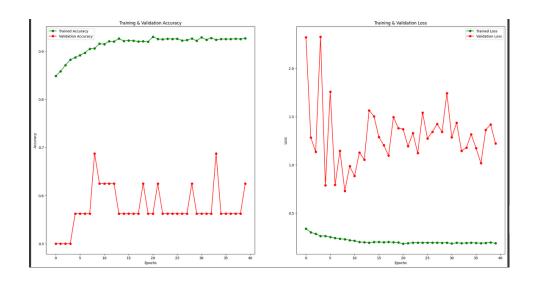
# Médicale Image Recognition

### **Tested Model and their predict**

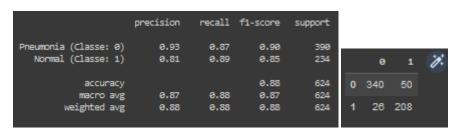
### Activation sigmoid for all layers

						М	odel	= S	eque	ntial						
i	Cou	Со	Cou	Cou	С	Cou	Cou	С	Cou	Cou	С	Cou	С	Cou	С	Cou
d	che	uc	che	che	О	che	che	О	che	che	О	che	О	che	О	che
	1	he	3	4	uc	6	7	uc	9	10	uc	12	u	14	u	16
		2			he			he			he		С		С	
					5			8			11		h		h	
													е		е	
													1		1	
													3		5	
В	Con	М	Bat	Con	М	Bat	Con	М	Bat	Con	М	Bat	Fl	Den	D	Den
a	v2D,	ax	chN	v2D	ax	chN	v2D	ax	chN	v2D	ax	chN	at	se,	r	se,
S	32,3	Ро	orm	,	Р	orm	,	Р	orm	,	Р	orm	te	128	О	1,ac
е	,	ol2	aliz	64,	О	aliz	128	О	aliz	256	О	aliz	n	,	р	tivat
	Acti	D(	atio	3	ol	atio	,3,	ol	atio	,3,	ol	atio		acti	0	ion=
	vati	2,	n	Acti	2	n	Acti	2	n	Acti	2	n		vati	u	sig
	on=s	2)		vati	D		vati	D		vati	D			on=	t,	moi
	igm			on=	(2		on=	(2		on=	(2			sig	0.	d
	oid			sig	,		sig	,		sig	,			moi	5	
	Inpu			moi	2)		moi	2)		moi	2)			d		
	t_sh			d			d			d						
	ape(															
	224,															
	224,															
	1)															

C	ompilatio	n		Fit			Da	ata	
Opti	Loss	metr	Ер	Final loss	Final accuracy	Batch	Hei	Wi	Class_
mizer		ics	och			_size	ght	dth	mode
Ada	Binary_cros	accu	40	0.28894641	0.87820512	32	224	22	binary
m	sentropy	racy		99542999	05635071			4	



### <u>Prédiction</u>:

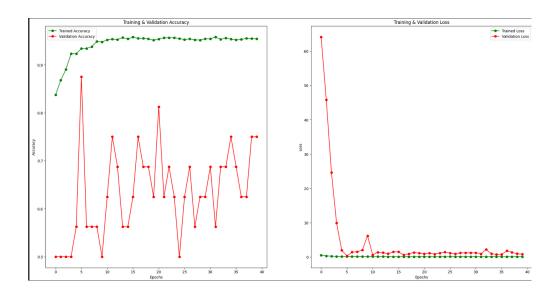


## Activation sigmoid for layers 16

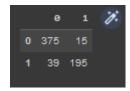
	1															
						M	1ode	el = S	Sequ	enti	al					
id	Со	Со	Со	Со	Со	Со	Со	Со	Со	Со	Со	Со	Со	Со	Со	Со
	uc	uc	uc	uc	uc	uc	uc	uc	uc	uc	uc	uc	uc	uc	uc	uc
	he	he	he	he	he	he	he	he	he	he	he	he	he	he	he	he
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bas	Со	Ma	Bat	Co	Ma	Bat	Со	Ma	Bat	Со	Ma	Bat	Fla	De	Dr	De
е	nv	хP	ch	nv	хP	ch	nv	хP	ch	nv	хP	ch	tte	nse	ор	nse
	2D,	ool	No	2D,	ool	No	2D,	ool	No	2D,	ool	No	n	,	out	,
	32,	2D(	rm	64,	2D	rm	12	2D	rm	25	2D	rm		12	,	1,a
	3,	2,	aliz	3	(2,	aliz	8,3	(2,	aliz	6,3	(2,	aliz		8,	0.5	cti
	Act	2)	ati	Act	2)	ati	,	2)	ati	,	2)	ati		act		vat
	iva		on	iva		on	Act		on	Act		on		iva		ion
	tio			tio			iva			iva				tio		=si
	n=r			n=r			tio			tio				n=r		gm
	elu			elu			n=r			n=r				elu		oid
	Inp						elu			elu						
	ut_															

