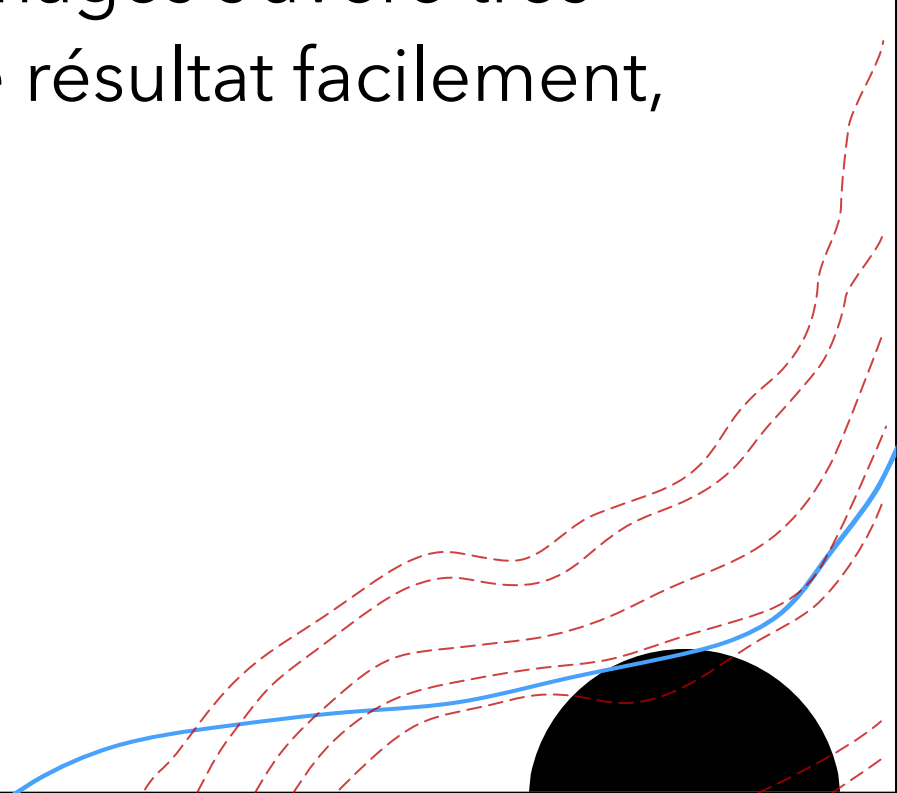


sigma=1.0



Problématique

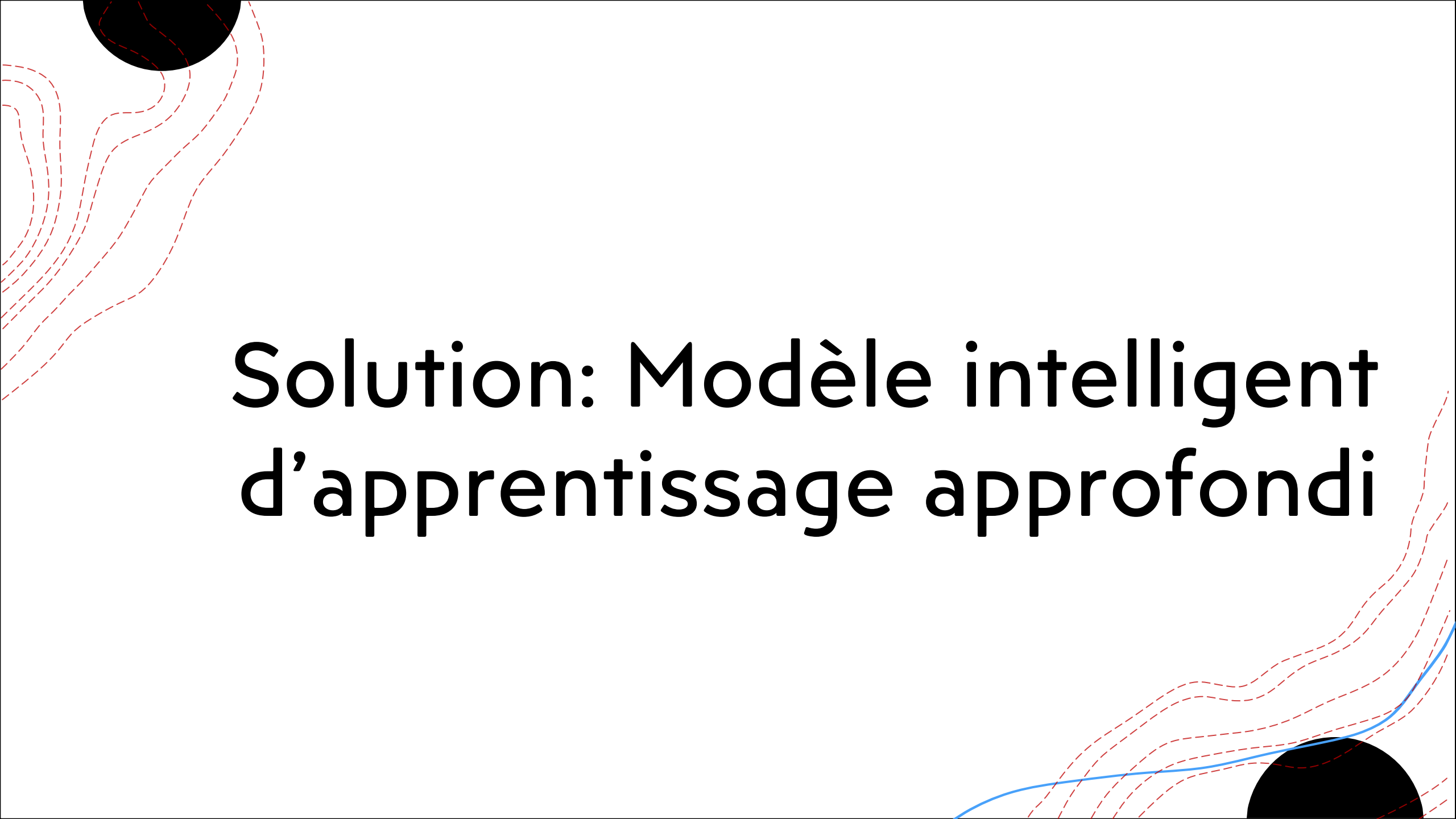
Faire l'inventaire des caractéristiques des images s'avère très consommant, alors que le but est d'obtenir le résultat facilement, effectivement et immédiatement.



Méthode d'évaluation

+ Matrice de confusion:

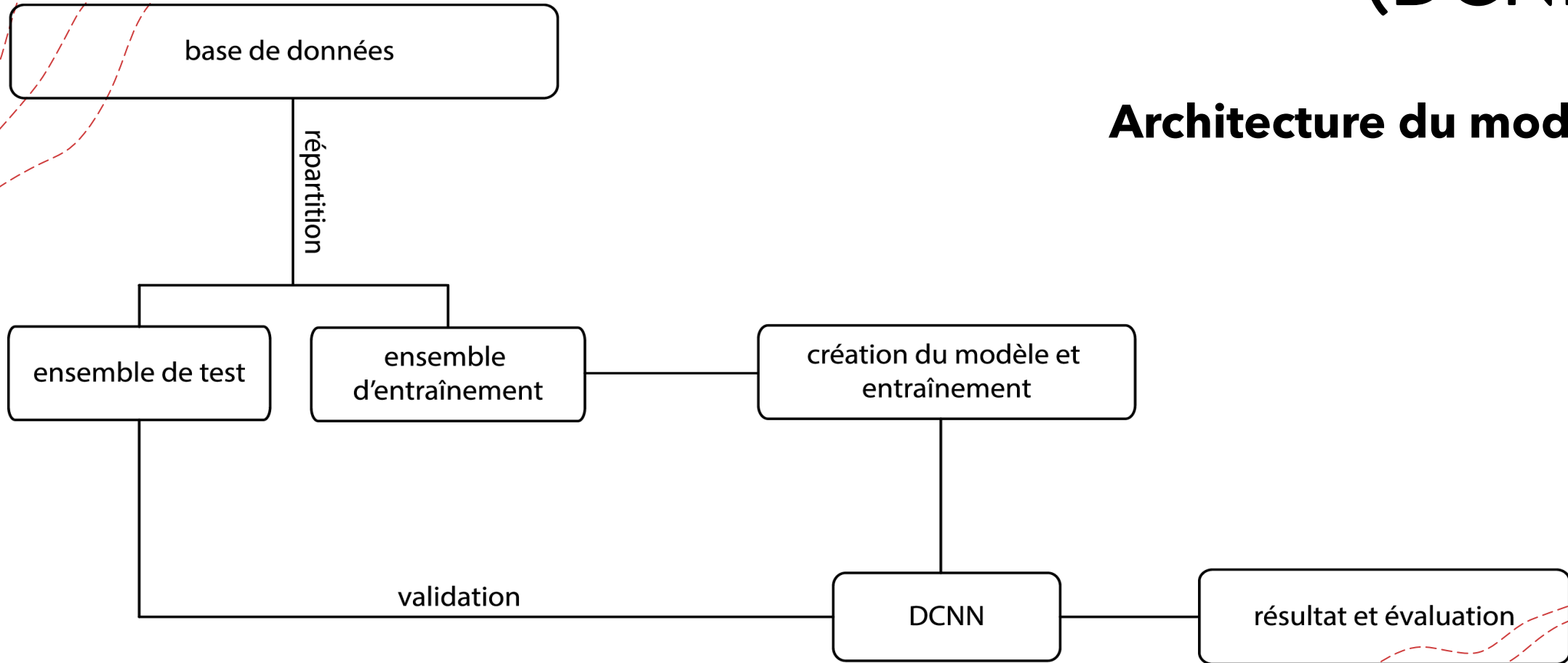
		Réalité	
		Sain 0	Infecté 1
Prédiction	Sain 0	True Negative	False Negative
	Infecté 1	False Positive	True Positive



**Solution: Modèle intelligent
d'apprentissage approfondi**

Deep Convolutional Neural Network (DCNN)

Architecture du modèle



Deep Convolutional Neural Network (DCNN)

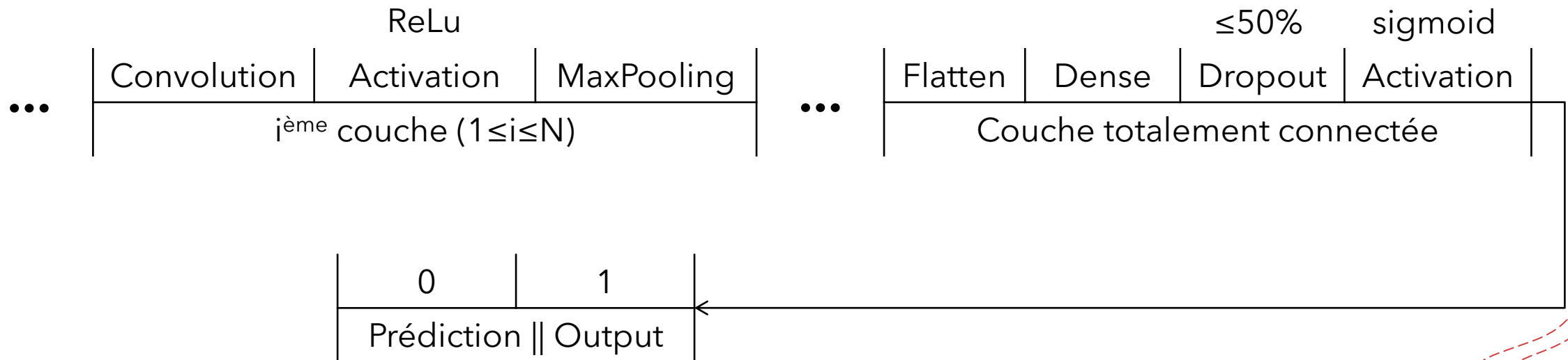
Répartition de la base de données

```
#splitting the dataset for the training
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(dataset, labels, test_size = 0.20, random_state = 0)
```

+ L'ensemble de test/validation constitue 20% de la base de données initiale.

