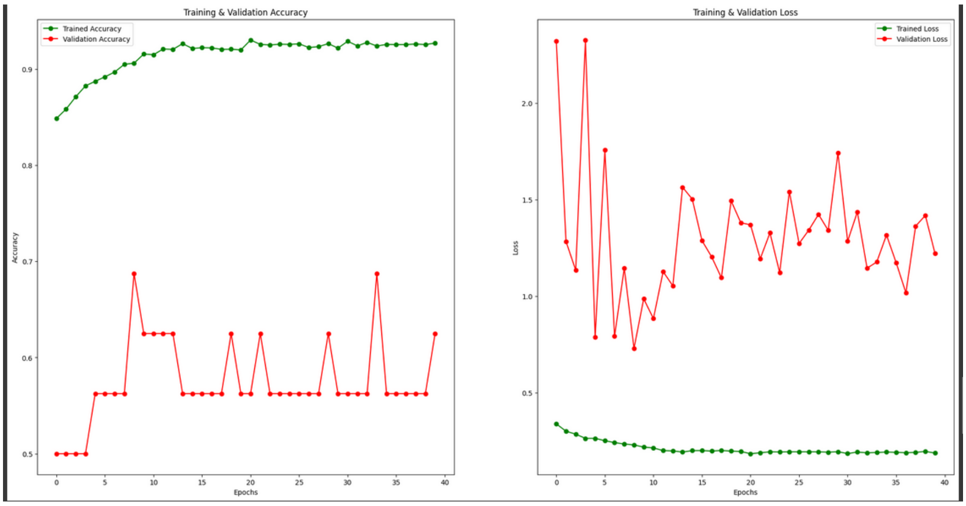
Médicale Image Recognition

Tested Model and their predict

Activation sigmoid for all layers

	Model = Sequential															
id	Couche 1	Couche 2	Couche 3	Couche 4	Couche 5	Couche 6	Couche 7	Couche 8	Couche 9	Couche 10	Couche 11	Couche 12	Couche 13	Couche 14	Couche 15	Couche 16
Base	Conv2D, 32,3 , Activation=sigmoid Input_shape(224, 224, 1)	MaxPool2D(2, 2)	BatchNormalization	Conv2D , 64, 3 Activation=sigmoid	MaxPool2D(2 , 2)	BatchNormalization	Conv2D , 128 ,3, Activation=sigmoid	MaxPool2D(2 , 2)	BatchNormalization	Conv2D , 256 ,3, Activation=sigmoid	MaxPool2D(2 , 2)	BatchNormalization	Flaten	Dense, 128 , activation=sigmoid	Dropout, 0.5	Dense, 1,activation=sigmoid

Compilation			Fit			Data			
Optimizer	Loss	metrics	Epoch	Final loss	Final accuracy	Batch_size	Height	Width	Class_mode
Adam	Binary_crossentropy	accuracy	40	0.2889464199542999	0.8782051205635071	32	224	224	binary



Prédiction :

```
                       precision    recall  f1-score   support

Pneumonia (Classe: 0)       0.93      0.87      0.90       390
   Normal (Classe: 1)       0.81      0.89      0.85       234

             accuracy                           0.88       624
            macro avg       0.87      0.88      0.87       624
         weighted avg       0.88      0.88      0.88       624
```

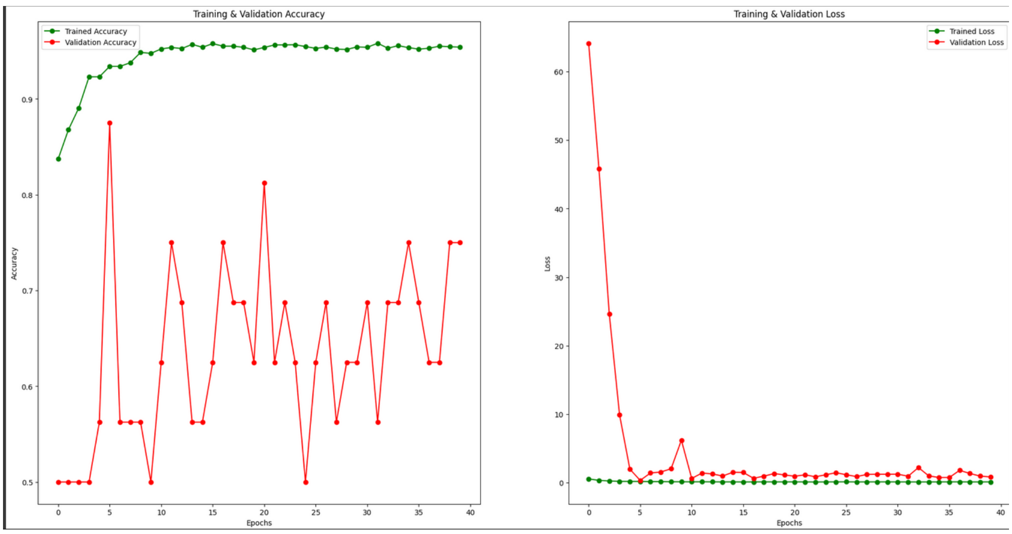
	0	1
0	340	50
1	26	208

Activation sigmoid for layers 16

	Model = Sequential															
id	Couche 1	Couche 2	Couche 3	Couche 4	Couche 5	Couche 6	Couche 7	Couche 8	Couche 9	Couche 10	Couche 11	Couche 12	Couche 13	Couche 14	Couche 15	Couche 16
Base	Conv2D, 32, 3, Activation=relu Input_	MaxPool2D(2, 2)	Batch Normalization	Conv2D, 64, 3 Activation=relu	MaxPool2D (2, 2)	Batch Normalization	Conv2D, 128,3 , Activation=relu	MaxPool2D (2, 2)	Batch Normalization	Conv2D, 256,3 , Activation=relu	MaxPool2D (2, 2)	Batch Normalization	Flatten	Dense , 128, activation=relu	Dropout , 0.5	Dense , 1,activation=sigmoid

	shape(224,224,1)															

Compilation			Fit			Data			
Optimizer	Loss	metrics	Epoch	Final loss	Final accuracy	Batch_size	Height	Width	Class_mode
Adam	Binary_crossentropy	accuracy	40	0.24228760600090027	0.91346156597137745	32	224	224	binary



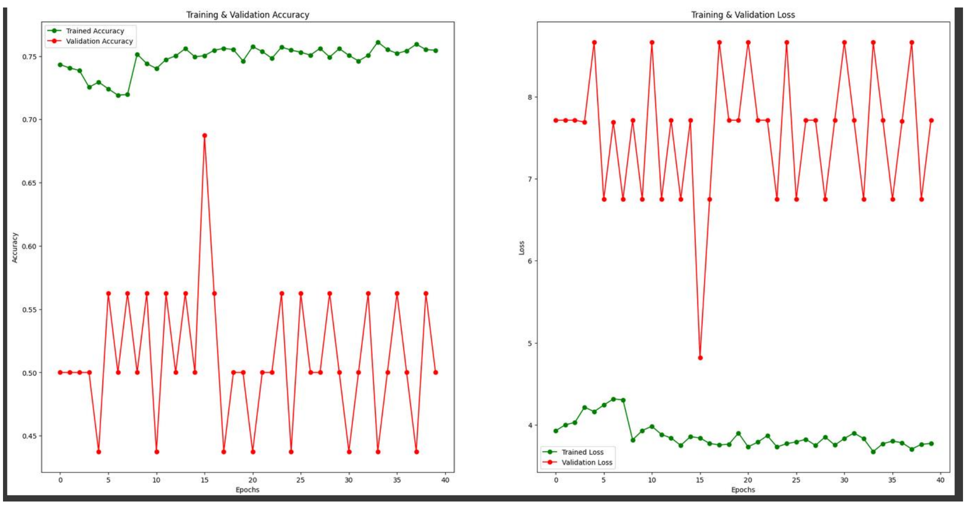
Prédiction

	0	1
0	375	15
1	39	195

Activation relu for all layers :

