Modele COVID-19 Recognition :

Import des biblio :

In [1]:
```
import glob
import shutil
import cv2
import os
from PIL import Image
import seaborn as sns
import numpy as np
from matplotlib import pyplot as plt
import tensorflow as tf
from tensorflow.keras.layers import *
from tensorflow.keras.models import *
from tensorflow.keras.preprocessing import *
from keras.applications.vgg16 import VGG16
from keras import layers

# from skimage.feature import local_binary_pattern

import warnings
# Ignore waring
warnings.filterwarnings('ignore')
```

Ce code est destiné à charger un ensemble d'images de radiographies de patients atteints de COVID-19 depuis un répertoire, à les redimensionner et à leur attribuer des étiquettes.

X : contiendra toutes les images sous forme de tableaux de pixels, chacune ayant été redimensionnée à 128x128 pixels.

y : contiendra uniquement des 1, indiquant que toutes les images sont étiquetées comme appartenant à la classe 'Covid'.

In [2]:
```
name_list = glob.glob("C:/Users/yosser/OneDrive/Bureau/3DNI/Projets/developpemen

labels = ['NORMAL', 'Covid']

#Load the training images and labels
X= []
y = []
for name in name_list:
  y.append(1)
  img = cv2.imread(name)
  img = tf.keras.preprocessing.image.img_to_array(img)
  img = cv2.resize(img,(128,128))

  X.append((img))
len(X)
```

Out[2]: 3616

ici , on fait le meme démarche pour charger les images des patients normal

x: contiendra toutes les images sous forme de tableaux de pixels, chacune ayant été redimensionnée à 128x128 pixels.

y: contiendra des 1(indique classe COVID) , et des 0 (indique classe Normal)

In [3]:
```
name_list = glob.glob("C:/Users/yosser/OneDrive/Bureau/3DNI/Projets/developpemen

for name in name_list:
  y.append(0)
  img = cv2.imread(name)
  img = tf.keras.preprocessing.image.img_to_array(img)
  img = cv2.resize(img,(128,128))
  X.append((img))
len(X)
```

Out[3]: 7841

Conversion de X et Y en deux tableaux NumPy et les redimensionner :

Cela permet de faciliter le traitement numérique, car NumPy est optimisé pour les opérations sur les tableaux.

In [4]:
```
X = np.array(X)
y = np.array(y).reshape(-1,1)
```

Affichage des informations sur les tableaux X et y après leur conversion en tableaux NumPy.

In [5]:
```
print(len(X))
print(X.shape)
print(y.shape)
```

```
7841
(7841, 128, 128, 3)
(7841, 1)
```

Division des données d'images et d'étiquettes en trois parties : un ensemble d'entraînement (90%), un ensemble de validation (2 %) et un ensemble de test(8%) .

In [6]:
```
from sklearn.model_selection import train_test_split
X_train, X_test,y_train, y_test = train_test_split(X, y, test_size=0.1, random_s
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.2, r

print( X_train.shape) #Train
print(X_val.shape)    #validation
print( X_test.shape)  #Test
```

```
(7056, 128, 128, 3)
(157, 128, 128, 3)
(628, 128, 128, 3)
```

Normalisation des images : (255 pixels)

In [7]:
```
X_train = X_train /255
X_test_scaled = X_test / 255
X_val = X_val/255
```

VGG16 : C'est une architecture de réseau de neurones convolutifs (CNN) pré-entraînée, célèbre pour ses performances en classification d'images.

In [8]:
```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(128,128,


for layer in base_model.layers:
    layer.trainable = False

model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(1024))
model.add(Activation("relu"))
model.add(Dense(512))
model.add(Activation("relu"))
model.add(Dense(1))
model.add(Activation("sigmoid"))
# Summary model
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 1024)              8389632

 activation (Activation)     (None, 1024)              0

 dense_1 (Dense)             (None, 512)               524800

 activation_1 (Activation)   (None, 512)               0

 dense_2 (Dense)             (None, 1)                 513

 activation_2 (Activation)   (None, 1)                 0

=================================================================
Total params: 23,629,633
Trainable params: 8,914,945
Non-trainable params: 14,714,688
_________________________________________________________________
```

Configuration et entraînement d'un modèle d'apprentissage automatique en utilisant TensorFlow et Keras.