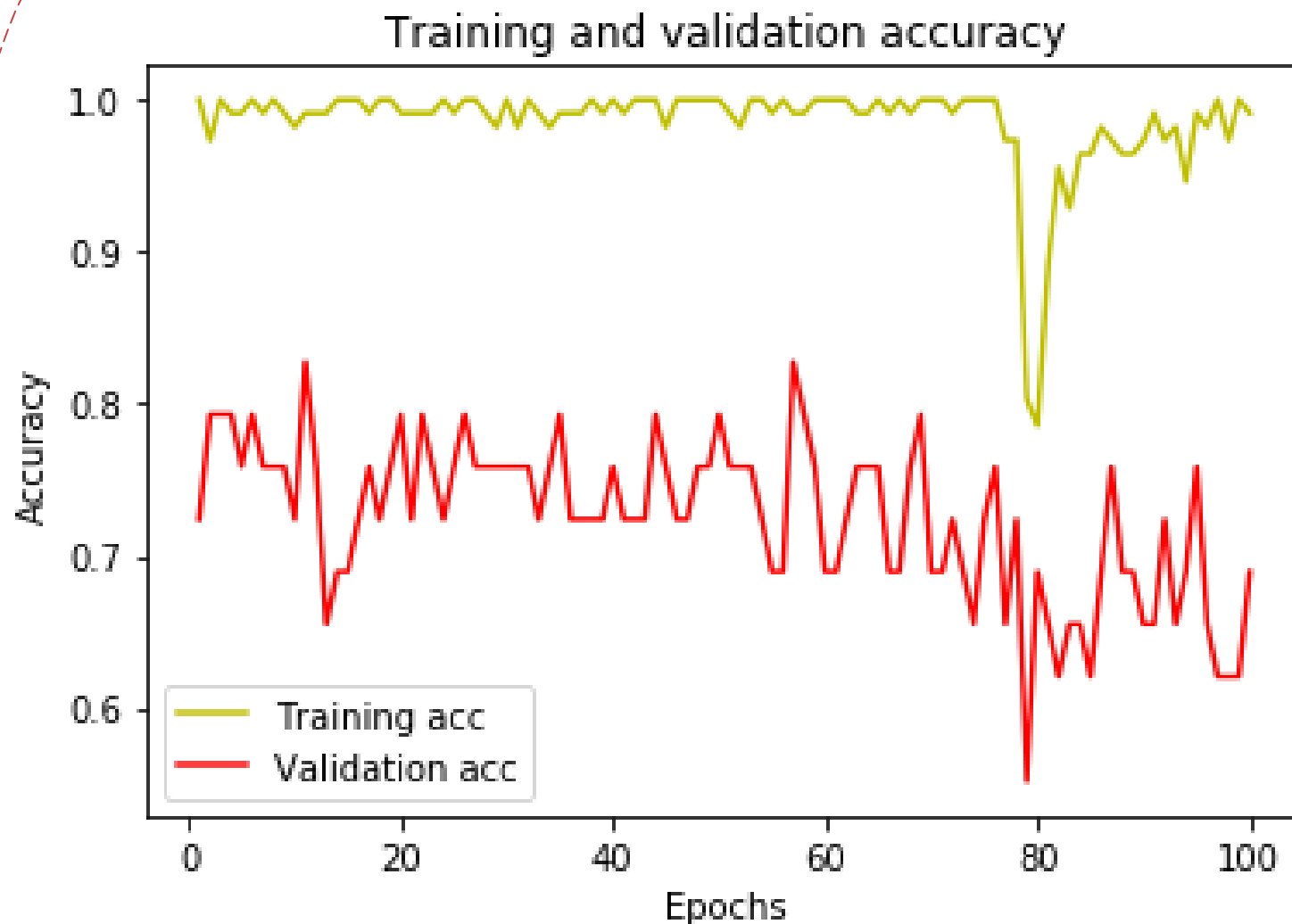
Deep Convolutional Neural Network (DCNN)

Création du modèle



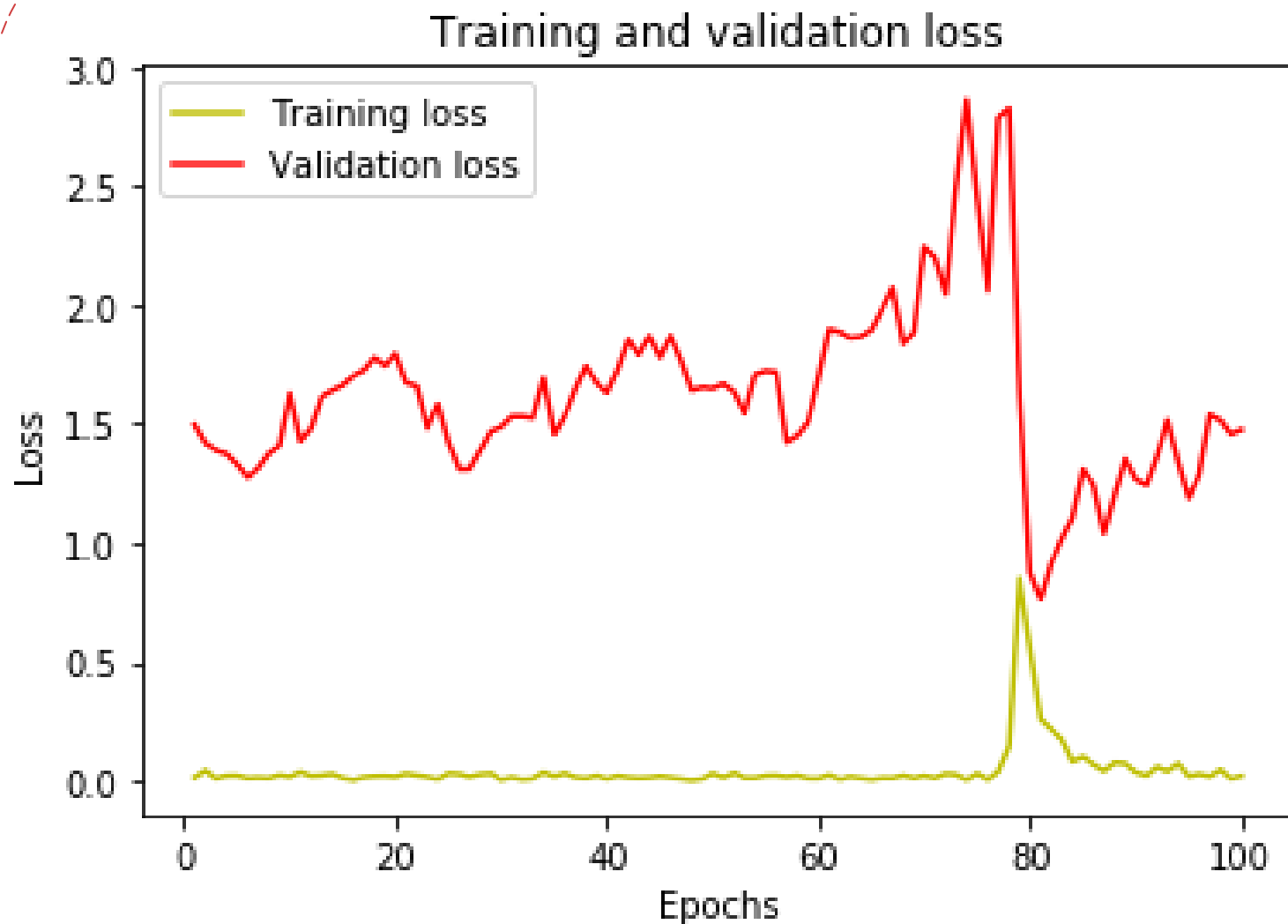
Deep Convolutional Neural Network (DCNN)

Essai#01



Deep Convolutional Neural Network (DCNN)

Essai#01



Deep Convolutional Neural Network (DCNN)

Essai#01

+ Précision générale = 68.96551847457886 %

+ Matrice de confusion: $\begin{pmatrix} 11 & 2 \\ 7 & 9 \end{pmatrix}$

Moyens d'optimisation de la précision des modèles intelligents

- Ajouter plus de données; avoir une base de données large.
- Élargir le réseau des neurones par l'ajout de couches de convolution supplémentaires.
- Minimiser les valeurs du Dropout (20% ~ 50%).
- Augmenter la valeur du taux d'apprentissage ($\text{learning_rate} \geq 0,9$).
- Pas de BatchNormalization(); empêche de traiter les données brutes.
- Cross Validation.

Optimisation#01

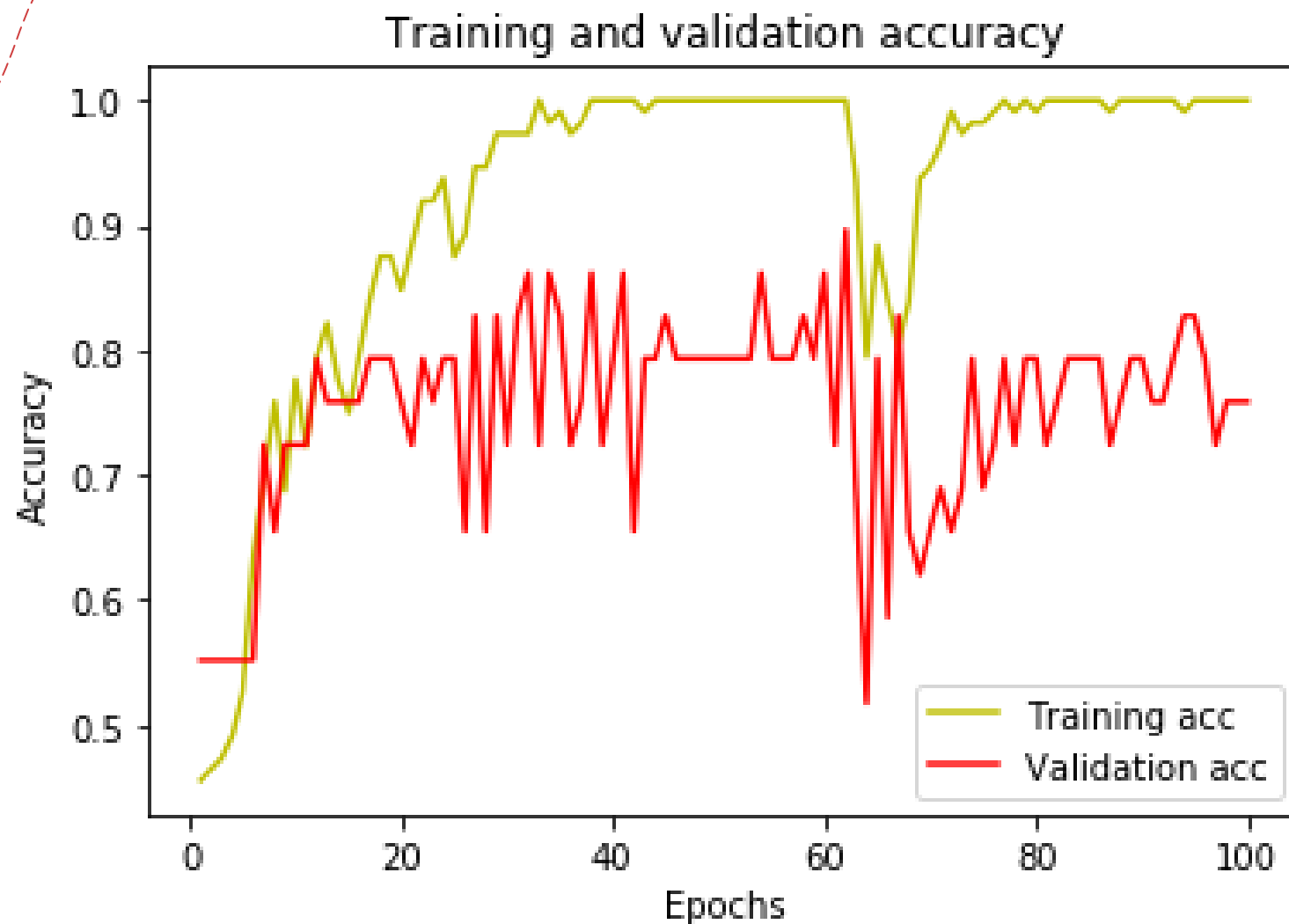
+ Valeur du Dropout=20%=0.2 donne:

➤ Précision générale = 75.86206793785095 %

➤ Matrice de confusion: $\begin{pmatrix} 11 & 2 \\ 6 & 10 \end{pmatrix}$

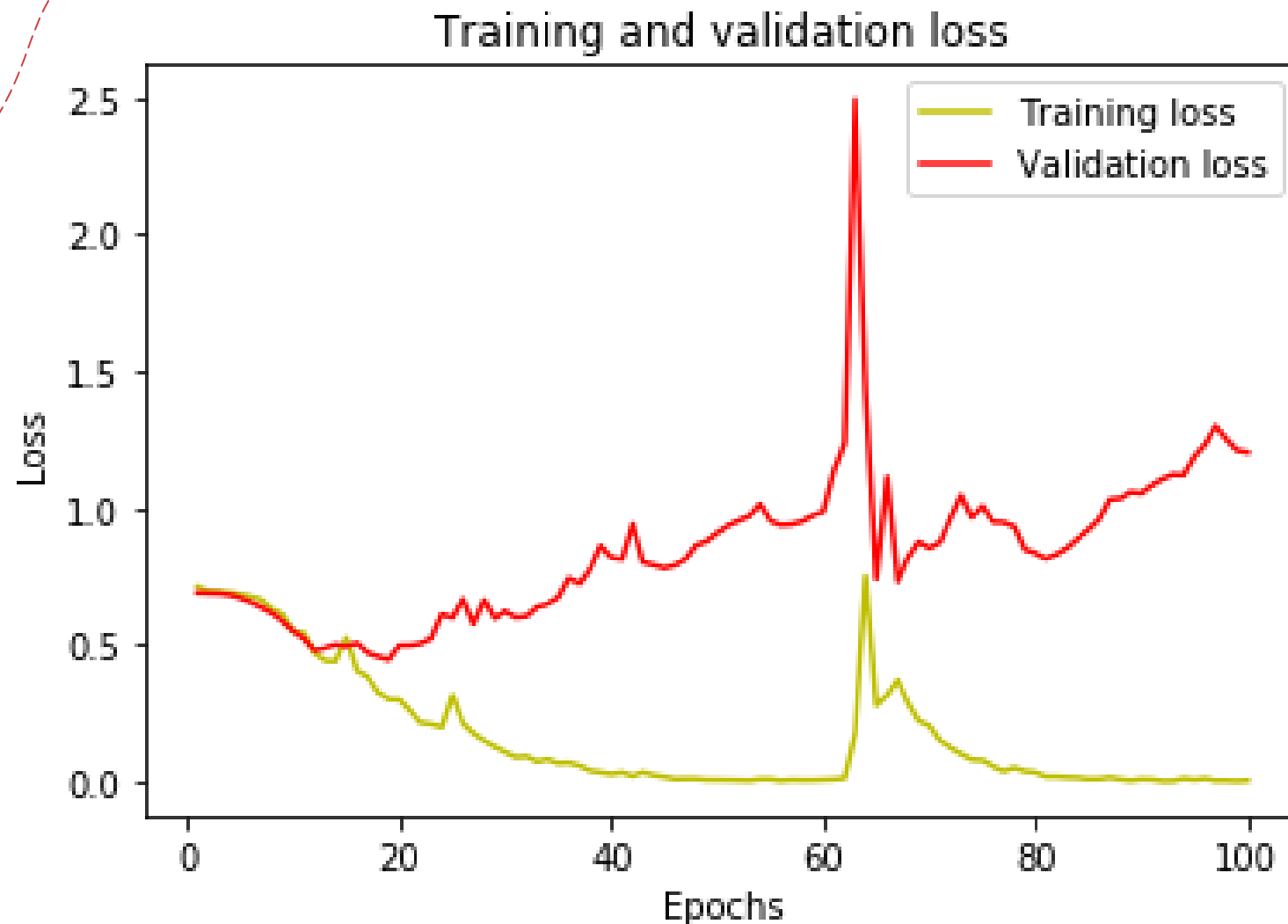
Deep Convolutional Neural Network (DCNN)

Essai#02



Deep Convolutional Neural Network (DCNN)

Essai#02

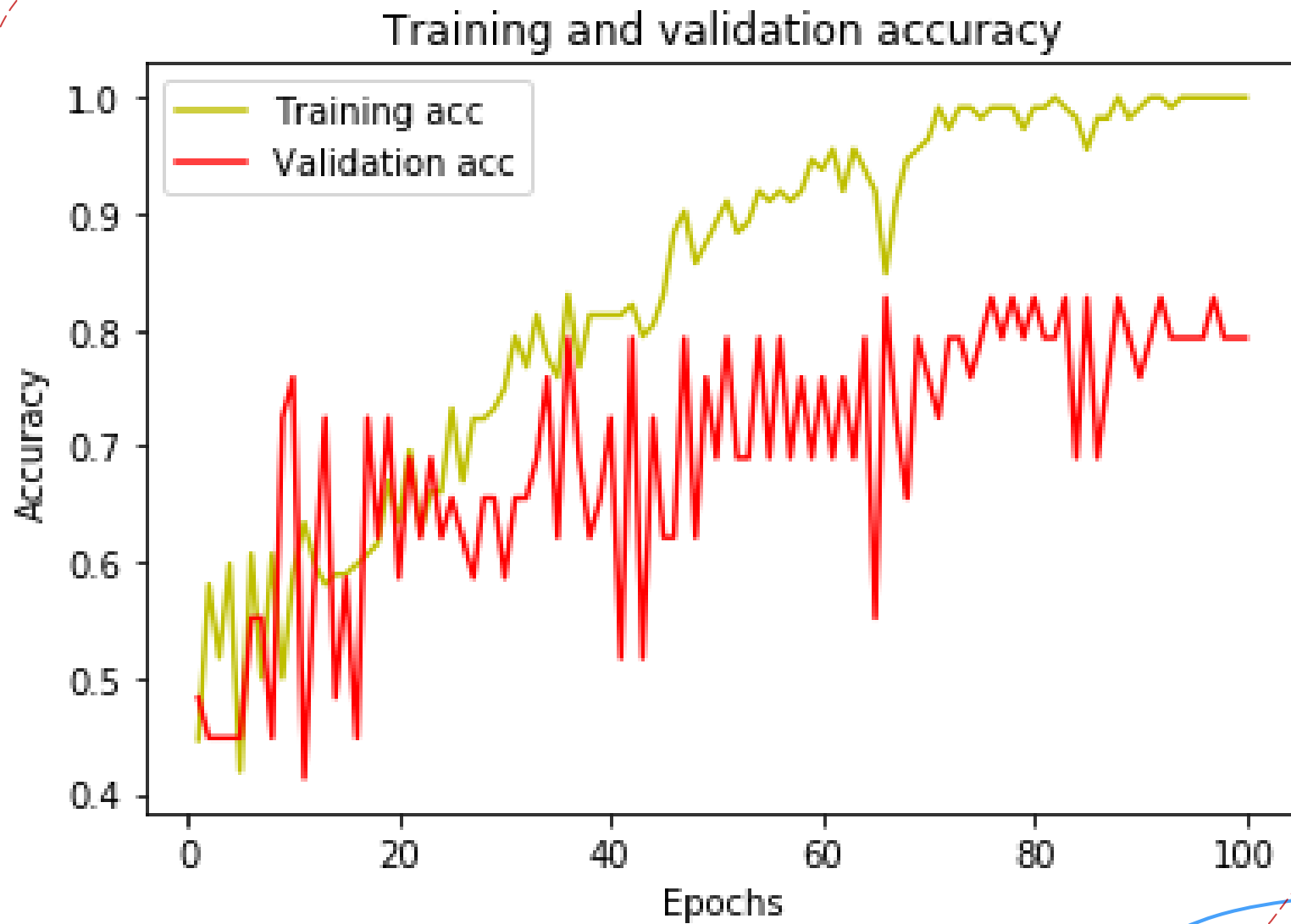


Optimisation#02

- + Valeur du Dropout=20%=0.2.
- + Ajout de deux autres couches de convolution et inversion d'ordre.
 - Précision générale = 79.31034564971924 %
 - Matrice de confusion: $\begin{pmatrix} 10 & 3 \\ 4 & 12 \end{pmatrix}$

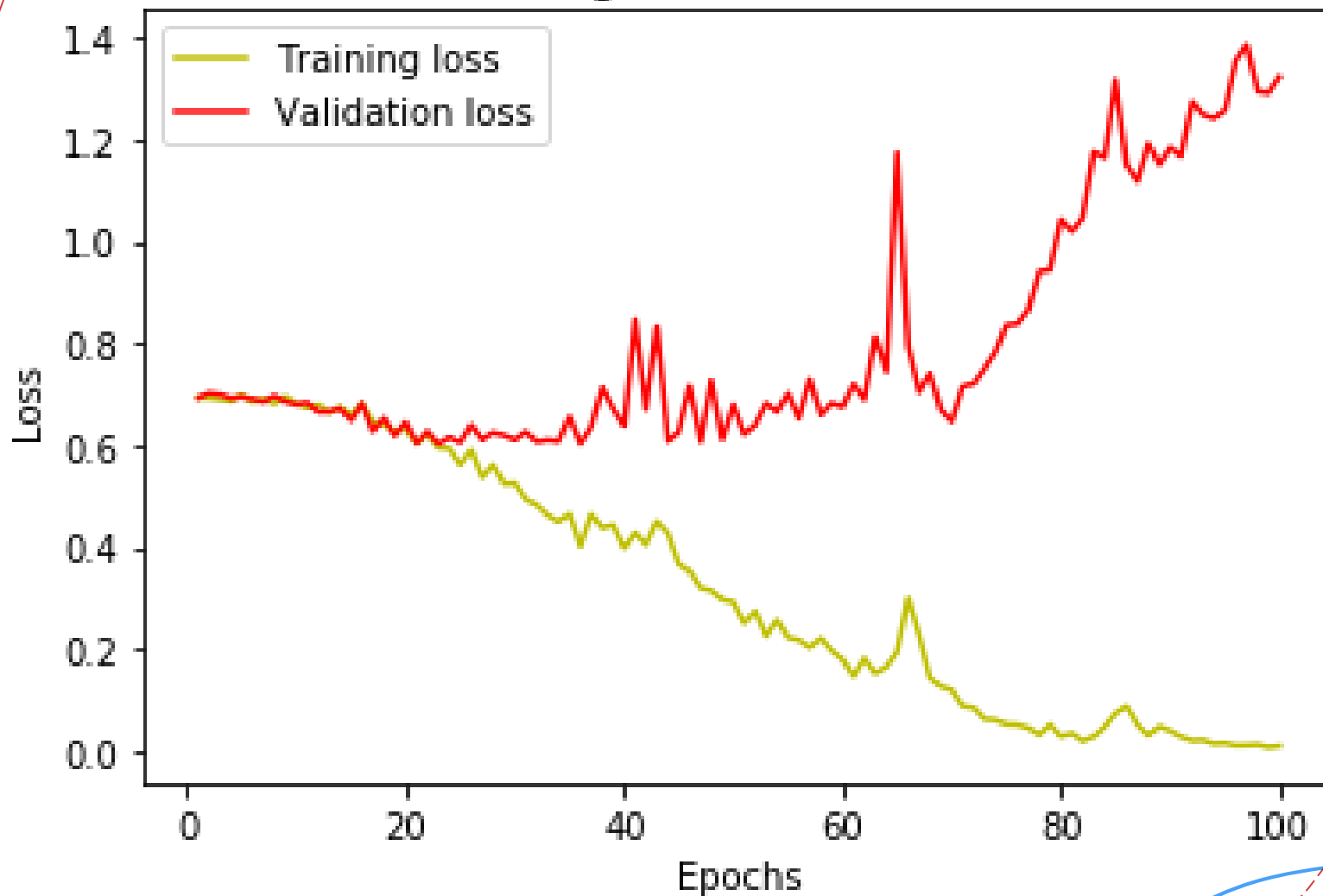
Deep Convolutional Neural Network (DCNN)