	Model = Sequential															
id	Couche 1	Couche 2	Couche 3	Couche 4	Couche 5	Couche 6	Couche 7	Couche 8	Couche 9	Couche 10	Couche 11	Couche 12	Couche 13	Couche 14	Couche 15	Couche 16
Base	Conv2D, 32,3, Activation =relu Input_shape(224, 224, 1)	MaxPool 2D (2, 2)	BatchNormalization	Conv2D, 64, 3 Activation=relu	MaxPool 2D (2, 2)	BatchNormalization	Conv2D, 128,3 , Activation=relu	MaxPool 2D (2, 2)	BatchNormalization	Conv2D, 256,3 , Activation=relu	MaxPool 2D (2, 2)	BatchNormalization	Flatten	Dense , 128, activation=relu	Dropout , 0.5	Dense , 1,activation=relu

Compilation			Fit			Data			
Optimizer	Loss	metrics	Epoch	Final loss	Final accuracy	Batch_size	Height	Width	Class_mode
Adam	Binary_crossentropy	accuracy	40	5.75963592529296 9	0.626602590084075 9	32	224	224	binary



Prédiction

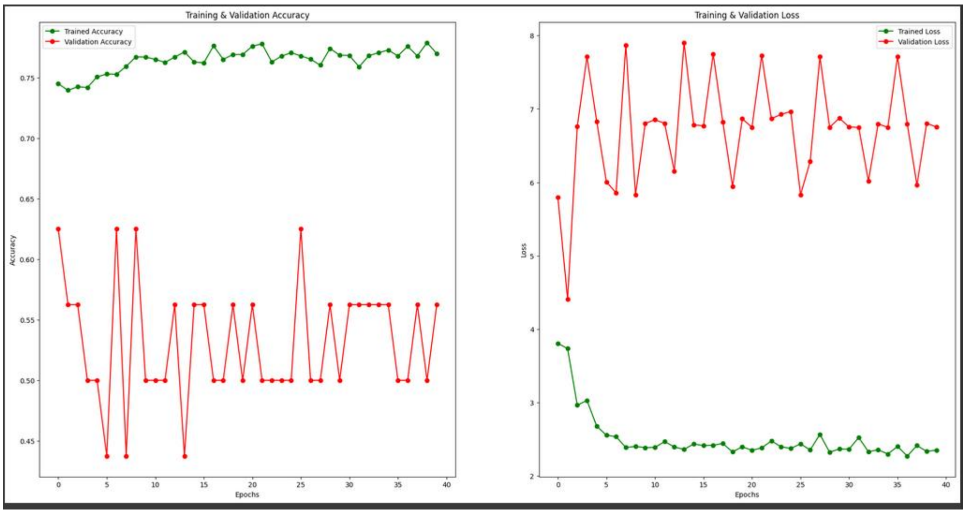
```
                          precision    recall  f1-score   support

 Pneumonia (Classe: 0)       0.63      1.00      0.77       390
    Normal (Classe: 1)       1.00      0.00      0.01       234

              accuracy                           0.63       624
             macro avg       0.81      0.50      0.39       624
          weighted avg       0.77      0.63      0.48       624
```

	0	1
0	390	0
1	233	1

Activation relu for layers 16:

	Model = Sequential															
id	Couche 1	Couche 2	Couche 3	Couche 4	Couche 5	Couche 6	Couche 7	Couche 8	Couche 9	Couche 10	Couche 11	Couche 12	Couche 13	Couche 14	Couche 15	Couche 16
Base	Conv2D, 32,3 , Activation=sigmoid Input_shape(224, 224, 1)	MaxPool2D(2, 2)	BatchNormalization	Conv2D , 64, 3 Activation=sigmoid	MaxPool2D (2 , 2)	BatchNormalization	Conv2D , 128 ,3, Activation=sigmoid	MaxPool2D (2 , 2)	BatchNormalization	Conv2D , 256 ,3, Activation=sigmoid	MaxPool2D (2 , 2)	BatchNormalization	Flatten	Dense, 128 , activation=sigmoid	Dropout, 0.5	Dense , 1,activation =relu

Compilation			Fit			Data			
Optimizer	Loss	metrics	Epoch	Final loss	Final accuracy	Batch_size	Height	Width	Class_mode
Adam	Binary_crossentropy	accuracy	40	1.3189243078231812	0.8333333134651184	32	224	224	binary



Prdiction

	precision	recall	f1-score	support
Pneumonia (Classe: 0)	0.82	0.94	0.88	390
Normal (Classe: 1)	0.86	0.66	0.75	234
accuracy			0.83	624
macro avg	0.84	0.80	0.81	624
weighted avg	0.84	0.83	0.83	624

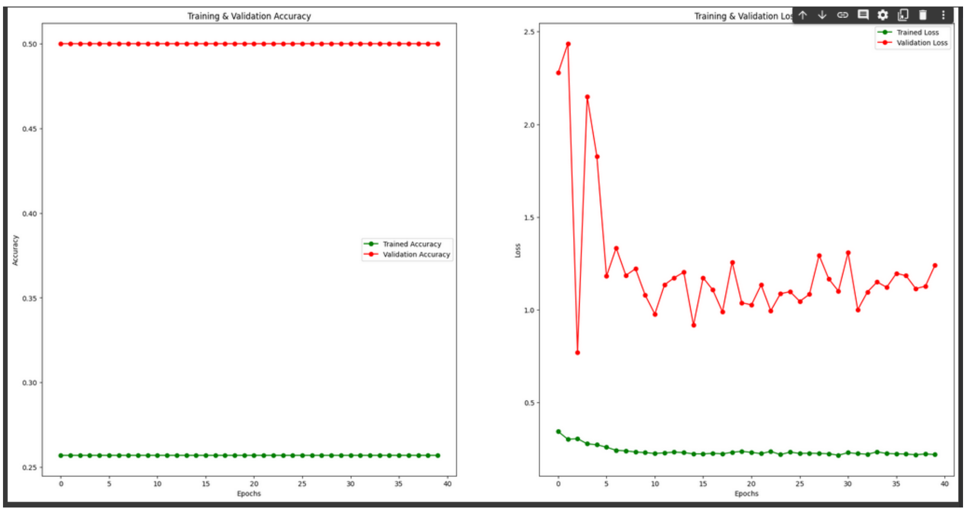
	0	1
0	365	25
1	79	155

Activation softmax for layers 16 :

	Model = Sequential

id	Couche 1	Couche 2	Couche 3	Couche 4	Couche 5	Couche 6	Couche 7	Couche 8	Couche 9	Couche 10	Couche 11	Couche 12	Couche 13	Couche 14	Couche 15	Couche 16
Base	Conv2D, 32,3 , Activation=sigmoid Input_shape(224, 224, 1)	MaxPool2D (2, 2)	BatchNormalization	Conv2D , 64, 3 Activation=sigmoid	MaxPool2D (2 , 2)	BatchNormalization	Conv2D , 128 ,3, Activation=sigmoid	MaxPool2D (2 , 2)	BatchNormalization	Conv2D , 256 ,3, Activation=sigmoid	MaxPool2D (2 , 2)	BatchNormalization	Flatten	Dense, 128 , activation=sigmoid	Dropout, 0.5	Dense, 1,activation=softmax