In [9]:
```
from tensorflow.keras.callbacks import EarlyStopping
learning_rate = 0.00001  # taux d'apprentissage
decay_steps = 10         # le taux d'apprentissage sera ajusté toutes les 10 épo
decay_rate = 1            # le facteur par lequel le taux d'apprentissage doit ê

# Initialisation de l'ordonnanceur de taux d'apprentissage
lr_scheduler = tf.keras.optimizers.schedules.ExponentialDecay(learning_rate, dec
optimizer1 = tf.keras.optimizers.Adam(learning_rate = lr_scheduler)
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=

model.compile(optimizer = optimizer1, loss = 'binary_crossentropy',metrics =['ac
# Entraînement du modèle
history = model.fit(X_train,y_train,batch_size = 32, epochs =15,validation_data
```

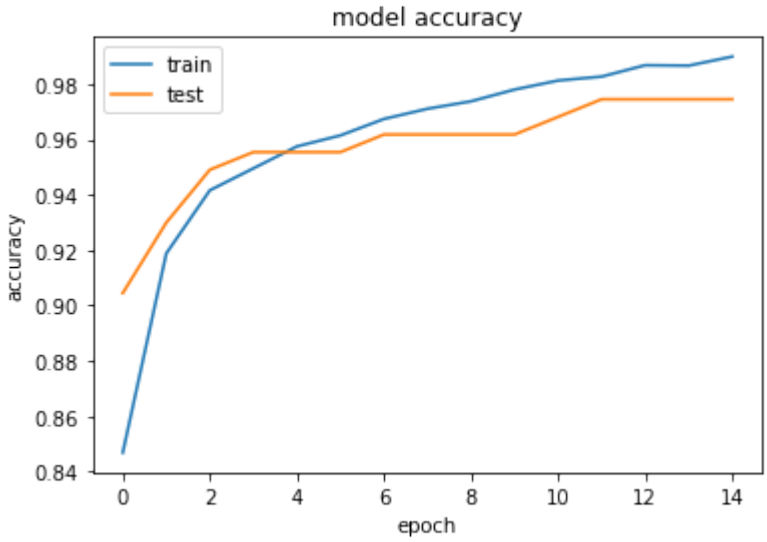
```
Epoch 1/15
221/221 [==============================] - 215s 970ms/step - loss: 0.3945 - accur
acy: 0.8468 - val_loss: 0.2785 - val_accuracy: 0.9045
Epoch 2/15
221/221 [==============================] - 216s 978ms/step - loss: 0.2273 - accur
acy: 0.9188 - val_loss: 0.2054 - val_accuracy: 0.9299
Epoch 3/15
221/221 [==============================] - 221s 998ms/step - loss: 0.1780 - accur
acy: 0.9416 - val_loss: 0.1754 - val_accuracy: 0.9490
Epoch 4/15
221/221 [==============================] - 219s 992ms/step - loss: 0.1506 - accur
acy: 0.9495 - val_loss: 0.1538 - val_accuracy: 0.9554
Epoch 5/15
221/221 [==============================] - 219s 990ms/step - loss: 0.1321 - accur
acy: 0.9575 - val_loss: 0.1412 - val_accuracy: 0.9554
Epoch 6/15
221/221 [==============================] - 223s 1s/step - loss: 0.1175 - accurac
y: 0.9615 - val_loss: 0.1315 - val_accuracy: 0.9554
Epoch 7/15
221/221 [==============================] - 215s 974ms/step - loss: 0.1040 - accur
acy: 0.9674 - val_loss: 0.1227 - val_accuracy: 0.9618
Epoch 8/15
221/221 [==============================] - 206s 934ms/step - loss: 0.0927 - accur
acy: 0.9711 - val_loss: 0.1213 - val_accuracy: 0.9618
Epoch 9/15
221/221 [==============================] - 210s 949ms/step - loss: 0.0845 - accur
acy: 0.9738 - val_loss: 0.1164 - val_accuracy: 0.9618
Epoch 10/15
221/221 [==============================] - 209s 947ms/step - loss: 0.0761 - accur
acy: 0.9780 - val_loss: 0.1071 - val_accuracy: 0.9618
Epoch 11/15
221/221 [==============================] - 206s 933ms/step - loss: 0.0679 - accur
acy: 0.9813 - val_loss: 0.1000 - val_accuracy: 0.9682
Epoch 12/15
221/221 [==============================] - 206s 931ms/step - loss: 0.0619 - accur
acy: 0.9827 - val_loss: 0.0988 - val_accuracy: 0.9745
Epoch 13/15
221/221 [==============================] - 206s 934ms/step - loss: 0.0549 - accur
acy: 0.9868 - val_loss: 0.0956 - val_accuracy: 0.9745
Epoch 14/15
221/221 [==============================] - 206s 934ms/step - loss: 0.0506 - accur
acy: 0.9867 - val_loss: 0.0916 - val_accuracy: 0.9745
Epoch 15/15
221/221 [==============================] - 207s 939ms/step - loss: 0.0452 - accur
acy: 0.9899 - val_loss: 0.0845 - val_accuracy: 0.9745
```