Essai#03



Deep Convolutional Neural Network (DCNN)

Training and validation loss



Essai#03



Comparaison et conclusion

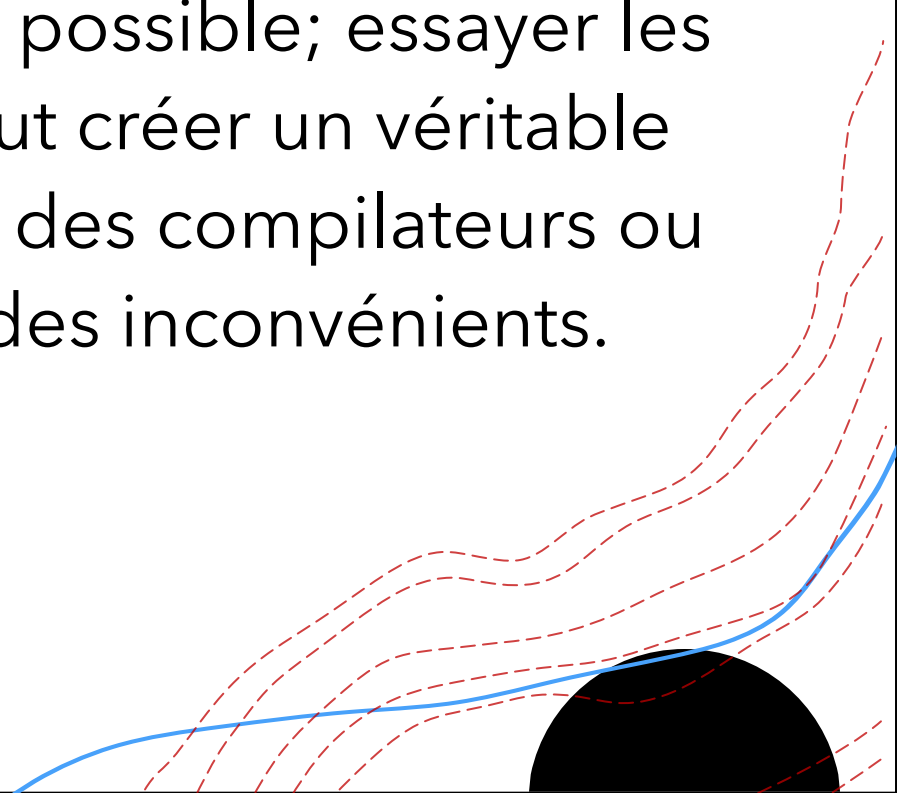
Remarques et comparaison

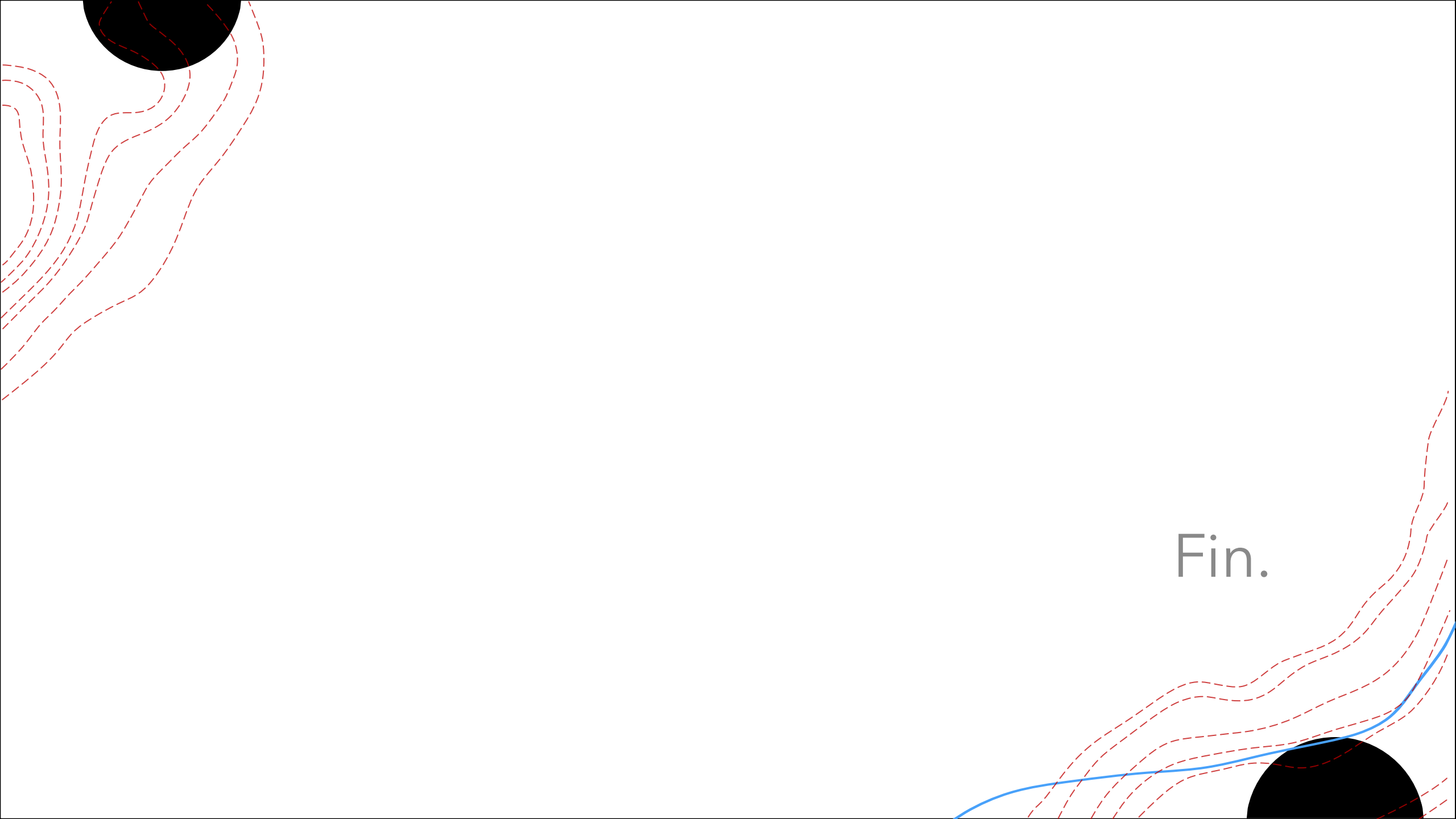
- La valeur du Dropout a immédiatement augmenté la précision (de 68.96% à 75.86%; différence de 6,9).
- L'ajout des couches de convolution et l'inversion de leur ordre a généré une précision de 79.31% (différence de 3,45) et notamment une amélioration dans la performance du modèle concernant la détection des cellules infectées.



Conclusion

L'amélioration de ce modèle reste toujours possible; essayer les autres méthodes d'optimisation citées peut créer un véritable changement. Pourtant, seule configuration des compilateurs ou ordinateurs utilisés, reste le plus grand des inconvénients.

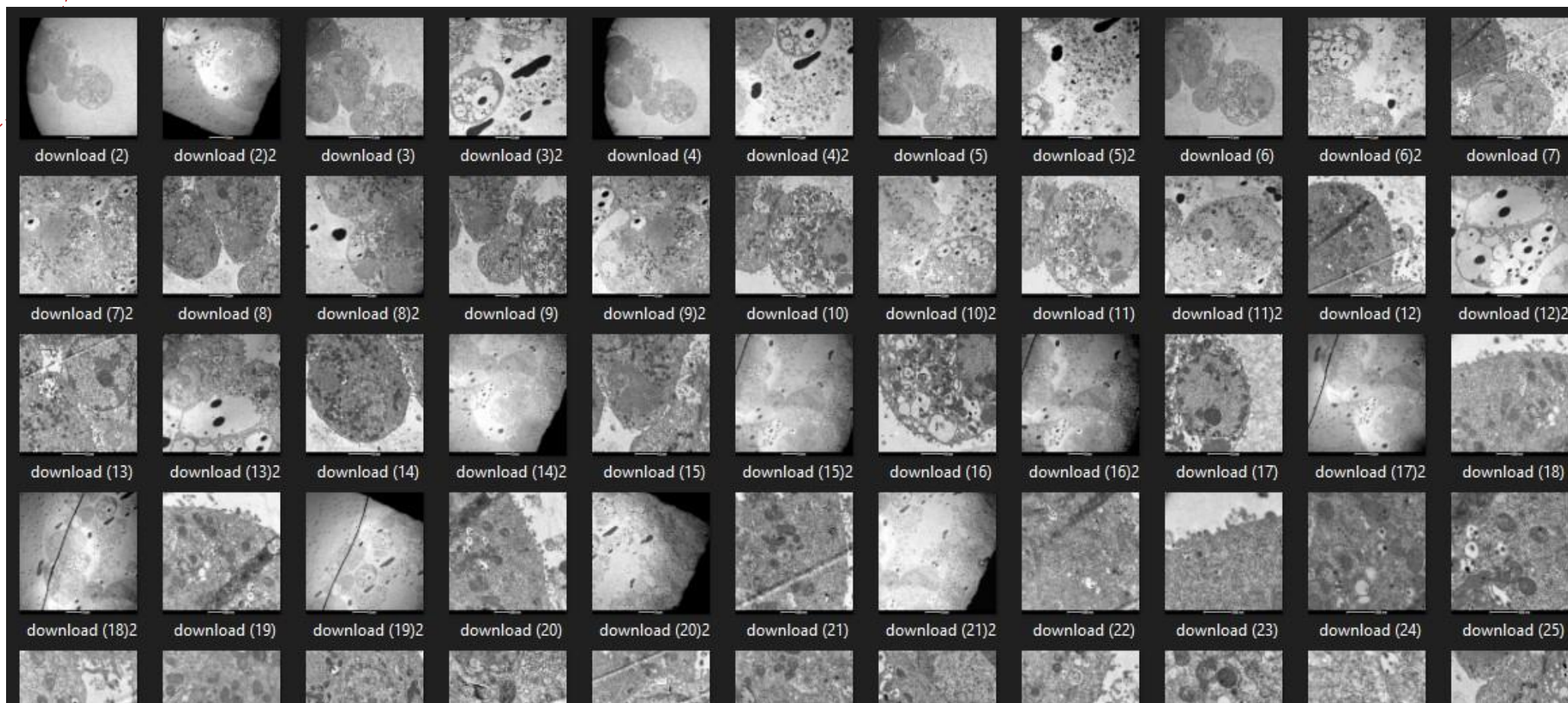




Fin.