Compilation			Fit			Data			
Optimizer	Loss	metrics	Epoch	Final loss	Final accuracy	Batch_size	Height	Width	Class_mode
Adam	Binary_crossentropy	accuracy	40			32	224	224	binary



Prédictions

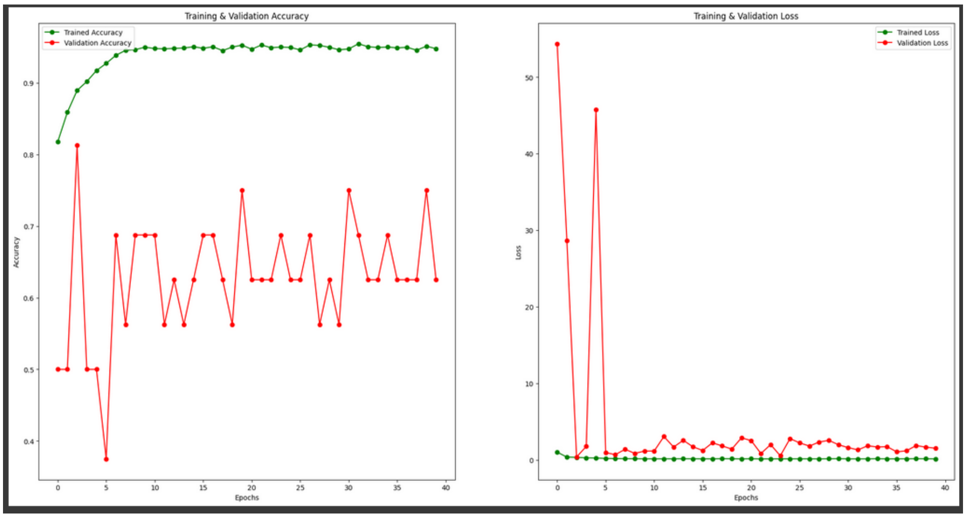
```
                      precision    recall  f1-score   support

Pneumonia (Classe: 0)      0.00      0.00      0.00       390
   Normal (Classe: 1)      0.38      1.00      0.55       234

             accuracy                          0.38       624
            macro avg      0.19      0.50      0.27       624
         weighted avg      0.14      0.38      0.20       624
```

	0	1
0	0	390
1	0	234

Optimizer rmsprop

	Model = Sequential															
id	Couche 1	Couche 2	Couche 3	Couche 4	Couche 5	Couche 6	Couche 7	Couche 8	Couche 9	Couche 10	Couche 11	Couche 12	Couche13	Couche 14	Couche 15	Couche 16
Base	Conv2D, 32,3, Activation=relu Input_shape(224,224, 1)	MaxPool2D(2, 2)	BatchNormalization	Conv2D, 64,3 Activation=relu	MaxPool2D (2, 2)	BatchNormalization	Conv2D, 128,3, Activation=relu	MaxPool2D (2, 2)	BatchNormalization	Conv2D, 256, 3, Activation=relu	MaxPool2D (2, 2)	BatchNormalization	Flatten	Dense, 128, activation=relu	Dropout, 0.5	Dense, 1,activation=sigmoid

Compilation			Fit			Data			
Optimizer	Loss	metrics	Epoch	Final loss	Final accuracy	Batch_size	Height	Width	Class_mode
rmsprop	Binary_crossentropy	accuracy	40	0.2708671987056732	0.9022436141967773	32	224	224	binary



Prédictions

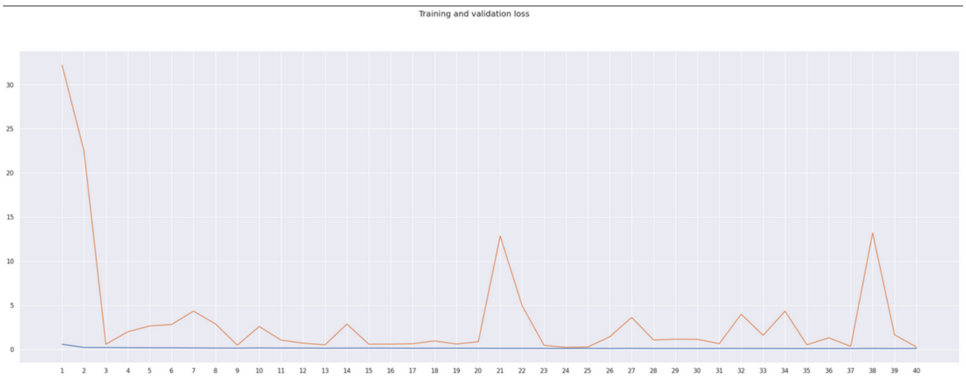
```
                       precision    recall  f1-score   support

Pneumonia (Classe: 0)       0.92      0.93      0.92       390
   Normal (Classe: 1)       0.87      0.86      0.87       234

             accuracy                           0.90       624
            macro avg       0.90      0.89      0.90       624
         weighted avg       0.90      0.90      0.90       624
```

	0	1
0	361	29
1	32	202

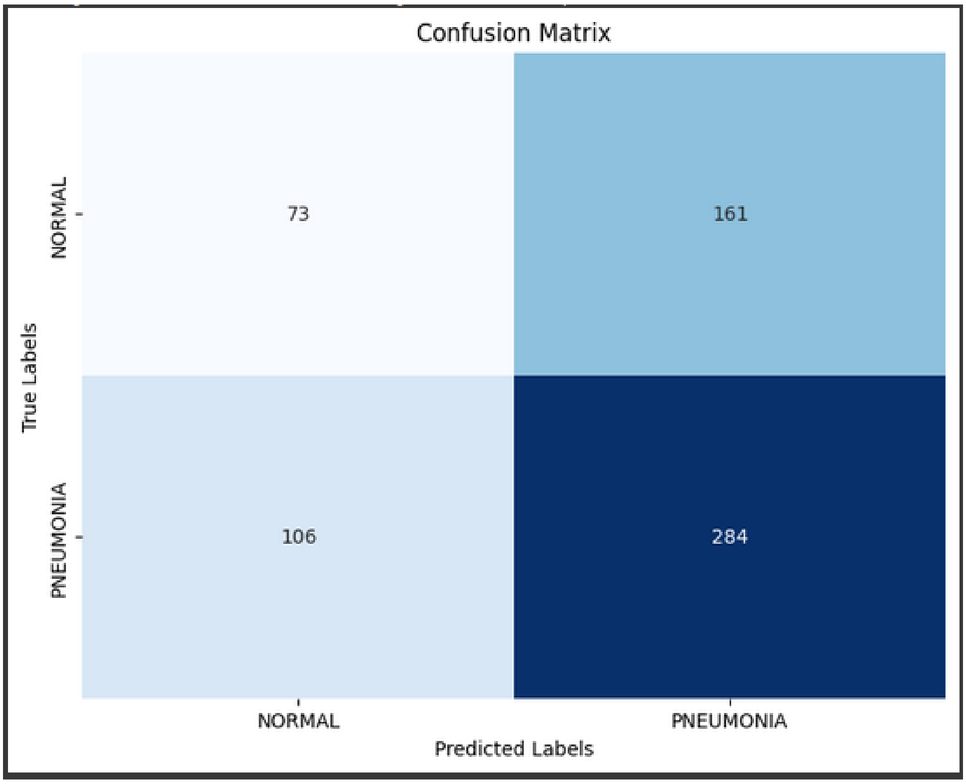
Model 2 sigmoid for last layer

```python
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))


model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))


model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))



model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy', tf.keras.metrics.Recall(name='recall') ])
```