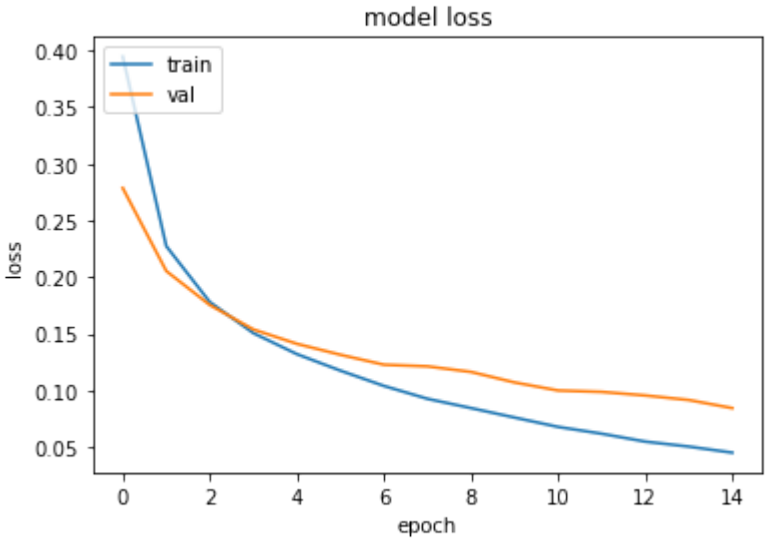
Visualisation de l'exactitude et la perte du modèle :

Remarque: Une diminution de la perte indique que le modèle apprend et s'améliore, tandis qu'une perte stagnante ou en augmentation peut indiquer un surapprentissage.

In [10]:
```
from matplotlib import pyplot  as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
```

```
plt.xlabel('epoch')
plt.legend(['train','test'],loc ='upper left')
plt.show()
#summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','val'],loc ='upper left')
plt.show()
```





Enregistrement du modèle pour l'utiliser aprés avec notre interface web

In [11]:
```
model.save('C:/Users/yosser/OneDrive/Bureau/3DNI/Projets/developpement/model1.h5
```

chargement du notre modèle de deep learning déjà entraîné et sauvegardé au format HDF5 :

In [12]:
```
model1 = load_model('C:/Users/yosser/OneDrive/Bureau/3DNI/Projets/developpement/
```

Utilisation de notre modele pour faire la prediction :

Les sorties de model1.predict() sont des valeurs comprises entre 0 et 1, représentant la probabilité que l'échantillon appartienne à la classe positive (ici, COVID-19).

Si la probabilité est supérieure ou égale à 0,5, la classe est considérée comme 1 (COVID).

Si la probabilité est inférieure à 0,5, la classe est considérée comme 0 (Normal).

In [13]:
```
y_hat = model1.predict(X_test_scaled) #  prédiction sur les données de test
def predict(y_hat):
    y_hat[y_hat >= 0.5] =1
    y_hat[y_hat<0.5] =0
    return y_hat
y_pred = predict(y_hat)             # Calcul de la précision (accuracy)
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
result= []
real_result = []
for i in y_pred:                # Conversion des prédictions en étiquettes de te
    if i==0:
        result.append('Normal')
    if i==1:
        result.append('Covid')
for i in y_pred:
    if i==0:
        real_result.append('Normal')
    if i==1:
        real_result.append('Covid')
```

```
0.9665605095541401
```

La precision est 0.9665605095541401

Utilisation des fonctions de la bibliothèque Scikit-learn pour évaluer les performances d'un modèle de machine learning.