sha	ı							
ре								
22								
4,2								
24								
1)								

Со	mpilati	on		Fit			Da	ita	
Optimi	Loss	metrics	Epoch	Final	Final	Batch_	Height	Width	Class_
zer				loss	accurac	size			mode
					у				
Adam	Binary_	accurac	40	0.242	0.913	32	224	224	binary
	crossen	у		28760	46156				
	tropy			60009	59713				
	- 7			0027	745				



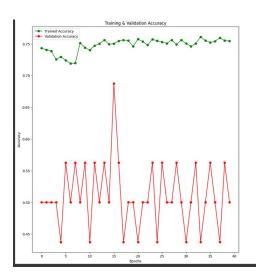
## <u>Prédiction</u>

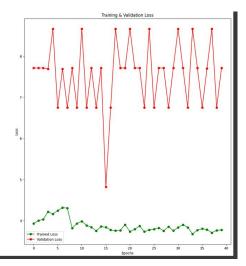


# <u>Activation relu for all layers</u>:

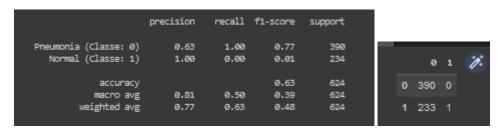
						Mc	del	= Se	equer	ntial						
i	Couc	Со	Cou	Со	Со	Cou	Со	Со	Cou	Со	Со	Cou	С	Со	С	Cou
d	he 1	uc	che	uch	uc	che	uch	uc	che	uch	uc	che	О	uc	О	che
		he	3	e 4	he	6	e 7	he	9	е	he	12	uc	he	u	16
		2			5			8		10	11		h	14	С	
													e		h	
													1		е	
													3		1	
															5	
В	Conv	Ma	Batc	Со	М	Batc	Со	М	Batc	Со	М	Batc	Fl	De	D	De
а	2D,	хP	hNo	nv	ax	hNo	nv	ax	hNo	nv	ax	hNo	at	nse	r	nse
S	32,3,	ool	rmal	2D,	Ро	rmal	2D,	Ро	rmal	2D,	Ро	rmal	te	,	0	,
е	Activ	2D	izati	64,	ol	izati	12	ol	izati	25	ol	izati	n	12	р	1,a
	ation	(2,	on	3	2	on	8,3	2	on	6,3	2	on		8,	0	ctiv
	=relu	2)		Act	D		,	D		,	D			act	ut	atio
	Inpu			iva	(2,		Act	(2,		Act	(2,			iva	,	n=r
	t_sh			tio	2)		iva	2)		iva	2)			tio	0.	elu
	ape(			n=r			tio			tio				n=r	5	
	224,			elu			n=r			n=r				elu		
	224,						elu			elu						
	1)															

Со	mpilati	on		Fit			Da	ita	
Optimi	Loss	metrics	Epoch	Final	Final	Batch_	Height	Width	Class_
zer				loss	accurac	size			mode
					У				
Adam	Binary_	accurac	40	5.759	0.626	32	224	224	binary
	crossen	у		63592	60259				
	,		52929	00840					
	. ,			69	759				