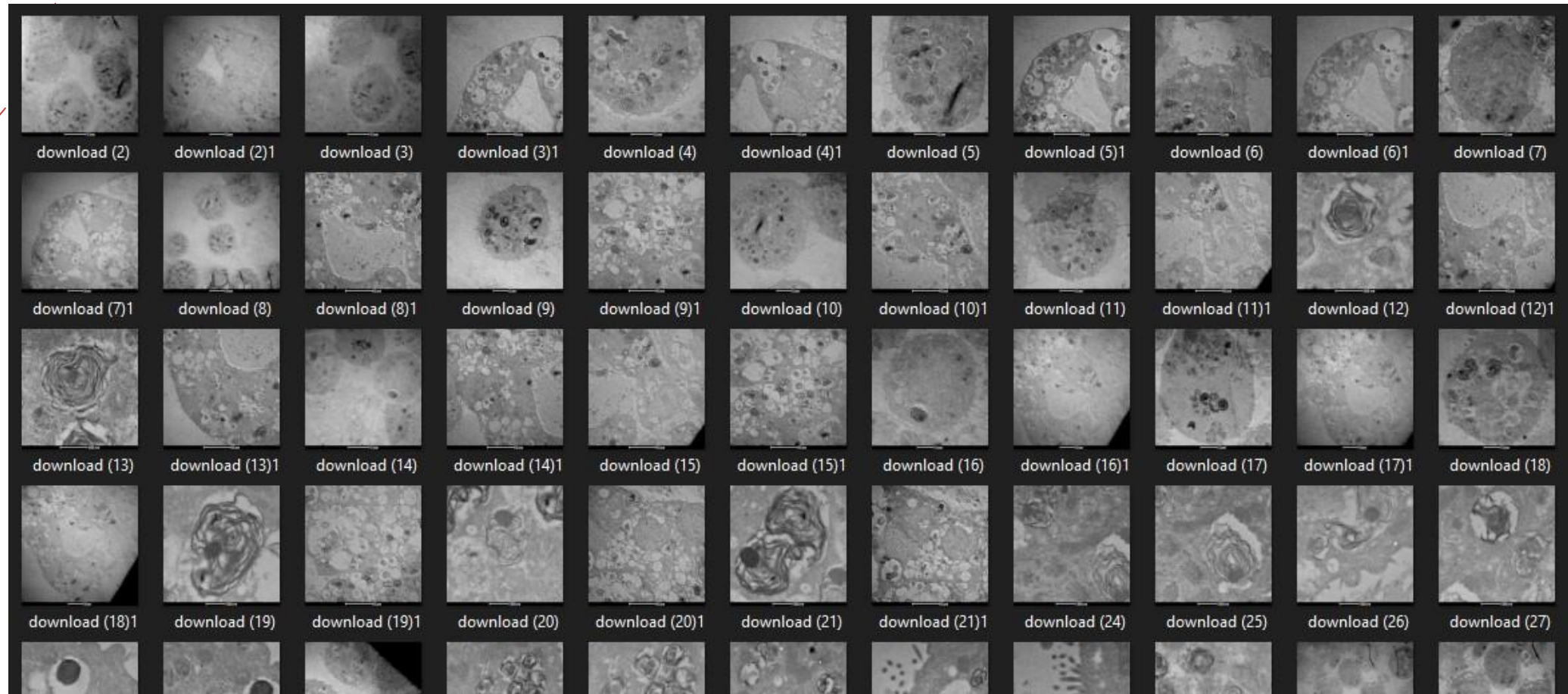
Présentation de la base de données

infected



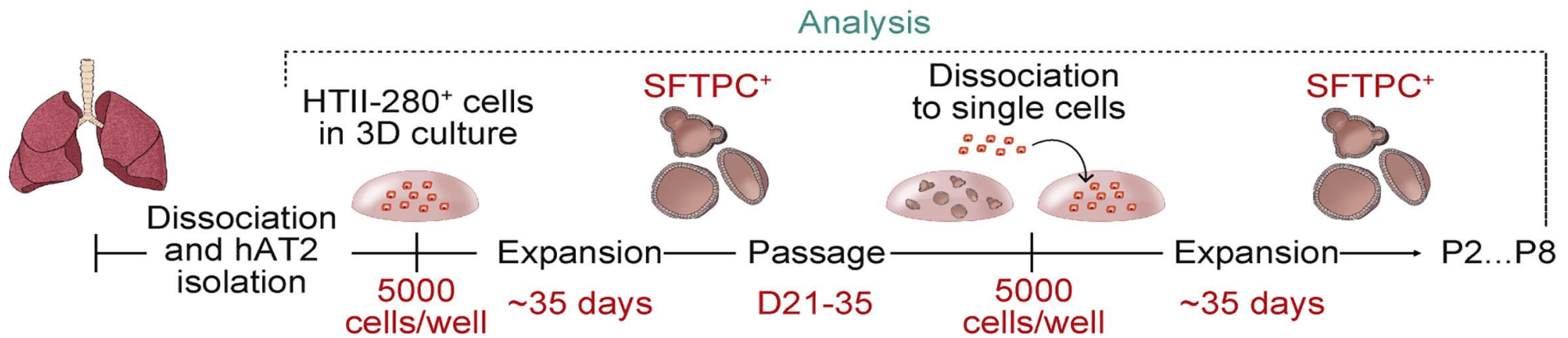
Présentation de la base de données

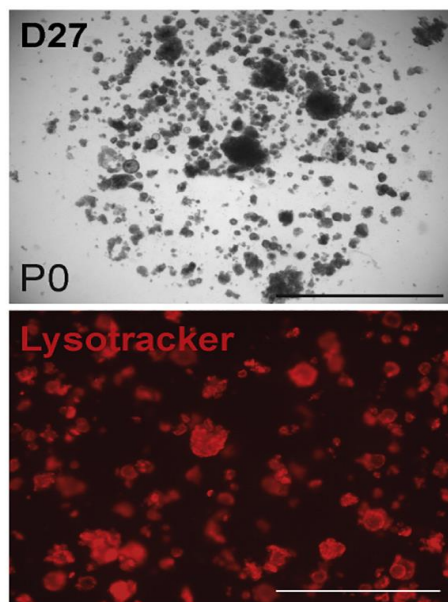
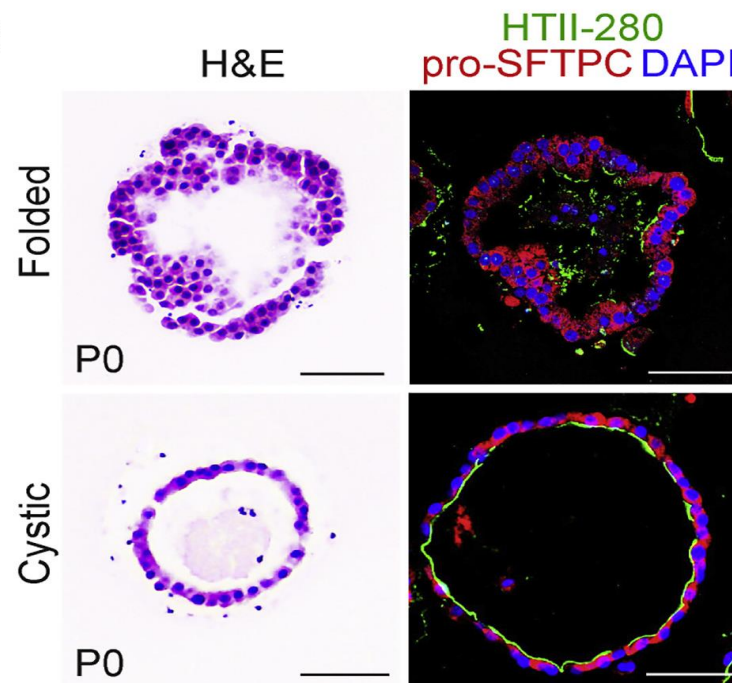
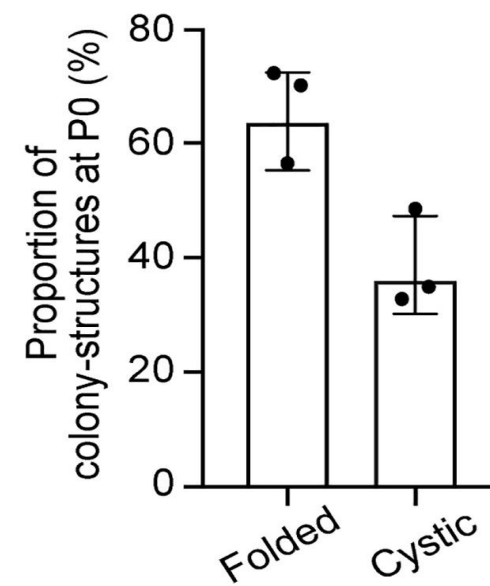
control