Compilation			Fit			Data			
Optimizer	Loss	metrics	Epoch	Final loss	Final accuracy	Batch_size	Height	Width	Class_mode
adam	Binary_crossentropy	accuracy	40	0.3533	0.8990	32	224	224	binary



Prédictions

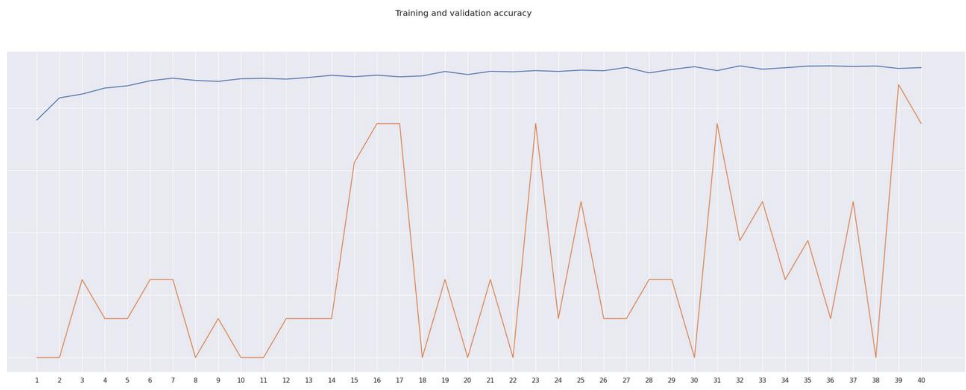



```
20/20 [==============================] - 10s 398ms/step
              precision    recall  f1-score   support

           0       0.35      0.27      0.31       234
           1       0.62      0.70      0.66       390

    accuracy                           0.54       624
   macro avg       0.48      0.49      0.48       624
weighted avg       0.52      0.54      0.52       624
```

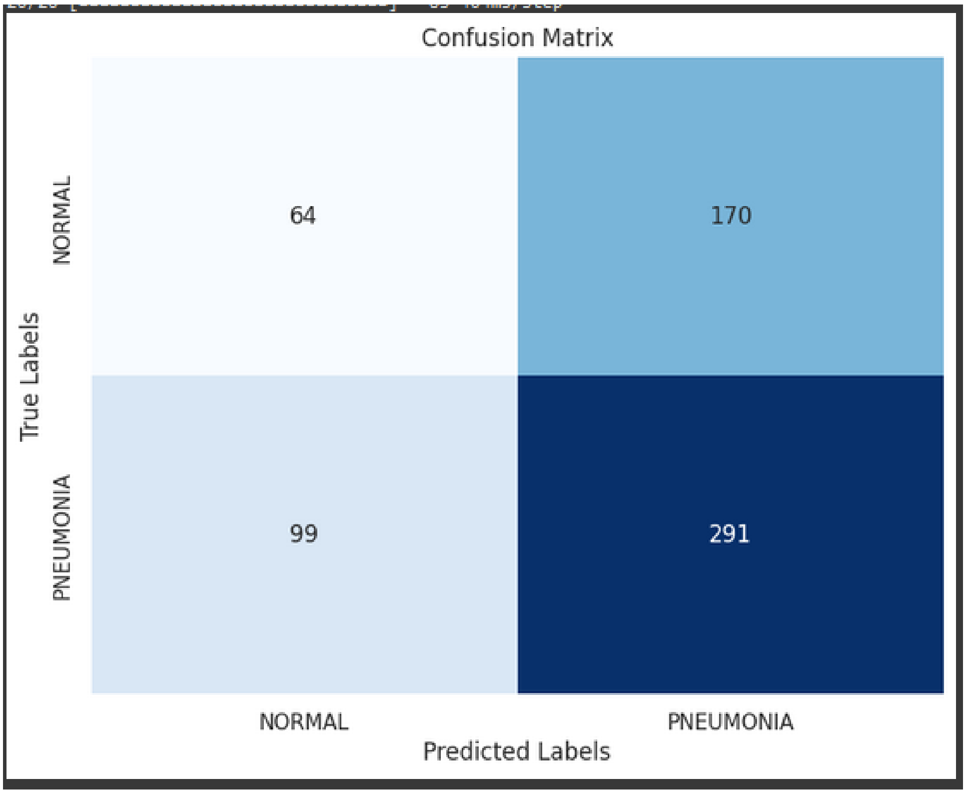
Model 2 Image taille 256

```python
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))


model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))


model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))



model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy', tf.keras.metrics.Recall(name='recall') ])
```

Compilation			Fit			Data			
Optimizer	Loss	metrics	Epoch	Final loss	Final accuracy	Batch_size	Height	Width	Class_mode
adam	Binary_crossentropy	accuracy	40	0.4275	0.8734	32	256	256	binary



<u>Prédictions</u>

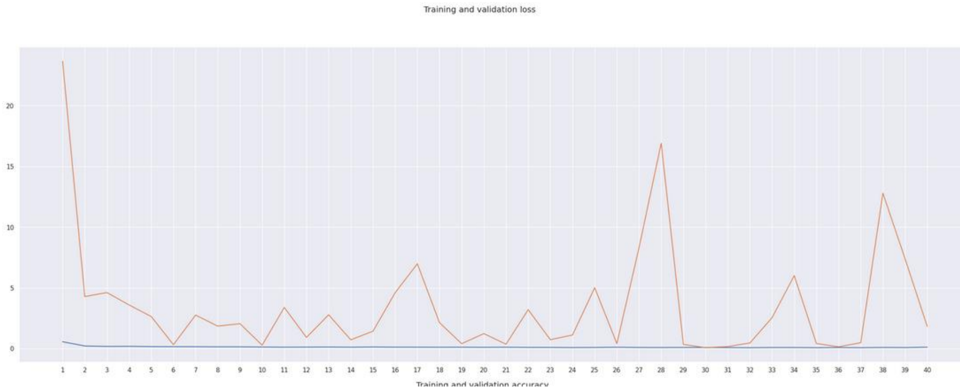


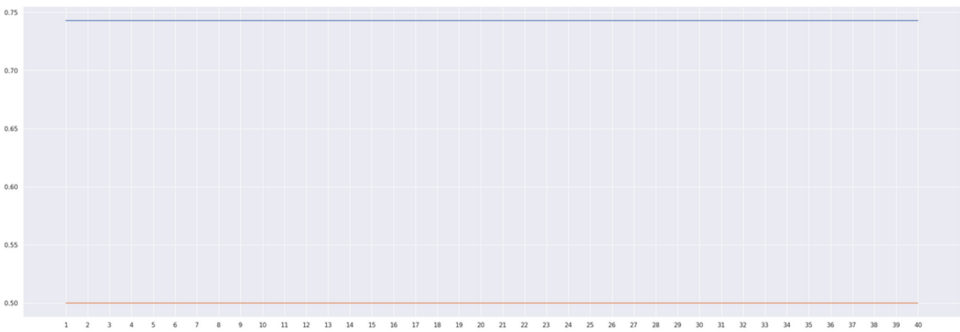
	precision	recall	f1-score	support
0	0.44	0.30	0.36	234
1	0.65	0.76	0.70	390
accuracy			0.59	624
macro avg	0.54	0.53	0.53	624
weighted avg	0.57	0.59	0.57	624

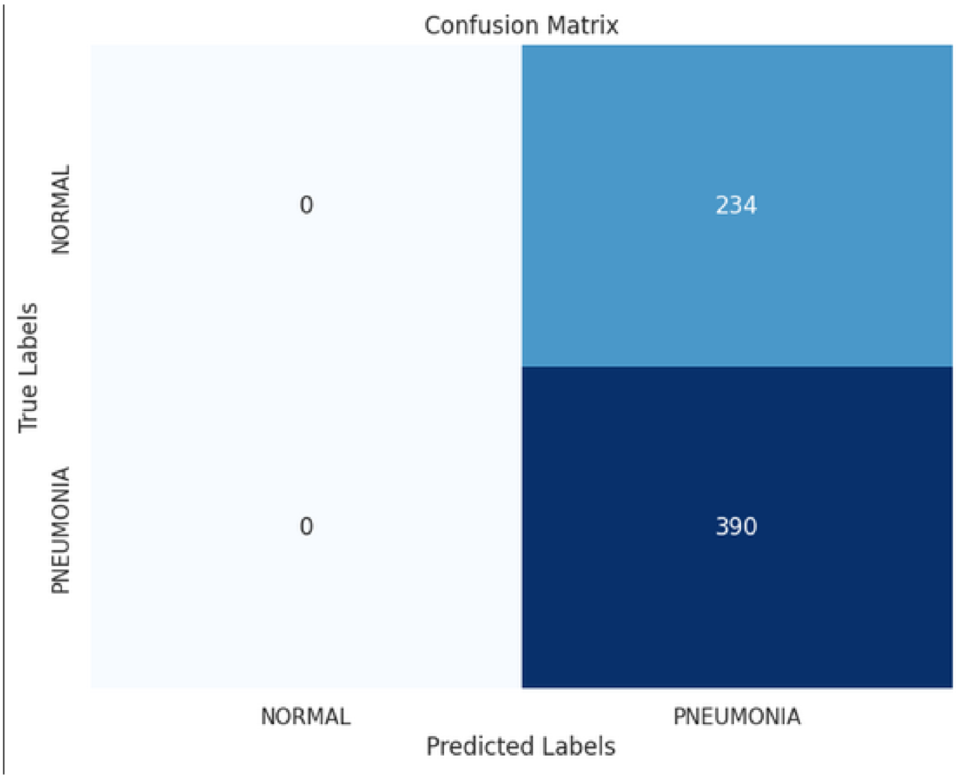
Model 2 sofmax for last layer

```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))


model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))


model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))



model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='softmax'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy', tf.keras.metrics.Recall(name='recall') ])
history = model.fit(train_generator,
          validation_data=validation_generator,
          epochs=40,
          verbose=1)
model.summary()
```

Compilation			Fit			Data			
Optimizer	Loss	metrics	Epoch	Final loss	Final accuracy	Batch_size	Height	Width	Class_mode
adam	Binary_crossentropy	accuracy	40	1.0812	0.6250	32	224	224	binary



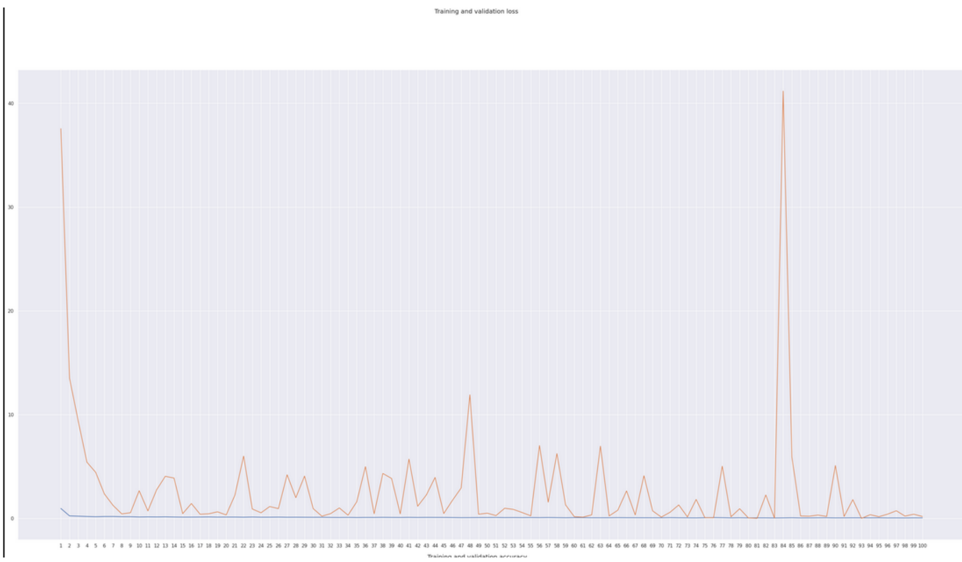


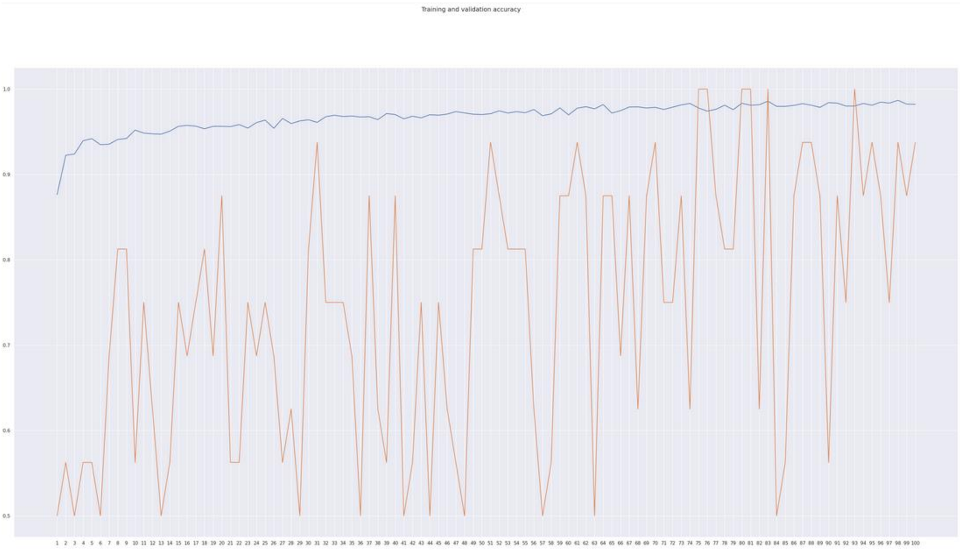



```
20/20 [==============================] - 8s 385ms/step
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       234
           1       0.62      1.00      0.77       390

    accuracy                           0.62       624
   macro avg       0.31      0.50      0.38       624
weighted avg       0.39      0.62      0.48       624
```