In [14]:
```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_sc
labels = ['Covid', 'Normal']
report = classification_report(y_test, y_pred, target_names=labels)
print(report)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
```

```
              precision    recall  f1-score   support

       Covid       0.97      0.97      0.97       344
      Normal       0.96      0.96      0.96       284

    accuracy                           0.97       628
   macro avg       0.97      0.97      0.97       628
weighted avg       0.97      0.97      0.97       628

Accuracy: 0.9665605095541401
```

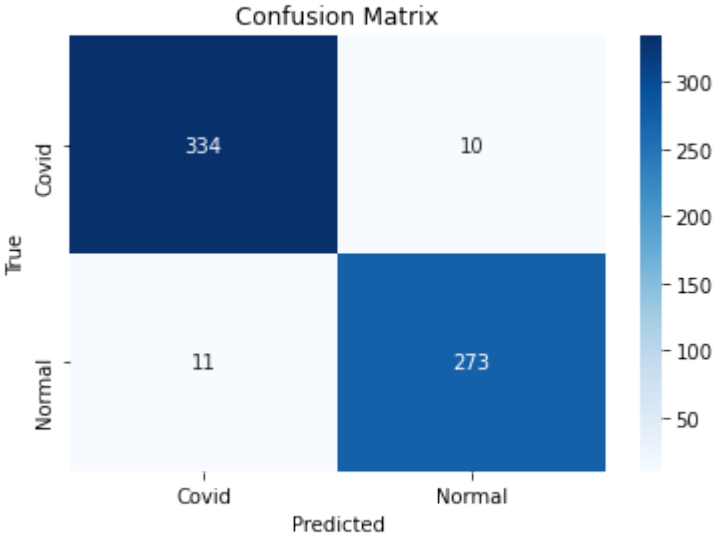
Affichage de matrice de confusion + Heatmap :

In [15]:
```
cm = confusion_matrix(y_test, y_pred)     #  Calcul de la matrice de confusion
print(cm)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues') #Affichage de heatmap


tick_labels = ['Covid', 'Normal']
plt.xticks(np.arange(len(tick_labels)) + 0.5, tick_labels)
plt.yticks(np.arange(len(tick_labels)) + 0.5, tick_labels)

plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```
[[334  10]
 [ 11 273]]
```



Affichage des images issues d'un ensemble de test, tout en affichant la prédiction du modèle et l'étiquette réelle (label) associée à chaque image.

In [16]:
```
import matplotlib.pyplot as plt

def show_image_with_prediction(image_array, prediction, label):
    plt.imshow(image_array)
    plt.axis('off')
    plt.title(f'Prediction: {prediction}, Label: {label}')
    plt.show()
```

```python
# In this case, we're only showing the first 10 images
image_arrays = X_test[:10]  # Select only the first 10 images
predictions = result[:10]    # Select the first 10 predictions
labels = real_result[:10]    # Select the first 10 actual labels

# Display 10 images with predicted and actual labels
for image_array, prediction, label in zip(image_arrays, predictions, labels):
    show_image_with_prediction(image_array / 255, prediction, label)
```