ANNEXE#01: Figures

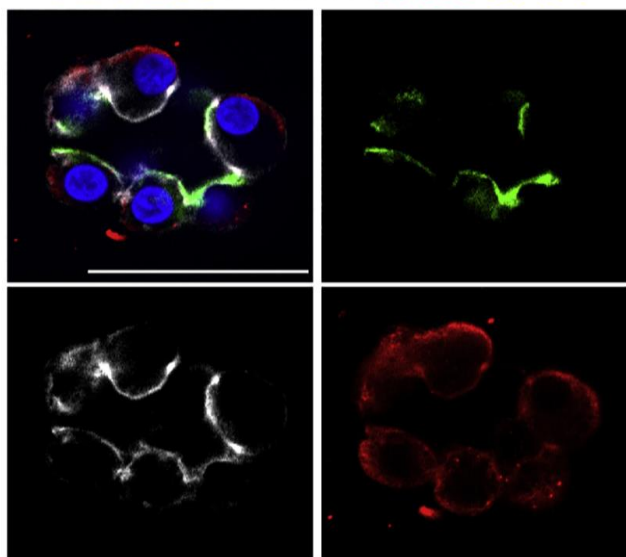
A



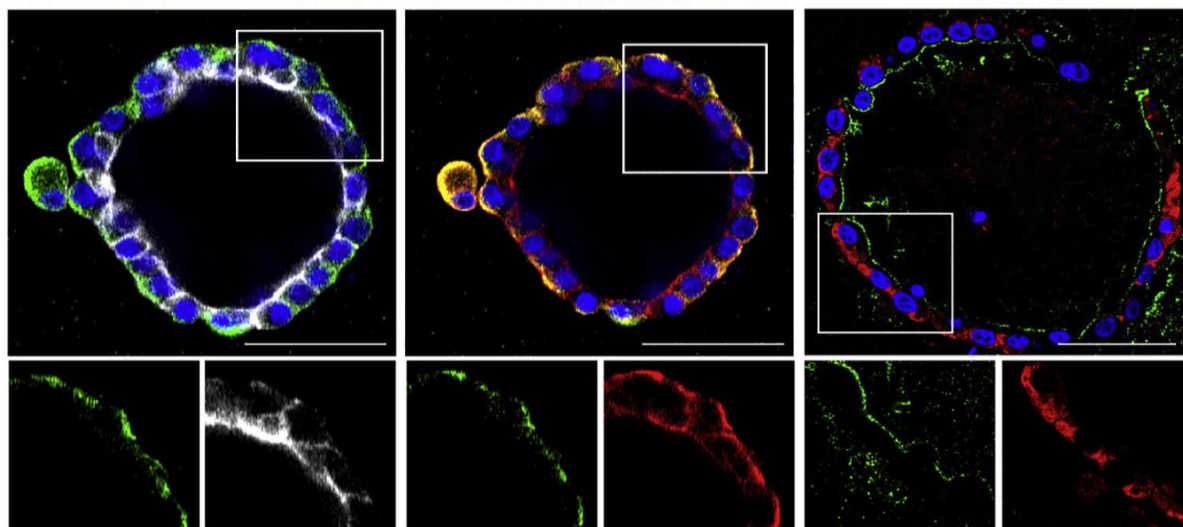
B**C****D**

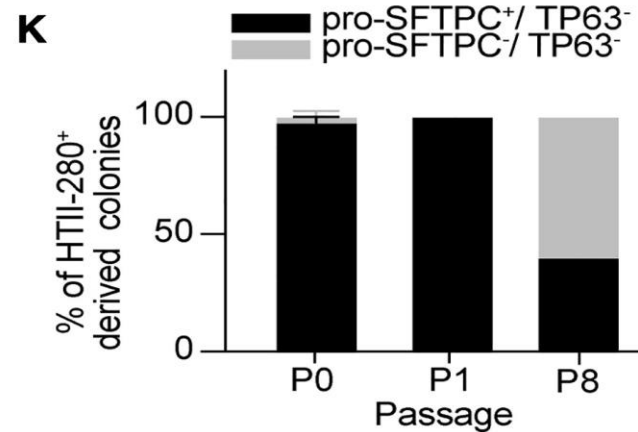
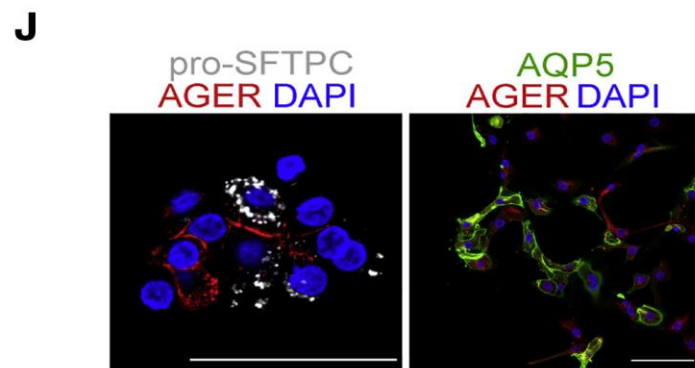
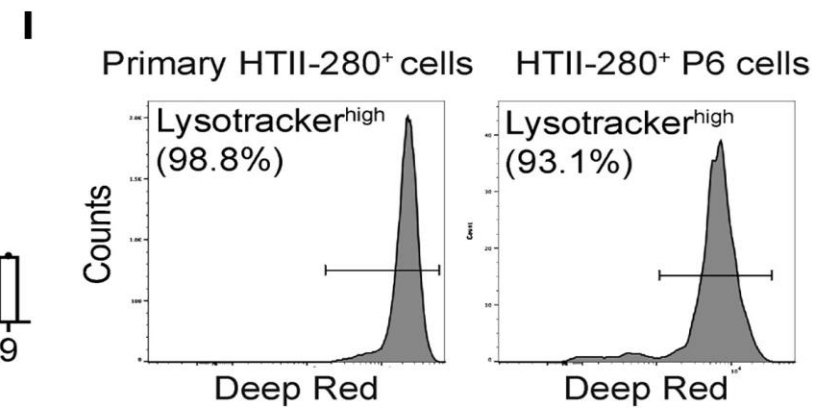
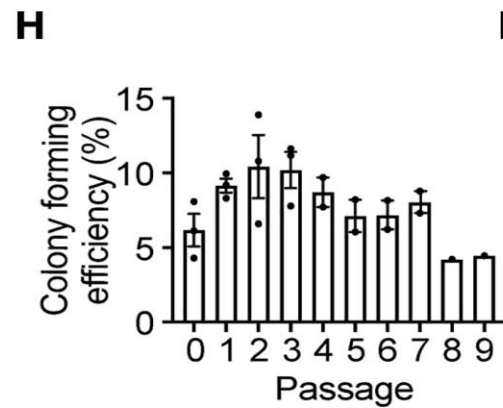
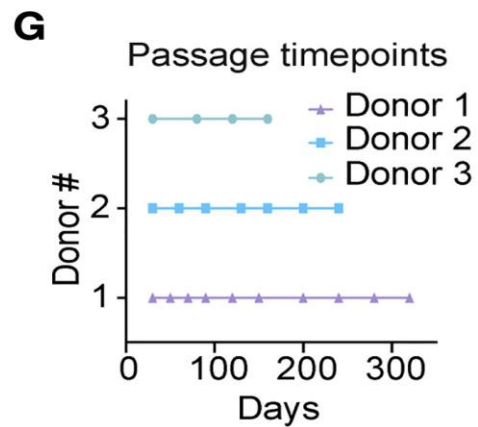
E

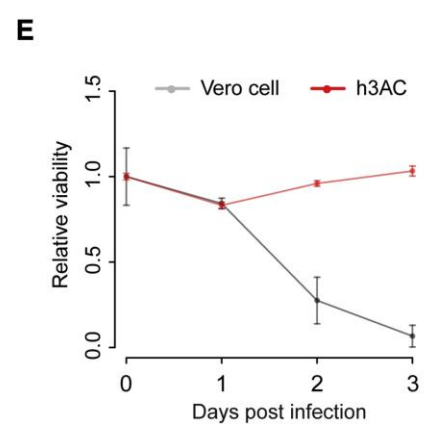
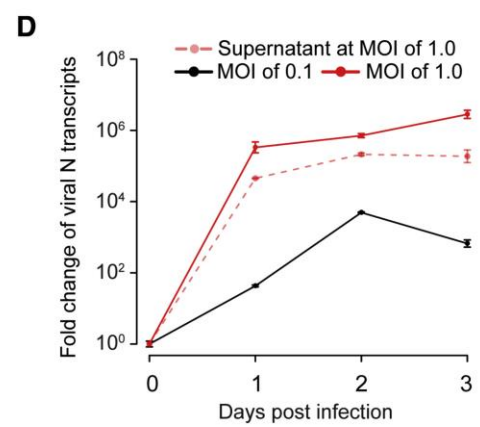
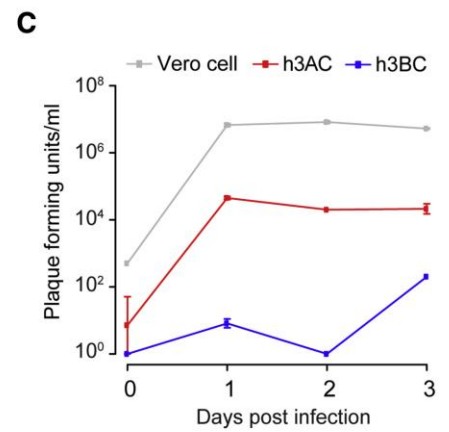
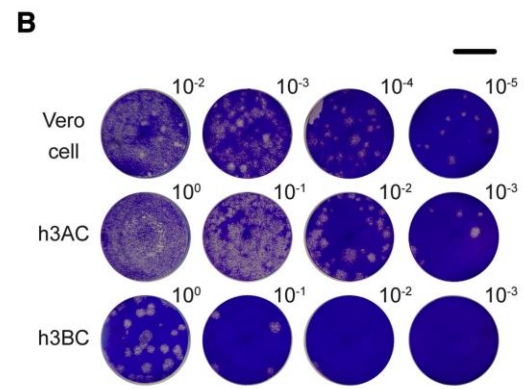
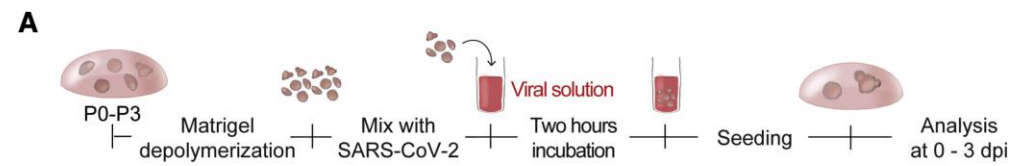
HTII-280 F-actin CRB3 DAPI