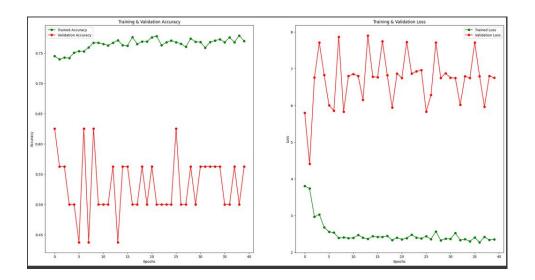
### <u>Prédiction</u>



### Activation relu for layers 16:

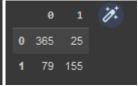
						Mo	odel	= Se	equei	ntial						
i	Cou	Со	Cou	Cou	С	Cou	Cou	С	Cou	Cou	С	Cou	С	Cou	С	Со
d	che	uc	che	che	0	che	che	О	che	che	О	che	О	che	О	uch
	1	he	3	4	uc	6	7	uc	9	10	uc	12	u	14	u	e
		2			he			he			he		С		С	16
					5			8			11		h		h	
													e		е	
													1		1	
													3		5	
В	Con	М	Batc	Con	М	Batc	Con	М	Batc	Con	М	Batc	Fl	Den	D	De
а	v2D,	ax	hNo	v2D	ax	hNo	v2D	ax	hNo	v2D	ax	hNo	at	se,	r	nse
S	32,3	Ро	rma	,	Ро	rma	,	Ро	rma	,	Ро	rma	te	128	0	,
е	,	ol2	lizat	64,	ol	lizat	128	ol	lizat	256	ol	lizat	n	,	р	1,a
	Acti	D(	ion	3	2	ion	,3,	2	ion	,3,	2	ion		acti	0	ctiv
	vati	2,		Acti	D		Acti	D		Acti	D			vati	u	ati
	on=s	2)		vati	(2		vati	(2		vati	(2			on=	t,	on
	igm			on=	,		on=	,		on=	,			sig	0.	=re
	oid			sig	2)		sig	2)		sig	2)			moi	5	lu
	Inpu			moi			moi			moi				d		
	t_sh			d			d			d						
	ape(															
	224,															
	224,															
	1)															

C	ompilatio	n		Fit			Da	ata	
Opti Loss metr Ep Final loss				Final accuracy	Batch	Hei	Wi	Class_	
mizer	•					_size	ght	dth	mode
Ada	Binary_cros	accu	40	1.31892430	0.83333331	32	224	22	binary
m	sentropy	racy		78231812	34651184			4	



### **Prdiction**

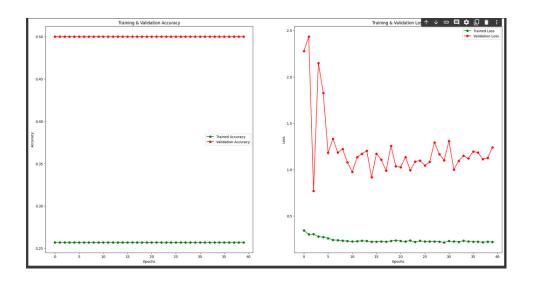
	precision	recall	f1-score	support
Pneumonia (Classe: 0)	0.82	0.94	0.88	390
Normal (Classe: 1)	0.86	0.66	0.75	234
accupacy			0.83	624
accuracy macro avg	0.84	0.80	0.83	624
weighted avg	0.84	0.83	0.83	624



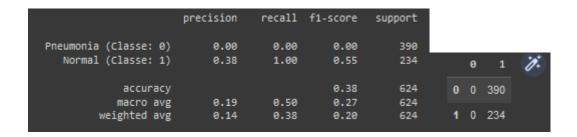
# Activation softmax for layers 16:

i	Cou	Со	Cou	Cou	С	Cou	Cou	С	Cou	Cou	С	Cou	С	Cou	С	Cou
d	che	uc	che	che	О	che	che	О	che	che	О	che	О	che	О	che
	1	he	3	4	uc	6	7	uc	9	10	uc	12	u	14	u	16
		2			he			he			he		С		С	
					5			8			11		h		h	
													e		е	
													1		1	
													3		5	
В	Con	М	Bat	Con	М	Bat	Con	М	Bat	Con	М	Bat	Fl	Den	D	Den
a	v2D,	ax	chN	v2D	ax	chN	v2D	ax	chN	v2D	ax	chN	at	se,	r	se,
S	32,3	Ро	orm	,	Р	orm	,	Р	orm	,	Р	orm	te	128	0	1,ac
е	,	ol	aliz	64,	0	aliz	128	0	aliz	256	0	aliz	n	,	р	tivat
	Acti	2D	atio	3	ol	atio	,3,	ol	atio	,3,	ol	atio		acti	0	ion=
	vati	(2,	n	Acti	2	n	Acti	2	n	Acti	2	n		vati	u	soft
	on=s	2)		vati	D		vati	D		vati	D			on=	t,	max
	igm			on=	(2		on=	(2		on=	(2			sig	0.	
	oid			sig	,		sig	,		sig	,			moi	5	
	Inpu			moi	2)		moi	2)		moi	2)			d		
	t_sh			d			d			d						
	ape(															
	224,															
	224,															
	1)															

	Compilation			Fit			Da	ata	
Optimiz	Loss	metric	Epoc	Fin	Final	Batch_si	Heig	Widt	Class_mo
er		S	h	al	accura	ze	ht	h	de
				loss	су				
Adam	Binary_crossent	accura	40			32	224	224	binary
	ropy	су							



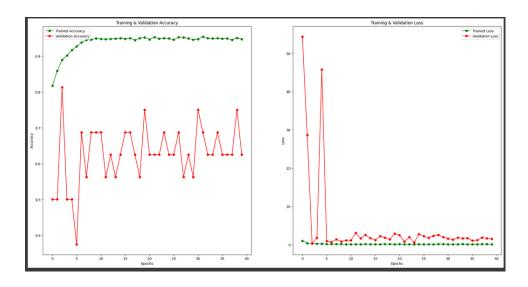
### **Prédictions**



### Optimizer rmsprop

	Model = Sequential															
i	Couc	С	Couc	С	Cou	Couc	С	Со	Со	Cou	Cou	Со	Со	С	Со	Со
d	he 1	О	he 3	О	che	he 6	О	uch	uc	che	che	uc	uch	0	uc	uc
		u		uc	5		uc	e 8	he	10	11	he	e1	uc	he	he
		С		h			h		9			12	3	h	15	16
		h		e			e							е		
		e		4			7							1		
		2												4		
В	Conv	М	Batc	С	Ma	Batc	С	Ma	Ва	Con	Ma	Ва	Fla	D	Dr	De
а	2D,	ax	hNor	0	хРо	hNor	0	хРо	tch	v2D,	хРо	tc	tte	е	ор	ns
S	32,3,	Р	mali	nv	ol2	mali	nv	ol2	No	256,	ol2	hN	n	ns	out	e,
е	Activ	0	zatio	2	D	zatio	2	D	rm	3,	D	or		e,	,	1,a
	ation	ol	n	D,	(2,	n	D,	(2,	ali	Acti	(2,	m		1	0.5	cti
	=relu	2		6	2)		1	2)	zat	vati	2)	ali		2		vat
	Input	D(		4,			2		ion	on=		zat		8,		ion
	_sha	2,		3			8,			relu		io		ac		=si
	pe(2	2)		Ac			3,					n		ti		gm
	24,22			ti			Ac							va		oid
	4, 1)			va			ti							ti		
				ti			va							0		
				0			ti							n		
				n			0							=r		
				=r			n							el		
				el			=r							u		
				u			el									
							u									

(	Compilation			Fit		Data				
Opti mize r	Loss	metrics	Epoch	Final loss	Final accuracy	Batch Height _size		Width	Class_ mode	
rmsp rop	Binary_c rossentr opy	accuracy	40	0.2708 671987 056732	0.9022 436141 967773	32	224	224	binary	

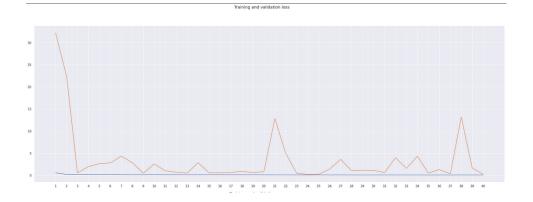


1
1
:
1 9 2

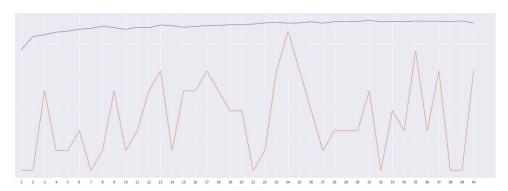
### Model 2 sigmoid for last layer

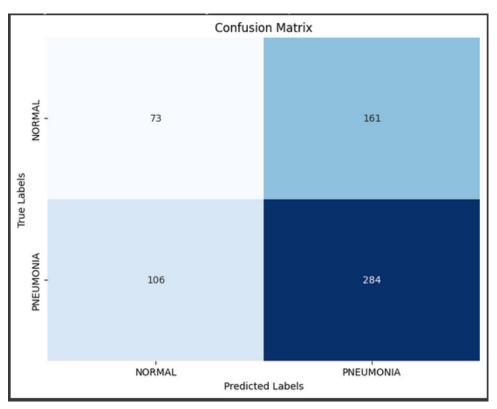
```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))
model.add(Conv2D(128, (3, 3), 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='sigmoid'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
             metrics=['accuracy', tf.keras.metrics.Recall(name='recall') ])
```

	Fit			Data					
Optimi	Loss	metric	Еро	Final	Final	Batch_s	Heig	Wid	Class_m
zer		S	ch	loss	accura	ize	ht	th	ode
					су				
adam	Binary_crossent	accura	40	0.35	0.899	32	224	224	binary
	ropy	су		33	0				



Training and validation accuracy





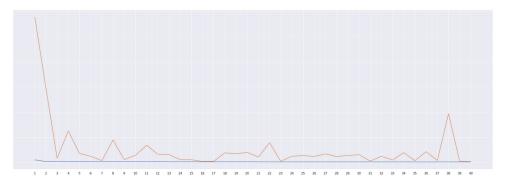
20/20 [=====			:===] - 10s	398ms/step
	precision	recall	f1-score	support
	0.35	0.07	0.34	224
0	0.35	0.27	0.31	234
1	0.62	0.70	0.66	390
accuracy			0.54	624
macro avg	0.48	0.49	0.48	624
weighted avg	0.52	0.54	0.52	624

### Model 2 Image taille 256

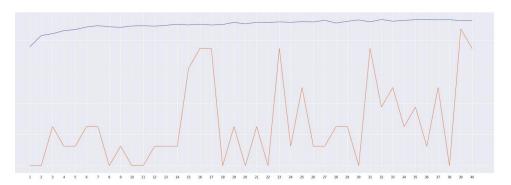
```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))
model.add(Conv2D(128, (3, 3), 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') ])
```

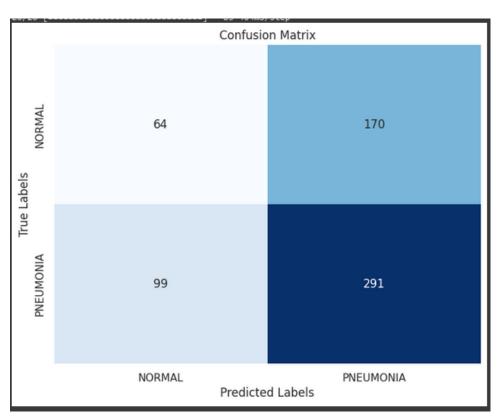
	Fit			Data					
Optimi	Loss	metric	Epo	Final	Final	Batch_s	Heig	Wid	Class_m
zer		S	ch	loss	accura	ize	ht	th	ode
					су				
adam	Binary_crossent	accura	40	0.42	0.873	32	256	256	binary
	ropy	су		75	4				

Training and validation loss



Training and validation accuracy



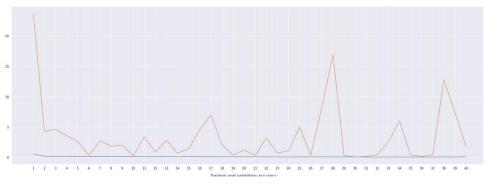


recision	recall	f1-score	support
0.44	0.30	0.36	234
0.65	0.76	0.70	390
		0.59	624
0.54	0.53	0.53	C04
0.54	0.53	0.53	624
0.57	0.59	0.57	624
	0.44 0.65 0.54 0.57	0.44 0.30 0.65 0.76 0.54 0.53	0.44 0.30 0.36 0.65 0.76 0.70 0.59 0.54 0.53 0.53

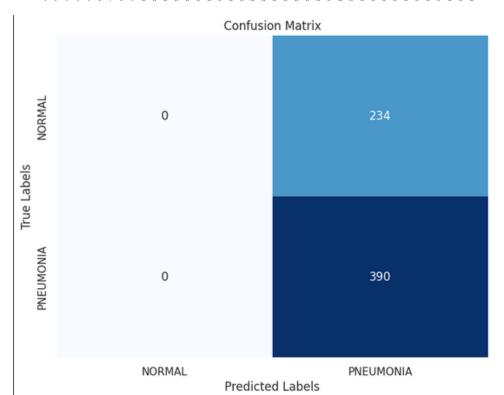
#### Model 2 sofmax for last layer

```
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			
Optimi	Loss	metric	Epo	Final	Final	Batch_s	Heig	Wid	Class_m
zer		S	ch	loss	accura	ize	ht	th	ode
					су				
adam	Binary_crossent	accura	40	1.08	0.625	32	224	224	binary
	ropy	су		12	0				



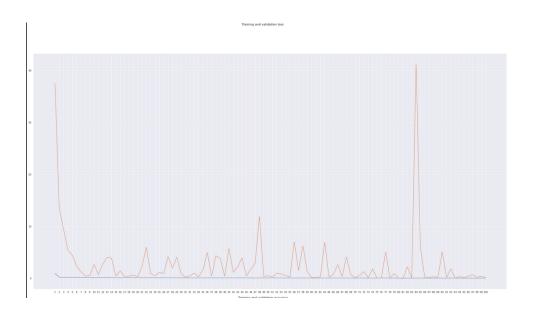


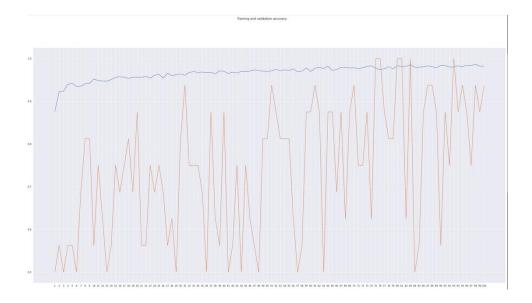


20/20 [=====	precision		==] - 8s 3 f1-score	885ms/step support
0	0.00	0.00	0.00	234
1	0.62	1.00	0.77	390
accuracy			0.62	624
macro avg	0.31	0.50	0.38	624
weighted avg	0.39	0.62	0.48	624

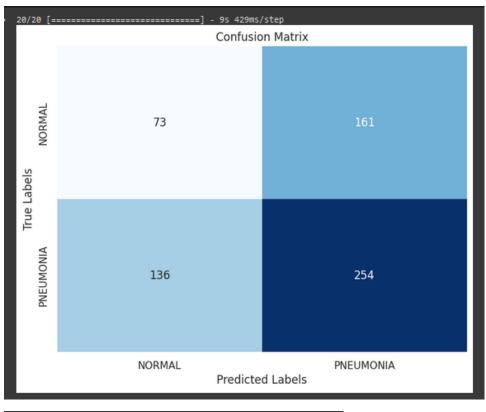
#### Model 2 epoch 100 with image 256

```
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') ])
```





	Fit			Data					
Optimi	Loss	metric	Epo	Final	Final	Batch_s	Heig	Wid	Class_m
zer		S	ch	loss	accura	ize	ht	th	ode
					су				
adam	Binary_crossent	accura	40	0.28	0.934	32	256	256	binary
	ropy	су		77	3				

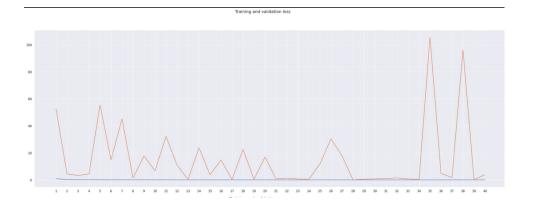


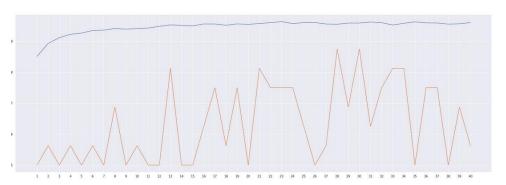
0	0.38	0.34	0.36	234
1	0.63	0.67	0.65	390
accuracy			0.54	624
macro avg	0.50	0.50	0.50	624
weighted avg	0.53	0.54	0.54	624

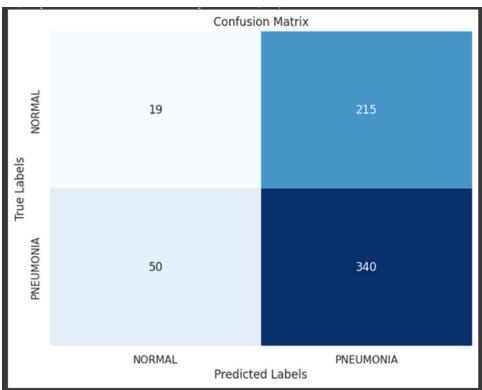
Model 2 optimizer rmsprop

```
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			
Optimi	Loss	metric	Epo	Final	Final	Batch_s	Heig	Wid	Class_m	
zer		S	ch	loss	accura	ize	ht	th	ode	
					су					
rmspro	Binary_crossent	accura	40	2.23	0.735	32	256	256	binary	
р	ropy	су		32	6					



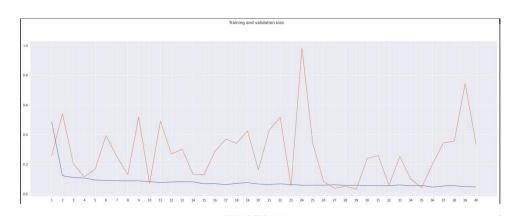




	precision	recall	f1-score	support
0	0.38	0.11	0.17	234
1	0.63	0.89	0.73	390
accuracy			0.60	624
macro avg	0.50	0.50	0.45	624
weighted avg	0.53	0.60	0.52	624

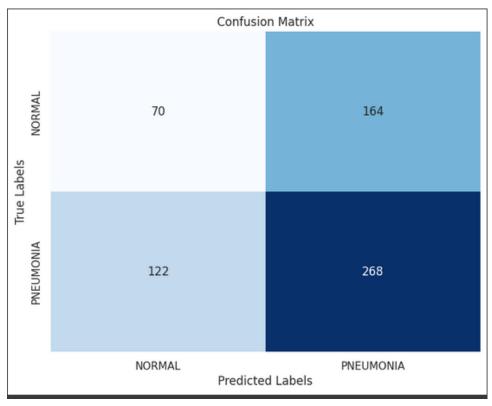
## Model VGG16 sans mise à jour du model

```
vgg16 = vGs16(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'entrainement 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(platten())
model.add(platten())
model.add(platten())
model.add(platten();
model.
```





Compilation		Fit		Data					
Optimi	Loss	metric	Еро	Final	Final	Batch_s	Heig	Wid	Class_m
zer		S	ch	loss	accura	ize	ht	th	ode
					су				
adam	Binary_crossent	accura	40	0.37	0.910	32	224	224	binary
	ropy	су		26	3				



	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

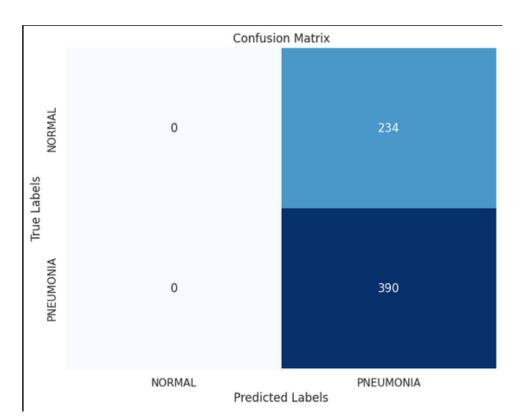




#### Training and validation accuracy



Compilation			Fit		Data				
Optimi	Loss	metric	Еро	Final	Final	Batch_s	Heig	Wid	Class_m
zer		S	ch	loss	accura	ize	ht	th	ode
					су				
adam	Binary_crossent	accura	40	0.69	0.625	32	224	224	binary
	ropy	су		64	0				



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

#### Simple train test split model

```
scores = []
for train_index, val_index in kf.split(train_generator.filenames):
    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,
       classs_mode='binary',
classes=["NORMAL", "PNEUMONIA"],
        shuffle=False,
        subset='validation',
        seed=42)
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.

#### Comparaison CNN et train test split

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.