<u>Model 2 epoch 100 with image 256</u>

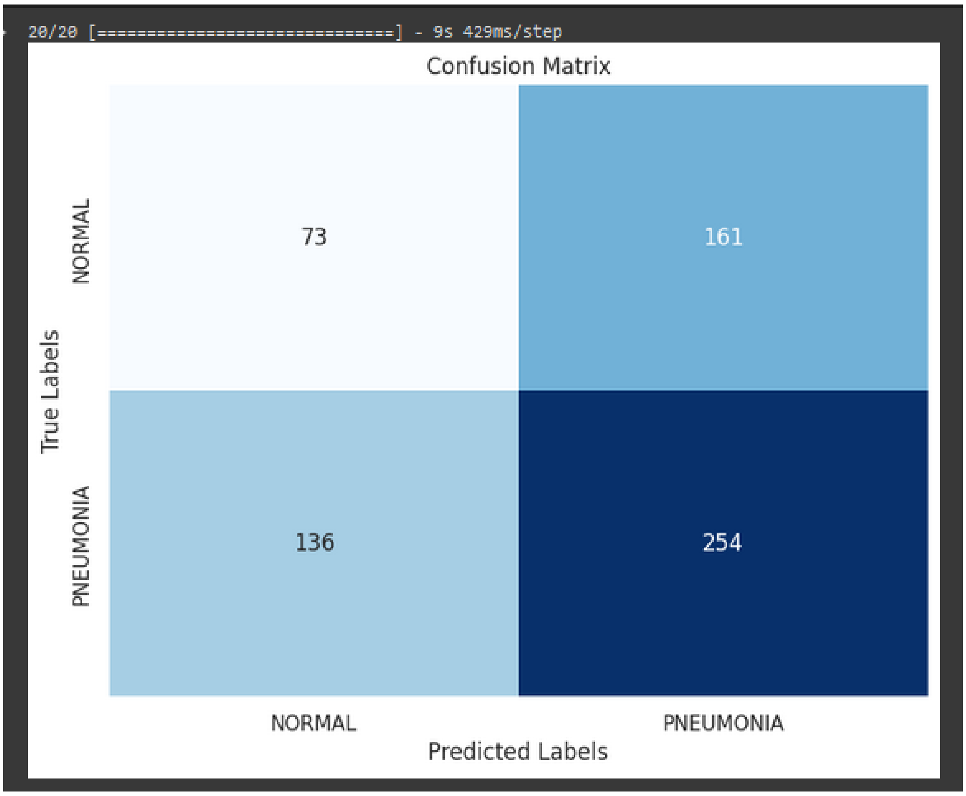
```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))


model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))


model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))



model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy', tf.keras.metrics.Recall(name='recall') ])
```





Compilation			Fit			Data			
Optimizer	Loss	metrics	Epoch	Final loss	Final accuracy	Batch_size	Height	Width	Class_mode
adam	Binary_crossentropy	accuracy	40	0.2877	0.9343	32	256	256	binary

<u>Prédictions</u>

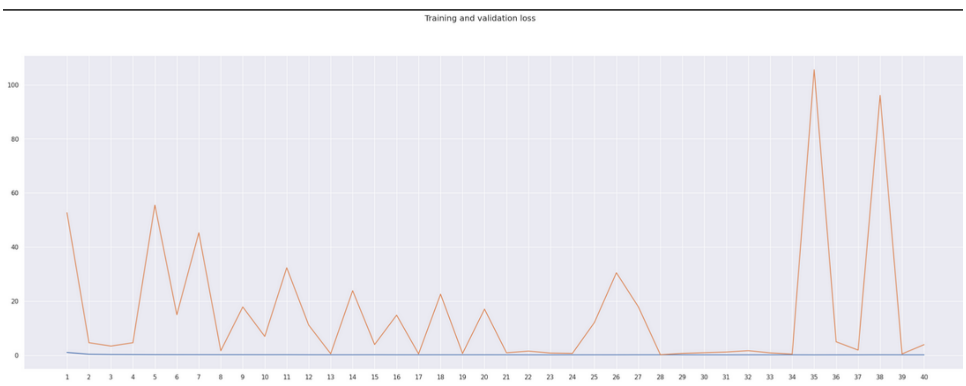


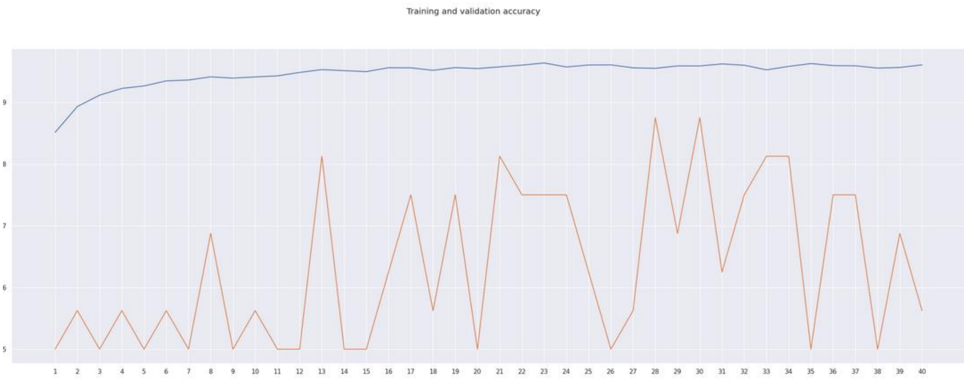
0	0.38	0.34	0.36	234
1	0.63	0.67	0.65	390
accuracy			0.54	624
macro avg	0.50	0.50	0.50	624
weighted avg	0.53	0.54	0.54	624

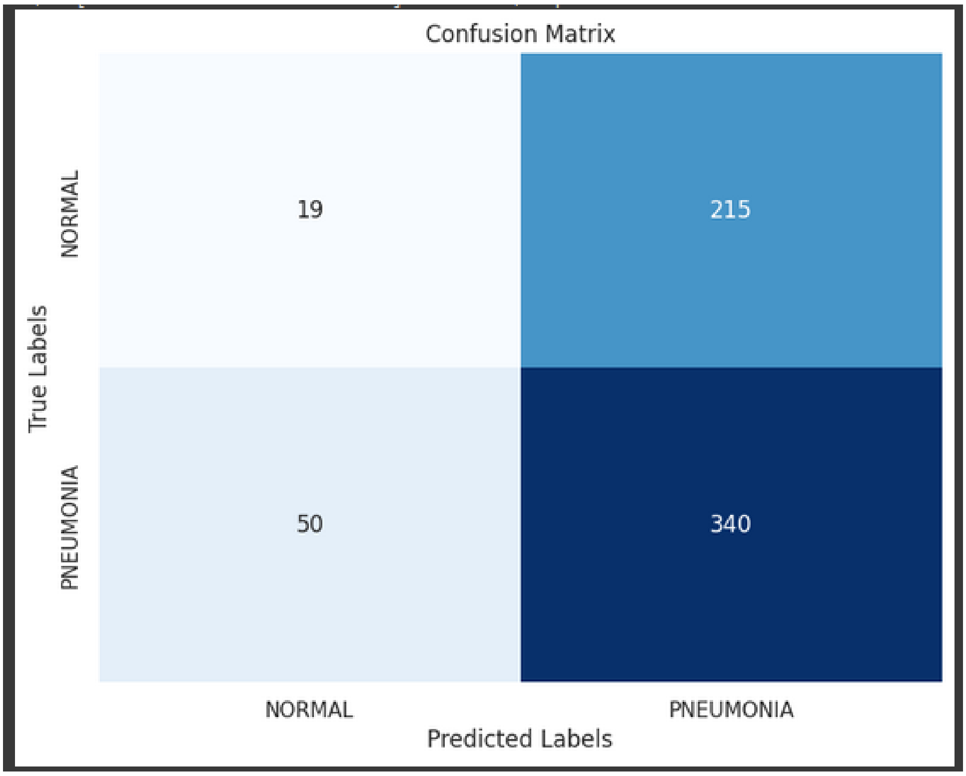
<u>Model 2 optimizer rmsprop</u>

```python
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))


model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))


model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))



model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy', tf.keras.metrics.Recall(name='recall') ])
history = model.fit(train_generator,
          validation_data=validation_generator,
          epochs=40,
          verbose=1)
model.summary()
```

Compilation			Fit			Data			
Optimizer	Loss	metrics	Epoch	Final loss	Final accuracy	Batch_size	Height	Width	Class_mode
rmsprop	Binary_crossentropy	accuracy	40	2.2332	0.7356	32	256	256	binary



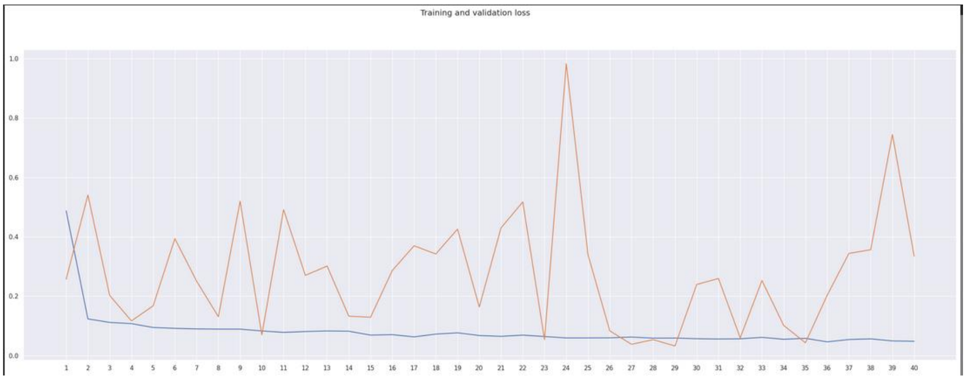


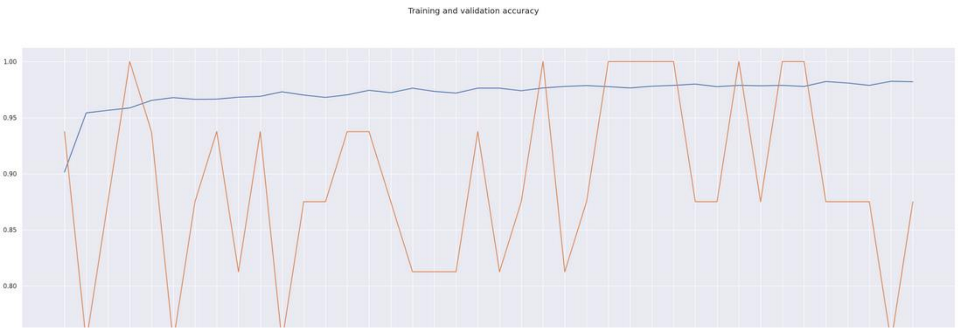


	precision	recall	f1-score	support
0	0.38	0.11	0.17	234
1	0.63	0.89	0.73	390
accuracy			0.60	624
macro avg	0.50	0.50	0.45	624
weighted avg	0.53	0.60	0.52	624

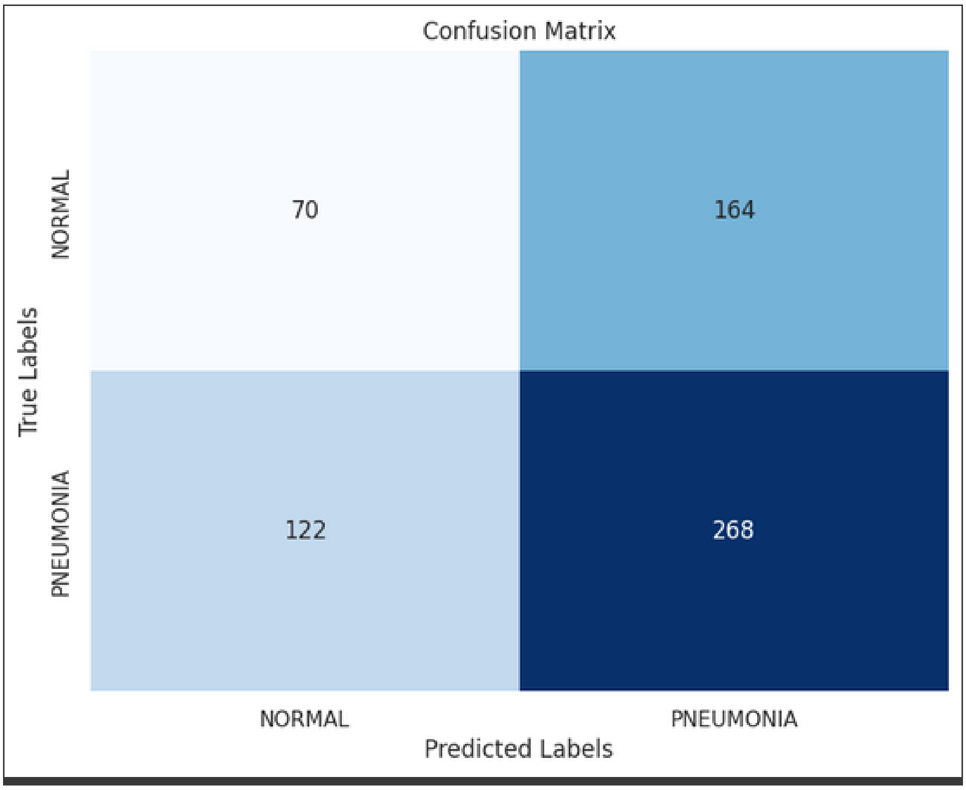
Model VGG16 sans mise à jour du model

```python
vgg16 = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Geler les poids des couches de base pour éviter leur mise à jour lors de l'entraînement permet de préserver les connaissances du modele et d'éviter le surajustement
for layer in vgg16.layers:
    layer.trainable = False

model = Sequential()

model.add(vgg16)

# Ajouter des couches supplémentaires pour la classification
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy', tf.keras.metrics.Recall(name='recall') ])
history = model.fit(train_generator,
          validation_data=validation_generator,
          epochs=40,
          verbose=1)
model.summary()
```





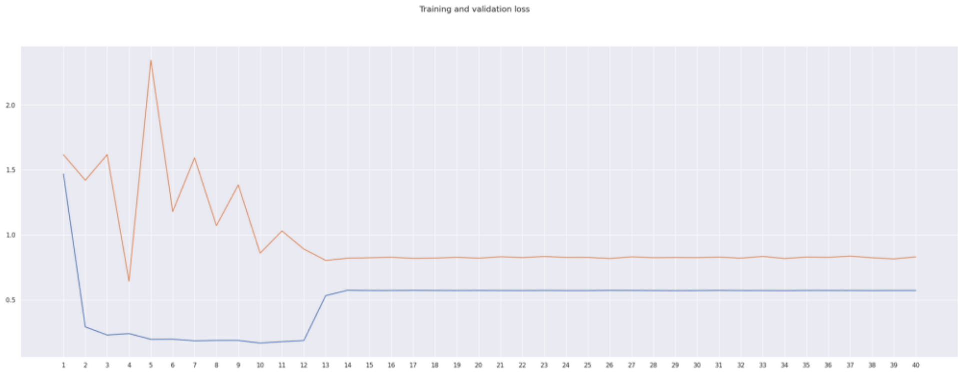
Compilation			Fit			Data			
Optimizer	Loss	metrics	Epoch	Final loss	Final accuracy	Batch_size	Height	Width	Class_mode
adam	Binary_crossentropy	accuracy	40	0.3726	0.9103	32	224	224	binary



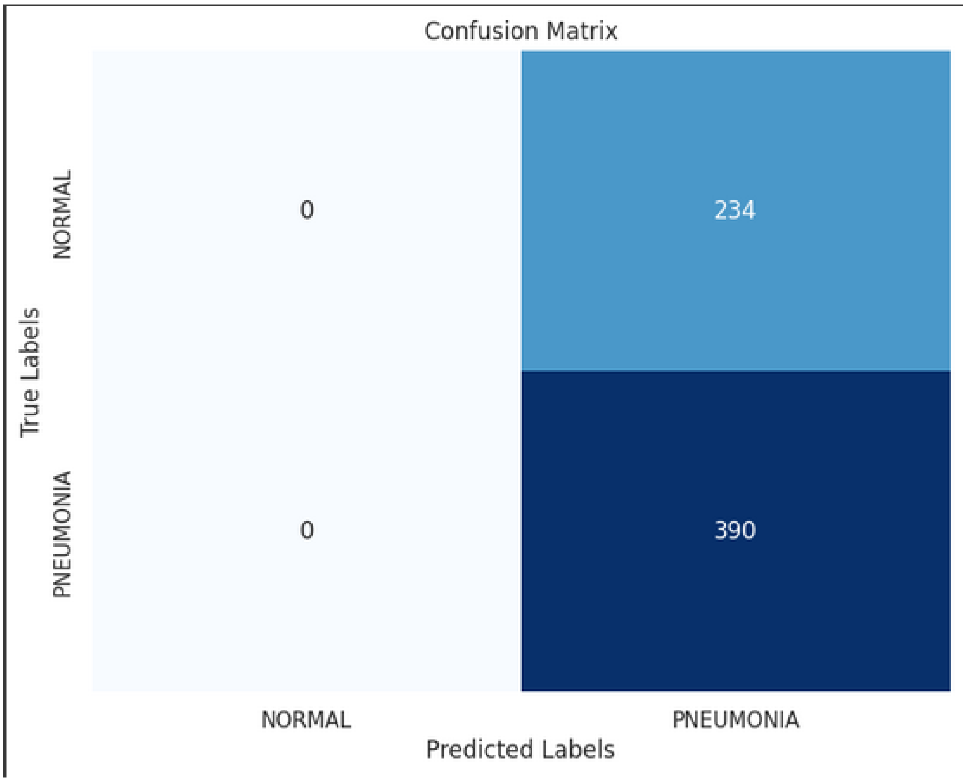
	precision	recall	f1-score	support
0	0.36	0.29	0.32	234
1	0.62	0.68	0.65	390
accuracy			0.54	624
macro avg	0.49	0.49	0.49	624
weighted avg	0.52	0.54	0.53	624

Model VGG16 avec mise à jour du model

```
vgg16 = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Geler les poids des couches de base pour éviter leur mise à jour lors de l'entraînement permet de préserver les connaissances du modele et d'éviter le surajustement
# for layer in vgg16.layers:
#   layer.trainable = False

model = Sequential()

model.add(vgg16)

# Ajouter des couches supplémentaires pour la classification
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy', tf.keras.metrics.Recall(name='recall') ])
history = model.fit(train_generator,
          validation_data=validation_generator,
          epochs=40,
          verbose=1)
model.summary()
```



Compilation			Fit			Data			
Optimizer	Loss	metrics	Epoch	Final loss	Final accuracy	Batch_size	Height	Width	Class_mode
adam	Binary_crossentropy	accuracy	40	0.6964	0.6250	32	224	224	binary



	precision	recall	f1-score	support
0	0.00	0.00	0.00	234
1	0.62	1.00	0.77	390
accuracy			0.62	624
macro avg	0.31	0.50	0.38	624
weighted avg	0.39	0.62	0.48	624

Simple train test split model

```
# Liste pour stocker les scores de chaque fold
scores = []

for train_index, val_index in kf.split(train_generator.filenames):
    # Diviser les données en ensembles d'entraînement et de validation pour chaque fold
    train_files = [train_generator.filenames[i] for i in train_index]
    val_files = [train_generator.filenames[i] for i in val_index]

    train_datagen_fold = ImageDataGenerator(rescale=1./255)
    train_generator_fold = train_datagen_fold.flow_from_directory(
        train_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode='binary',
        classes=["NORMAL", "PNEUMONIA"],
        shuffle=False,
        subset='training',
        seed=42)

    val_datagen_fold = ImageDataGenerator(rescale=1./255)
    val_generator_fold = val_datagen_fold.flow_from_directory(
        train_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode='binary',
        classes=["NORMAL", "PNEUMONIA"],
        shuffle=False,
        subset='validation',
        seed=42)

    # Créer, entraîner et évaluer le modèle pour chaque fold
model = Sequential()
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy', tf.keras.metrics.Recall(name='recall') ])
```

```
Fold 1 - Accuracy: 0.5
Mean Accuracy: 0.5
Test Accuracy: 0.625
```

Conclusion

Après plusieurs tests nous avons choisis comme model notre 2eme modèle (Modèle Principale) car c’est ce modèle qui nous donne les meilleurs résultats.

Le modèle est défini de la manière suivante :

```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))


model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))


model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))



model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy', tf.keras.metrics.Recall(name='recall') ])
```

On utilise 4 blocs similaires de couches comprenant une couche de convolution suivie d'une normalisation par lots (batch normalization) et d'une couche de pooling (max pooling). Pour les couches de convolution à travers les 4 blocs, le nombre de filtres augmente de 32 à 128 avec une taille de noyau de 3x3.

Après la couche de convolution dans chaque bloc, j'applique une normalisation par lots pour normaliser les résultats. Ensuite, ils sont introduits dans la fonction d'activation relu. Ensuite, j'applique la couche de pooling avec une taille de regroupement de 2x2.

Après les 4 blocs de couches de convolution et de pooling, on rajoute une couche Dense avec 512 neurones, avec relu comme fonction d'activation. Enfin, j'inclus une couche de sortie avec 1 neurone utilisant la fonction sigmoïde comme fonction d'activation pour notre tâche de classification binaire

La taille des images qui sont passé dans le modèle ont une taille de 224 par 224.

<u>Comparaison CNN et train test split</u>

Le train test split est une technique couramment utilisée en apprentissage automatique pour évaluer les performances d'un modèle. Elle consiste à diviser l'ensemble de données disponible en deux sous-ensembles : l'ensemble d'apprentissage (train) et l'ensemble de test (test). L'ensemble d'apprentissage est utilisé pour entraîner le modèle, tandis que l'ensemble de test est utilisé pour évaluer ses performances sur des données non vues auparavant.

Un réseau neuronal convolutif (CNN) est un type d'algorithme d'apprentissage profond couramment utilisé pour l'analyse de données visuelles, telles que les images et les vidéos. Les CNN sont particulièrement efficaces dans des tâches telles que la classification d'images, la détection d'objets et la reconnaissance d'images.

Dans notre cas le plus adapté est le CNN car il a une meilleure précision et la méthode d’entrainement est plus adapté.