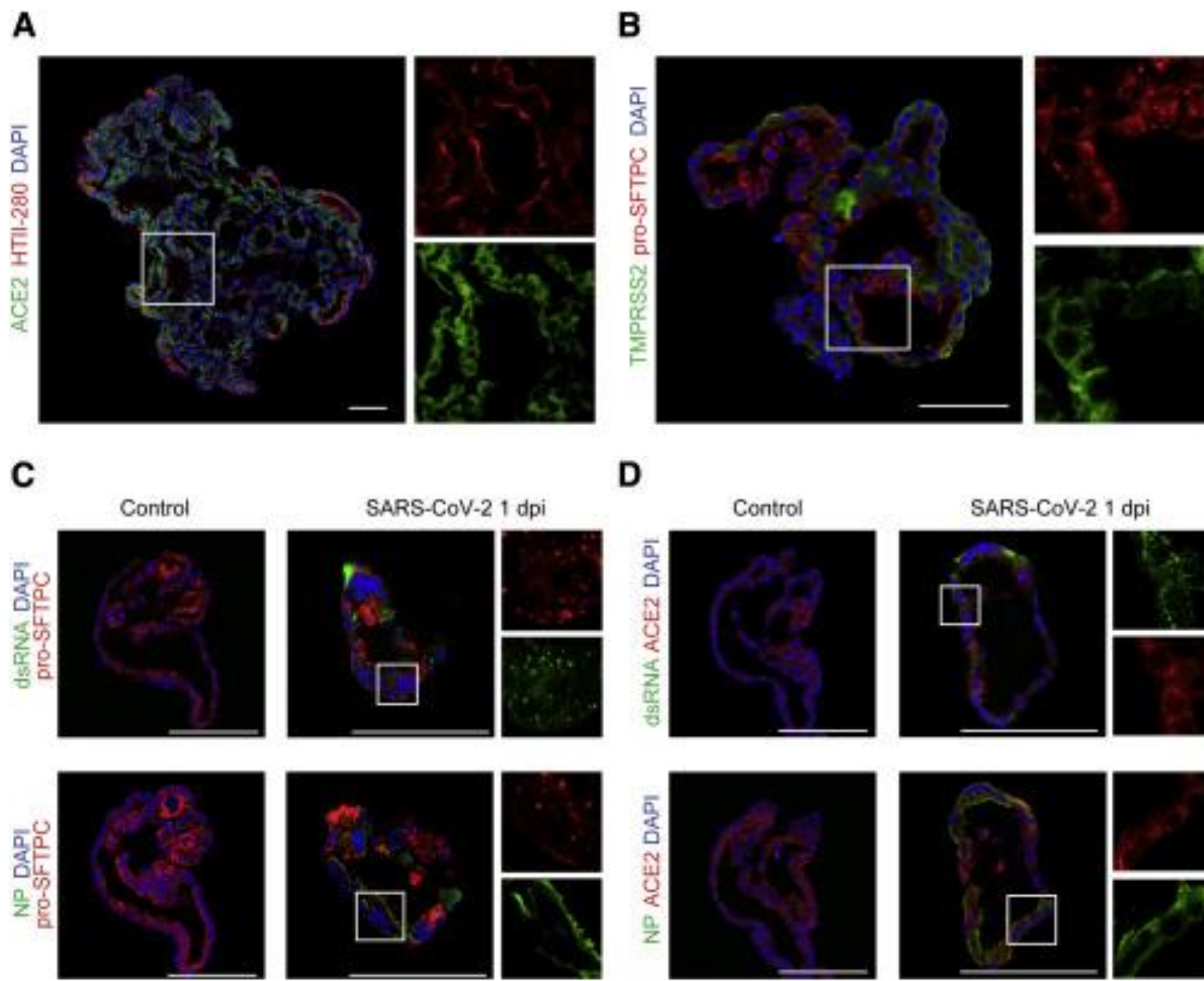
**F**

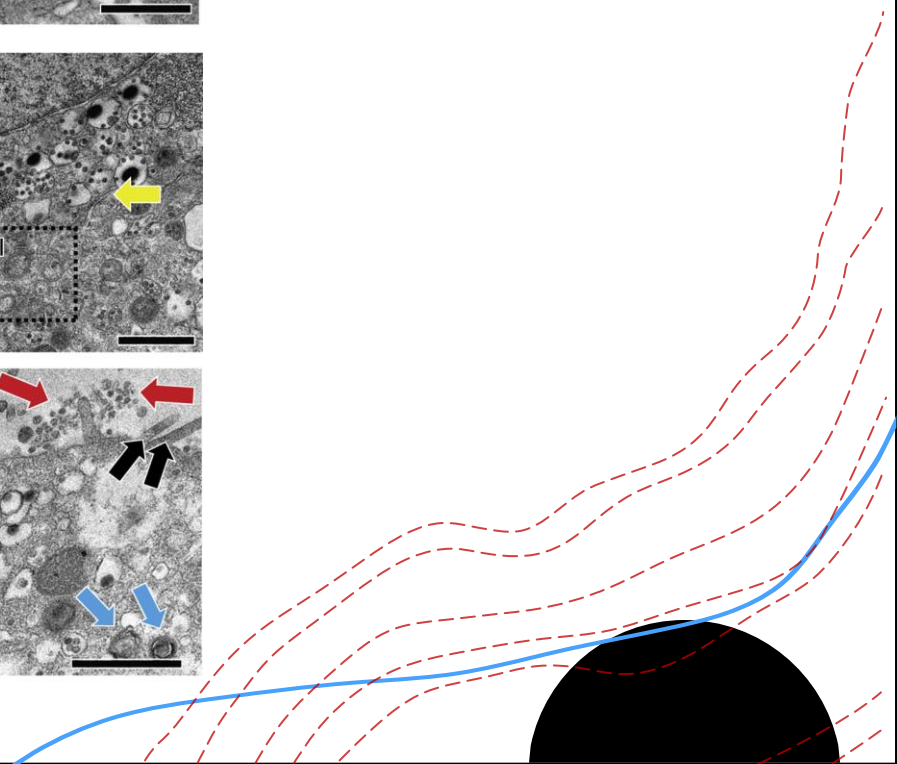
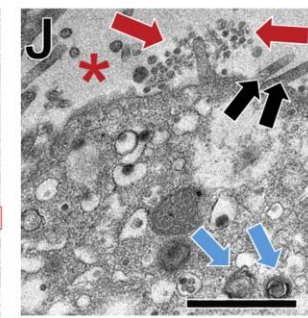
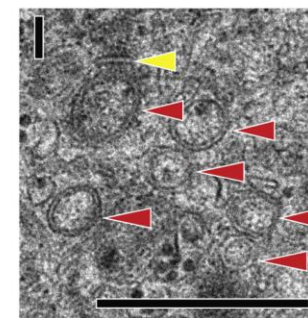
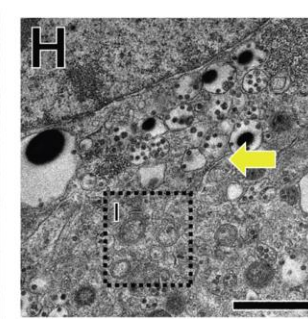
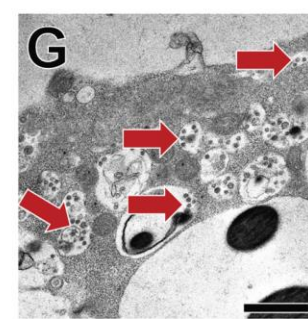
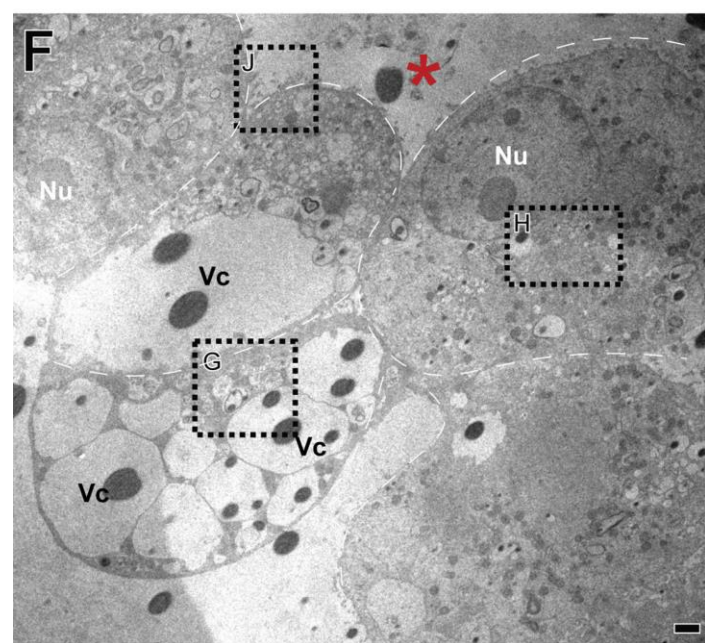
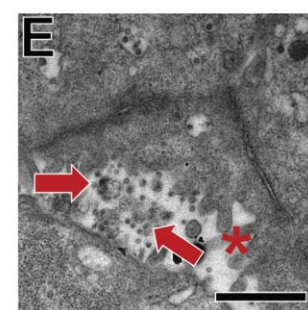
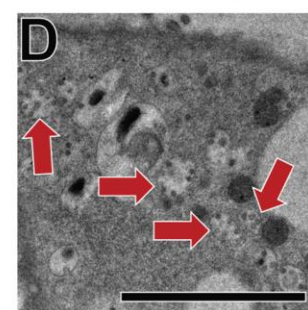
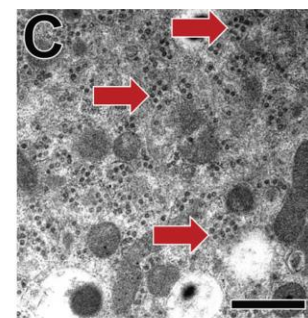
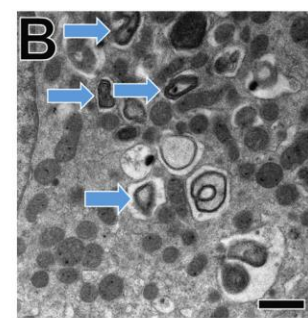
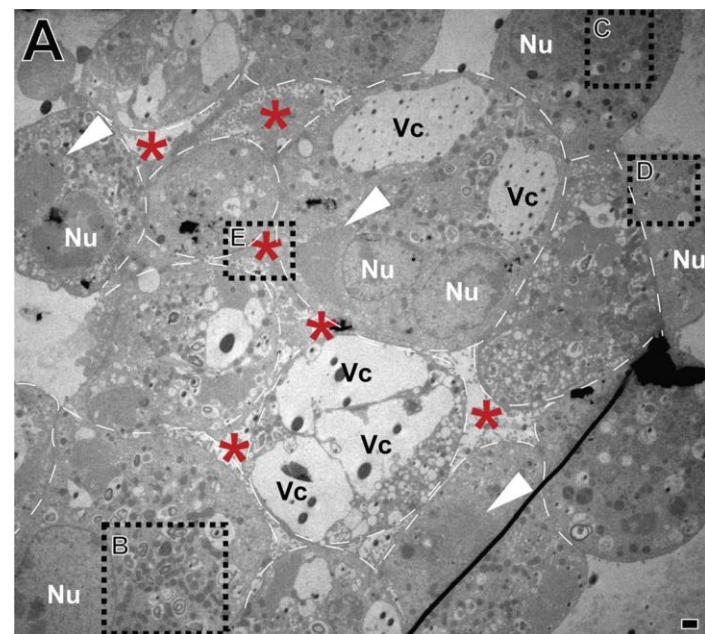
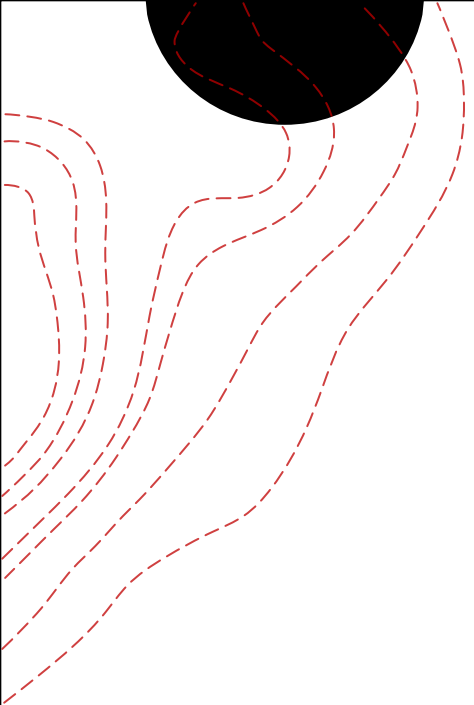
HTII-280 F-actin DAPI HTII-280 CRB3 DAPI HTII-280 SCRIB DAPI

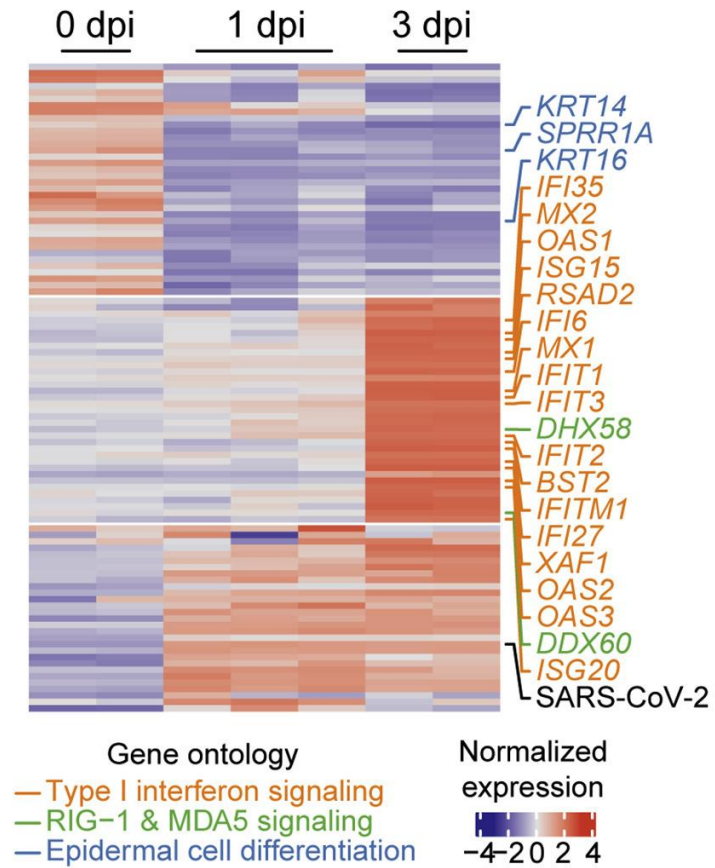
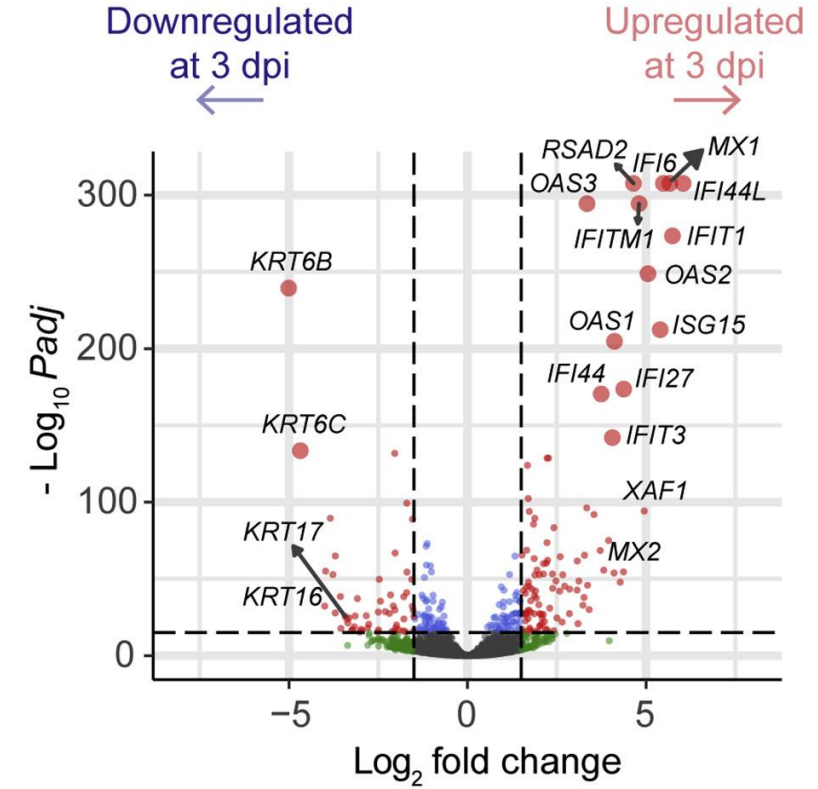


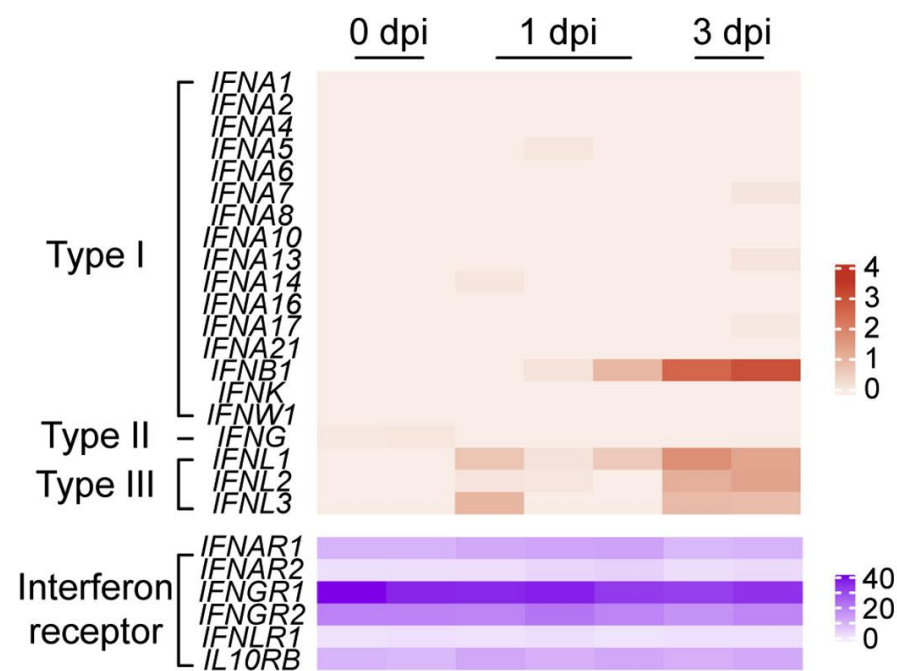
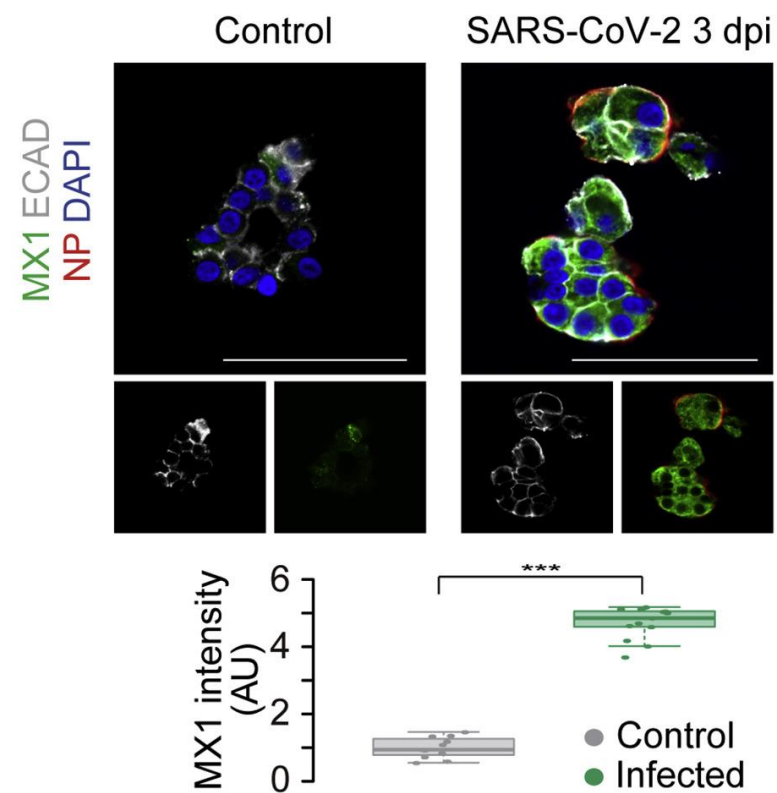








A**B**

C**D**

ANNEXE#02: Sobel & Laplacien

```
#SOBEL&&LAPLACIEN
img0 = cv2.imread(r"C:\Users\MonPc\Desktop\version 2.0\infected\download (3)2.jpg")
gray = cv2.cvtColor(img0, cv2.COLOR_BGR2GRAY) #grayscale
img = cv2.GaussianBlur(gray,(3,3),0) #denoising

# convolution kernels
laplacian = cv2.Laplacian(img,cv2.CV_64F)
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5) # x
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5) # y

plt.subplot(1,4,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(1,4,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(1,4,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(1,4,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()
```

✓ 1.8s

ANNEXE#03: Thresholding

```
img=cv2.imread(r"C:\Users\MonPc\Desktop\version 2.0\infected\download (3)2.jpg")
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret,th=cv2.threshold(gray,150,255,cv2.THRESH_BINARY)
plt.subplot(1,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(1,2,2), plt.imshow(th, cmap = 'gray')
plt.title('Threshold'), plt.xticks([]), plt.yticks([])
```

✓ 1.4s

ANNEXE#04: Filtre Gaussien, sigma=3.0

```
img = Image.open(r"C:\Users\MonPc\Desktop\version 2.0\infected\download (3)2.jpg")  
  
img_modified = scipy.ndimage.filters.gaussian_filter(img, sigma=3.0)  
  
plt.imshow(img_modified[:, :], cmap = plt.get_cmap('gray'), \  
           vmin=img_modified.min(), vmax=img_modified.max())  
plt.savefig("GaussianFilter1.png")
```

✓ 2.3s

ANNEXE#05: Filtre Gaussien, sigma=5.0

```
img = Image.open(r"C:\Users\MonPc\Desktop\version 2.0\infected\download (3)2.jpg")

img_modified = scipy.ndimage.filters.gaussian_filter(img, sigma=5.0)

plt.imshow(img_modified[:, :], cmap = plt.get_cmap('gray'), \
            vmin=img_modified.min(), vmax=img_modified.max())
plt.savefig("GaussianFilter2.png")
```

✓ 1.6s

ANNEXE#06: Filtre Gaussien, sigma=1.0

```
img = Image.open(r"C:\Users\MonPc\Desktop\version 2.0\infected\download (3)2.jpg")

img_modified = scipy.ndimage.filters.gaussian_filter(img, sigma=1.0)

plt.imshow(img_modified[:, :], cmap = plt.get_cmap('gray'), \
            vmin=img_modified.min(), vmax=img_modified.max())
plt.savefig("GaussianFilter3.png")
```

✓ 1.5s

ANNEXE#07: model_try_01.ipynb

```
#creatin za model
model = Sequential()

model.add(Conv2D(32, (3, 3), input_shape=(100, 100, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), kernel_initializer='he_uniform', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3), kernel_initializer='he_uniform', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(1))

model.add(Activation('sigmoid'))
```

✓ 0.8s

ANNEXE#09: model_try_02.ipynb

```
#creatin za model
model = Sequential()

model.add(Conv2D(32, (3, 3), input_shape=(100, 100, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), kernel_initializer='he_uniform', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3), kernel_initializer='he_uniform', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(1))

model.add(Activation('sigmoid'))
```

✓ 0.2s

ANNEXE#07: model_try_03.ipynb

```
#creatin za model
model = Sequential()

model.add(Conv2D(128, (3, 3), input_shape=(100, 100, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), kernel_initializer='he_uniform', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), kernel_initializer='he_uniform', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3), kernel_initializer='he_uniform', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

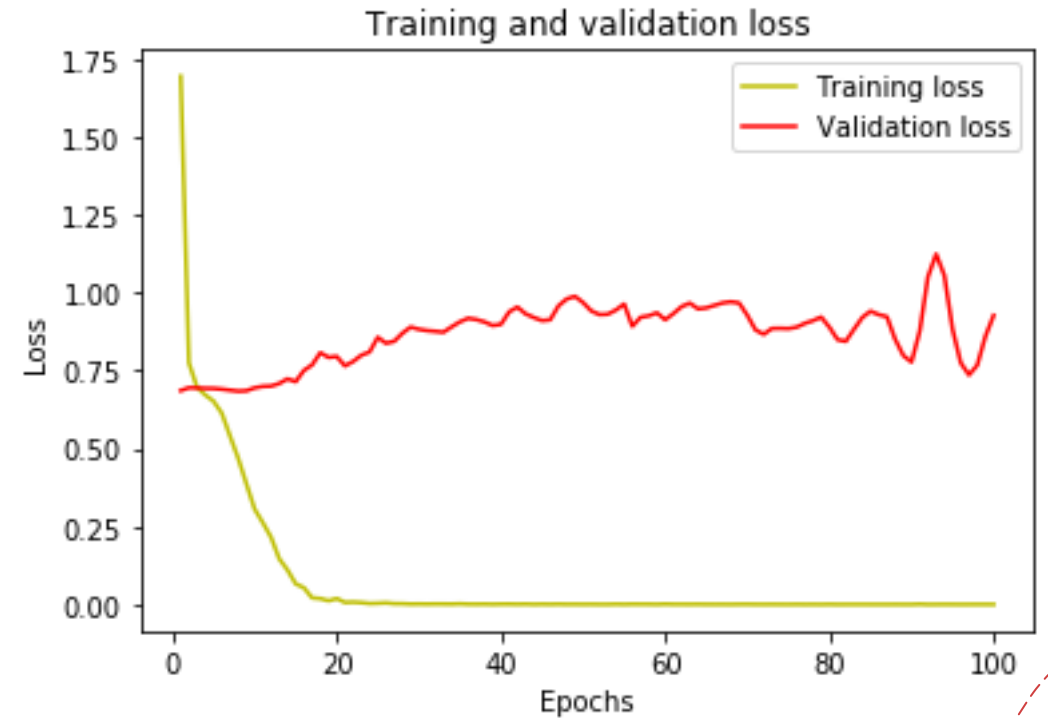
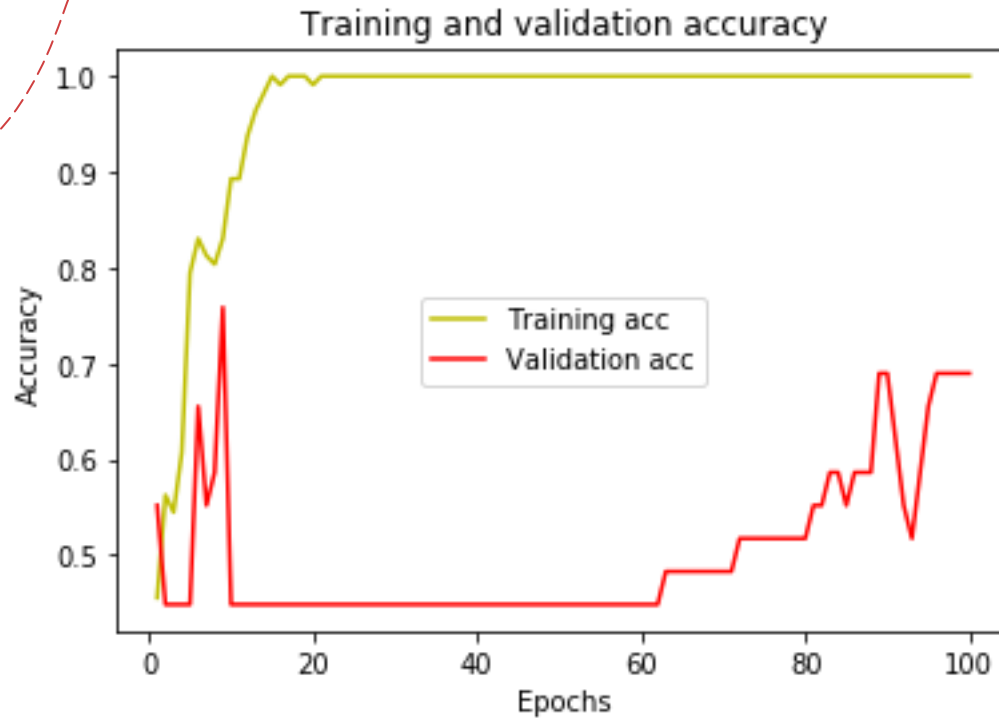
model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(1))

model.add(Activation('sigmoid'))
```

✓ 0.7s

ANNEXE#10: ajout de BatchNormalization()



+ Overfitting du modèle, de plus la précision reste la même.