Prediction: Normal, Label: Normal

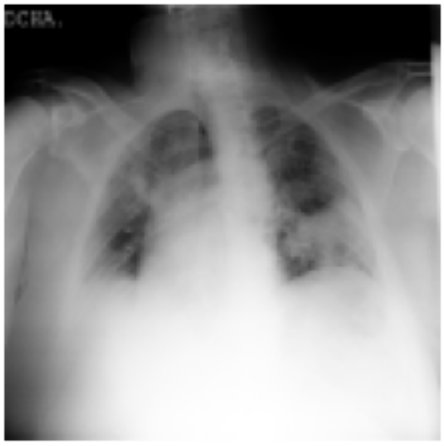
Prediction: Covid, Label: Covid

Prediction: Normal, Label: Normal

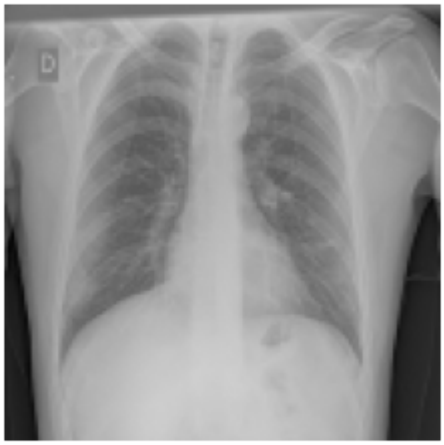
Prediction: Normal, Label: Normal

Prediction: Normal, Label: Normal

Prediction: Covid, Label: Covid



Prediction: Covid, Label: Covid



Prediction: Normal, Label: Normal

Prediction: Normal, Label: Normal

Prediction: Normal, Label: Normal

In [ ]:

In [17]:
```
name_list = glob.glob("C:/Users/yosser/OneDrive/Bureau/3DNI/Projets/developpemen

#Load the training images and labels
X_input= []
for name in name_list:
  img = cv2.imread(name)
  img = tf.keras.preprocessing.image.img_to_array(img)
  img = cv2.resize(img,(128,128))

  X_input.append((img))
```

Apres on va charger notre modèle, préparer des images, prédire leur classe, et convertir les probabilités en étiquettes.

In [18]:
```
X_input=np.array(X_input)
X_input = X_input/255
model1 = load_model('C:/Users/yosser/OneDrive/Bureau/3DNI/Projets/developpement/
y_hat = model1.predict(X_input)
y_pred = predict(y_hat)
```

In [19]:
```
result_pred =[]
for i in y_pred:
    if i==0:
        result_pred.append('Normal')
    if i==1:
        result_pred.append('Covid')

#ce code parcourt les prédictions numériques (0 ou 1) dans y_pred, et pour chaqu
#il convertit les valeurs numériques en étiquettes de classe ("Normal" ou "Covid
#À la fin de cette boucle, result_pred contiendra les étiquettes correspondantes
#le modèle.
```

Visualisation et Sauvegarde des Prédictions d'Images pour la Détection de COVID-19 :

In [20]:
```
import matplotlib.pyplot as plt
import os

def show_image_with_prediction(image_array, prediction):
    plt.imshow(image_array)
    plt.axis('off')
    plt.title(f'Prediction: {prediction}')
    plt.show()

image_arrays = X_input
predictions = result_pred
for image_array, prediction in zip(image_arrays, predictions):
    show_image_with_prediction(image_array, prediction)
output_dir = "C:/Users/yosser/OneDrive/Bureau/3DNI/Projets/developpement/Resulta
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
for i, (image_array, prediction) in enumerate(zip(image_arrays, predictions)):
    filename = f"image_{i}_prediction_{prediction}.jpg"
    file_path = os.path.join(output_dir, filename)
    plt.savefig(file_path)
```

Prediction: Covid

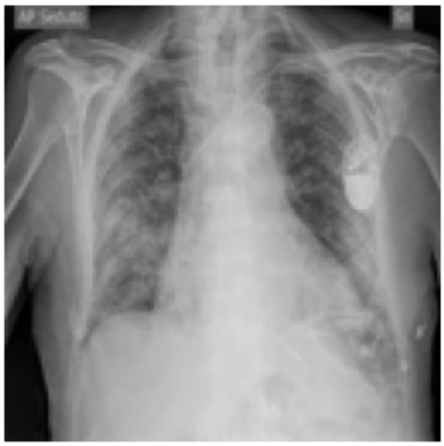
Prediction: Covid

Prediction: Covid

Prediction: Covid

Prediction: Covid

Prediction: Covid



Prediction: Covid

Prediction: Covid

Prediction: Covid

Prediction: Covid

Prediction: Normal

Prediction: Normal

Prediction: Normal

Prediction: Normal

Prediction: Normal

Prediction: Normal

Prediction: Normal

Prediction: Normal

Prediction: Normal

Prediction: Normal

<Figure size 432x288 with 0 Axes>

In [ ]: