1. Xây dựng mạng một kiến trúc mạng CNN để phân loại bệnh nhân bị viêm phổi hay không bị viêm phổi dựa trên ảnh X-Ray.

Import các thư viện

```
import os
import cv2
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import glob
import seaborn as sn
import tensorflow
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, Flatten, Input,
Reshape
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import load_model
from keras.models import Sequential
import sklearn.metrics
from sklearn.metrics import accuracy_score
import seaborn as sn
from sklearn.metrics import classification_report
from tensorflow.keras.applications import ResNet50V2
from sklearn.utils import shuffle
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.24.3
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
```

Chia 2 thư mục train và test

```
train_dir = '/kaggle/input/chest-xray-pneumonia/chest_xray/train'
test_dir = '/kaggle/input/chest-xray-pneumonia/chest_xray/test'
categories = ["NORMAL", "PNEUMONIA"]
img_size = 128

X_train_label = []
X_train_list = []
```

```
X_test_label = []
X_test_list = []
```

Load data và preprocessing

Train set

```
normal_train_list = glob.glob(train_dir+"/"+categories[0]+"/*")
pneumonia_train_list = glob.glob(train_dir+"/"+categories[1]+"/*")
for name in normal_train_list:
    X_train_label.append(0)
    img = cv2.imread(name)
#     img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    img = tensorflow.keras.preprocessing.image.img_to_array(img)
    img = cv2.resize(img,(img_size,img_size),
interpolation=cv2.INTER_AREA)
    X_train_list.append((img))
for name in pneumonia_train_list:
    X_train_label.append(1)
    img = cv2.imread(name)
#     img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    img = tensorflow.keras.preprocessing.image.img_to_array(img)
    img = cv2.resize(img,(img_size,img_size),
interpolation=cv2.INTER_AREA)
    X_train_list.append((img))
```

Test set

```
normal_test_list = glob.glob(test_dir+"/"+categories[0]+"/*")
pneumonia_test_list = glob.glob(test_dir+"/"+categories[1]+"/*")
for name in normal_test_list:
    X_test_label.append(0)
    img = cv2.imread(name)
#     img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    img = tensorflow.keras.preprocessing.image.img_to_array(img)
    img = cv2.resize(img,(img_size,img_size),
interpolation=cv2.INTER_AREA)
    X_test_list.append((img))
for name in pneumonia_test_list:
    X_test_label.append(1)
    img = cv2.imread(name)
#     img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    img = tensorflow.keras.preprocessing.image.img_to_array(img)
    img = cv2.resize(img,(img_size,img_size),
interpolation=cv2.INTER_AREA)
    X_test_list.append((img))

X_train = np.array(X_train_list, dtype = 'float32')
y_train = np.array(X_train_label, dtype = 'int32').reshape(-1,1)
```

```
X_test = np.array(X_test_list, dtype = 'float32')
y_test = np.array(X_test_label, dtype = 'int32').reshape(-1,1)
print(y_train.shape)
print(y_test.shape)
```

```
(5216, 1)
(624, 1)
```

Chuẩn hoá input, output

```
X_train_scaled = np.array(X_train)/255.
X_test_scaled = np.array(X_test)/255.
print(X_train_scaled.shape, X_test_scaled.shape)
```

```
(5216, 128, 128, 3) (624, 128, 128, 3)
```

Chia tập train thành tập train và tập validation

```
X_train_scaled, y_train = shuffle(X_train_scaled, y_train,
random_state=42)
X_train_scaled, X_val_scaled,y_train, y_val =
train_test_split(X_train_scaled, y_train, test_size=0.3)

print(X_train_scaled.shape)
print(X_val_scaled.shape)
print(X_test_scaled.shape)
```

```
(3651, 128, 128, 3)
(1565, 128, 128, 3)
(624, 128, 128, 3)
```

Xây dựng mô hình CNN

```
inp = Input(shape = (128,128,3))
cnn = Conv2D(filters = 32 , kernel_size = 3 ,activation ='relu')(inp)
cnn = BatchNormalization()(cnn)
pooling = MaxPooling2D(pool_size=(2,2))(cnn)
drop = Dropout(0.2)(pooling)

cnn = Conv2D(filters = 64, kernel_size = 3, activation='relu')(drop)
cnn = BatchNormalization()(cnn)
pooling = MaxPooling2D(pool_size = (2,2))(cnn)
drop = Dropout(0.2)(pooling)

cnn = Conv2D(filters = 128, kernel_size = 3, activation='relu')(drop)
cnn = BatchNormalization()(cnn)
pooling = MaxPooling2D(pool_size = (2,2))(cnn)
drop = Dropout(0.2)(pooling)
```

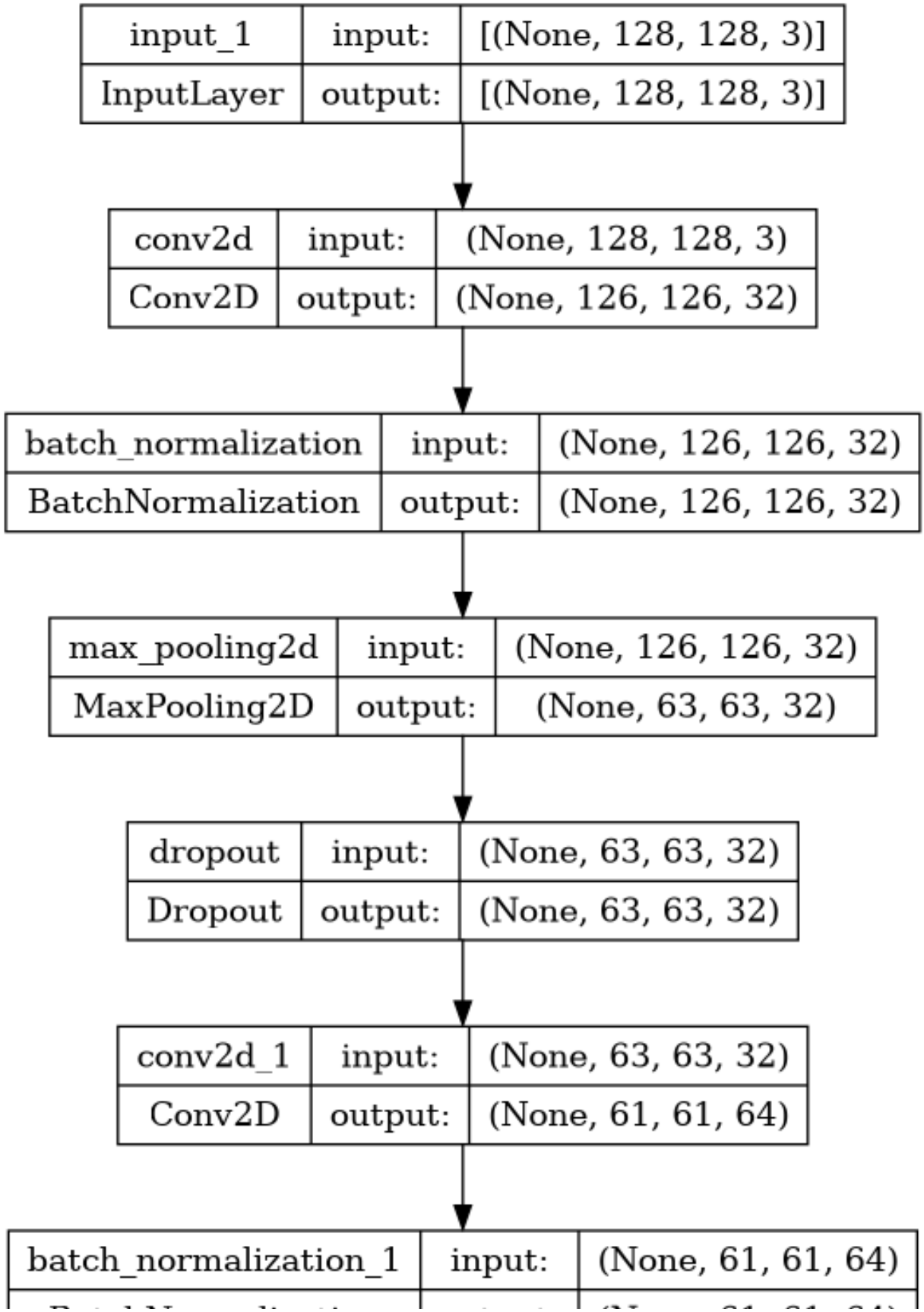
```
f = Flatten()(drop)

fc1 = Dense(units = 256, activation ='relu')(f)
fc2  = Dense(units = 16,activation = 'relu')(fc1)
out = Dense(units=1,activation='sigmoid')(fc2)

model = Model(inputs = inp, outputs =out)
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 128, 128, 3)]     0

 conv2d (Conv2D)             (None, 126, 126, 32)      896

 batch_normalization (Batch  (None, 126, 126, 32)      128
 Normalization)

 max_pooling2d (MaxPooling2  (None, 63, 63, 32)        0
 D)

 dropout (Dropout)           (None, 63, 63, 32)        0

 conv2d_1 (Conv2D)           (None, 61, 61, 64)        18496

 batch_normalization_1 (Bat  (None, 61, 61, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 30, 30, 64)        0
 g2D)

 dropout_1 (Dropout)         (None, 30, 30, 64)        0

 conv2d_2 (Conv2D)           (None, 28, 28, 128)       73856

 batch_normalization_2 (Bat  (None, 28, 28, 128)       512
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 14, 14, 128)       0
 g2D)

 dropout_2 (Dropout)         (None, 14, 14, 128)       0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 256)               6422784

 dense_1 (Dense)             (None, 16)                4112
```

```
 dense_2 (Dense)             (None, 1)                 17

=================================================================
Total params: 6521057 (24.88 MB)
Trainable params: 6520609 (24.87 MB)
Non-trainable params: 448 (1.75 KB)
_________________________________________________________________
```

Trực quan hoá mô hình

```
tensorflow.keras.utils.plot_model(model, show_layer_names=True,
show_shapes=True)
```


input_1 | input: | [(None, 128, 128, 3)]
InputLayer | output: | [(None, 128, 128, 3)]
conv2d | input: | (None, 128, 128, 3)
Conv2D | output: | (None, 126, 126, 32)
batch_normalization | input: | (None, 126, 126, 32)
BatchNormalization | output: | (None, 126, 126, 32)
max_pooling2d | input: | (None, 126, 126, 32)
MaxPooling2D | output: | (None, 63, 63, 32)
dropout | input: | (None, 63, 63, 32)
Dropout | output: | (None, 63, 63, 32)
conv2d_1 | input: | (None, 63, 63, 32)
Conv2D | output: | (None, 61, 61, 64)
batch_normalization_1 | input: | (None, 61, 61, 64)

Tối ưu hoá bằng Adam, huấn luyện mô hình

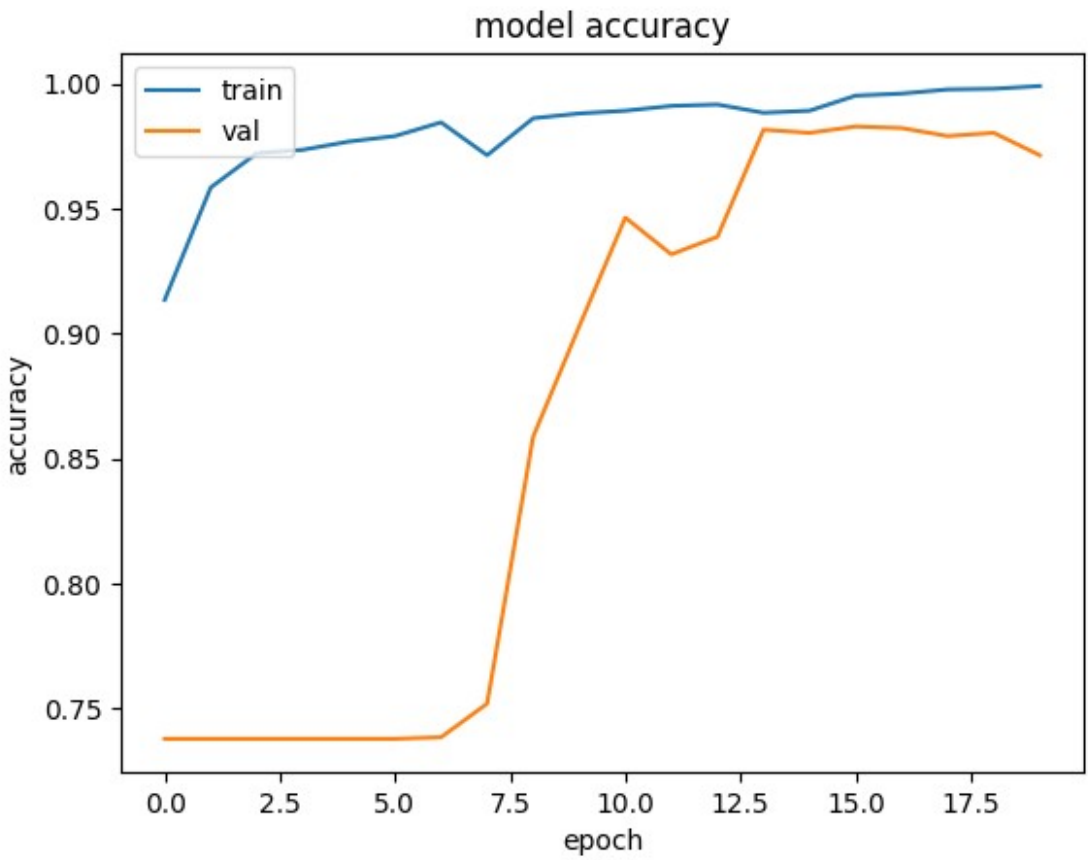
```
optimizer1 = tensorflow.keras.optimizers.Adam(learning_rate = 0.0001)
model.compile(optimizer = optimizer1,
loss='binary_crossentropy',metrics = ['accuracy'])
history = model.fit(X_train_scaled,y_train,batch_size=64,
epochs = 20, validation_data = (X_val_scaled, y_val))
```

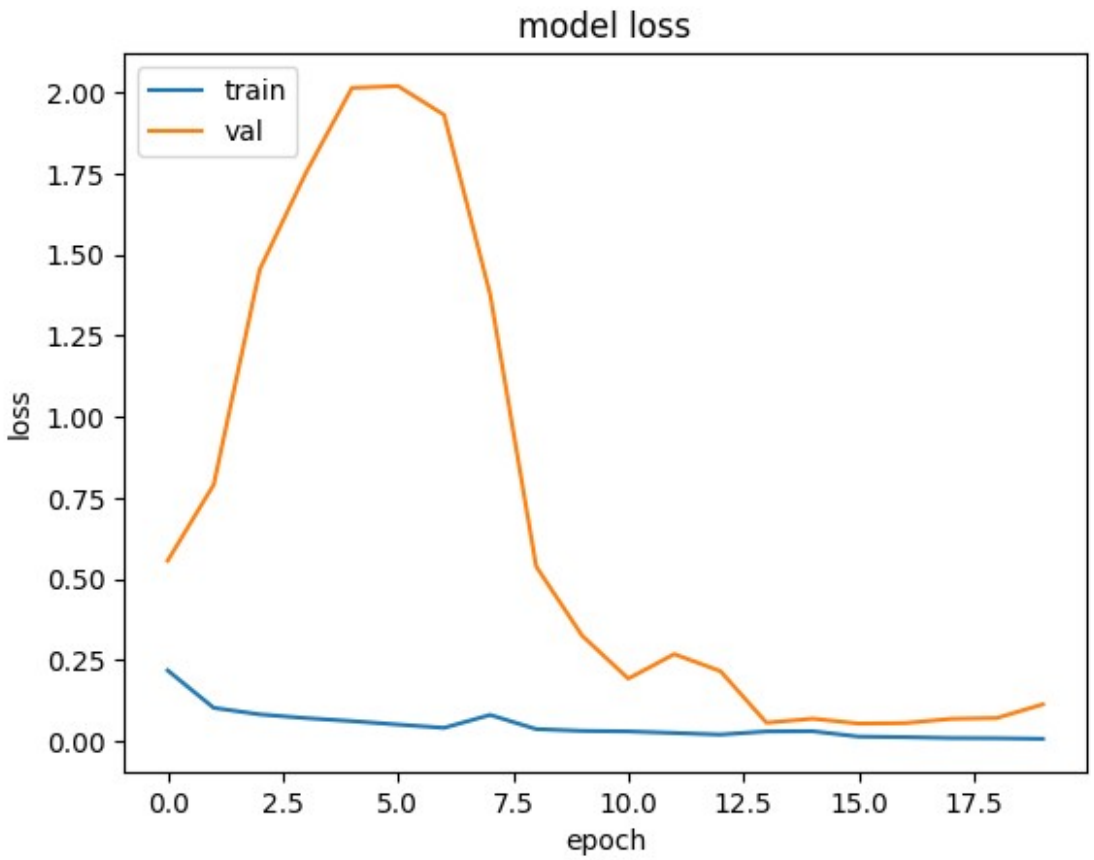
```
Epoch 1/20

2023-11-16 17:15:06.284333: E
tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] layout
failed: INVALID_ARGUMENT: Size of values 0 does not match size of
permutation 4 @ fanin shape inmodel/dropout/dropout/SelectV2-2-
TransposeNHWCToNCHW-LayoutOptimizer

58/58 [==============================] - 13s 62ms/step - loss: 0.2172
- accuracy: 0.9134 - val_loss: 0.5558 - val_accuracy: 0.7380
Epoch 2/20
58/58 [==============================] - 2s 43ms/step - loss: 0.1018 -
accuracy: 0.9584 - val_loss: 0.7905 - val_accuracy: 0.7380
Epoch 3/20
58/58 [==============================] - 2s 43ms/step - loss: 0.0816 -
accuracy: 0.9721 - val_loss: 1.4548 - val_accuracy: 0.7380
Epoch 4/20
58/58 [==============================] - 2s 43ms/step - loss: 0.0702 -
accuracy: 0.9734 - val_loss: 1.7522 - val_accuracy: 0.7380
Epoch 5/20
58/58 [==============================] - 2s 43ms/step - loss: 0.0610 -
accuracy: 0.9767 - val_loss: 2.0145 - val_accuracy: 0.7380
Epoch 6/20
58/58 [==============================] - 2s 43ms/step - loss: 0.0503 -
accuracy: 0.9789 - val_loss: 2.0202 - val_accuracy: 0.7380
Epoch 7/20
58/58 [==============================] - 2s 43ms/step - loss: 0.0401 -
accuracy: 0.9844 - val_loss: 1.9315 - val_accuracy: 0.7387
Epoch 8/20
58/58 [==============================] - 2s 43ms/step - loss: 0.0800 -
accuracy: 0.9712 - val_loss: 1.3778 - val_accuracy: 0.7521
Epoch 9/20
58/58 [==============================] - 2s 43ms/step - loss: 0.0363 -
accuracy: 0.9860 - val_loss: 0.5379 - val_accuracy: 0.8588
Epoch 10/20
58/58 [==============================] - 2s 42ms/step - loss: 0.0307 -
accuracy: 0.9879 - val_loss: 0.3241 - val_accuracy: 0.9029
Epoch 11/20
58/58 [==============================] - 2s 42ms/step - loss: 0.0292 -
accuracy: 0.9890 - val_loss: 0.1920 - val_accuracy: 0.9463
Epoch 12/20
58/58 [==============================] - 2s 42ms/step - loss: 0.0243 -
```

```
accuracy: 0.9910 - val_loss: 0.2676 - val_accuracy: 0.9316
Epoch 13/20
58/58 [==============================] - 2s 42ms/step - loss: 0.0194 -
accuracy: 0.9915 - val_loss: 0.2151 - val_accuracy: 0.9387
Epoch 14/20
58/58 [==============================] - 2s 42ms/step - loss: 0.0291 -
accuracy: 0.9882 - val_loss: 0.0558 - val_accuracy: 0.9815
Epoch 15/20
58/58 [==============================] - 2s 42ms/step - loss: 0.0297 -
accuracy: 0.9890 - val_loss: 0.0684 - val_accuracy: 0.9802
Epoch 16/20
58/58 [==============================] - 2s 43ms/step - loss: 0.0132 -
accuracy: 0.9951 - val_loss: 0.0534 - val_accuracy: 0.9827
Epoch 17/20
58/58 [==============================] - 2s 43ms/step - loss: 0.0114 -
accuracy: 0.9959 - val_loss: 0.0547 - val_accuracy: 0.9821
Epoch 18/20
58/58 [==============================] - 2s 42ms/step - loss: 0.0087 -
accuracy: 0.9975 - val_loss: 0.0678 - val_accuracy: 0.9789
Epoch 19/20
58/58 [==============================] - 2s 42ms/step - loss: 0.0081 -
accuracy: 0.9978 - val_loss: 0.0704 - val_accuracy: 0.9802
Epoch 20/20
58/58 [==============================] - 2s 42ms/step - loss: 0.0062 -
accuracy: 0.9989 - val_loss: 0.1127 - val_accuracy: 0.9712
```

2. Vẽ biểu đồ thể hiện kết quả quá trình huấn luyện mô hình.

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','val'],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()
```


model accuracy
train
val
1.00
0.95
0.90
0.85
0.80
0.75
accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epoch



Lưu mô hình, load mô hình đã lưu từ máy

```
model.save('model1.h5')
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/engine/
training.py:3000: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

```
model1 = load_model('/kaggle/working/model1.h5')
```

Tính y dự đoán từ mô hình đã lưu

```
y_hat = model1.predict(X_test_scaled)
```

```
20/20 [==============================] - 0s 8ms/step
```

Threshold binary của y dự đoán

```
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)
```

3. Đánh giá kết quả của mô hình thu được.

Precision, recall, F1-score

```
target_names = ['NORMAL', 'PNEUMONIA']
print(classification_report(y_test, y_pred,target_names=target_names))
```

```
              precision    recall  f1-score   support

      NORMAL       0.99      0.33      0.50       234
   PNEUMONIA       0.71      1.00      0.83       390

    accuracy                           0.75       624
   macro avg       0.85      0.67      0.67       624
weighted avg       0.82      0.75      0.71       624
```

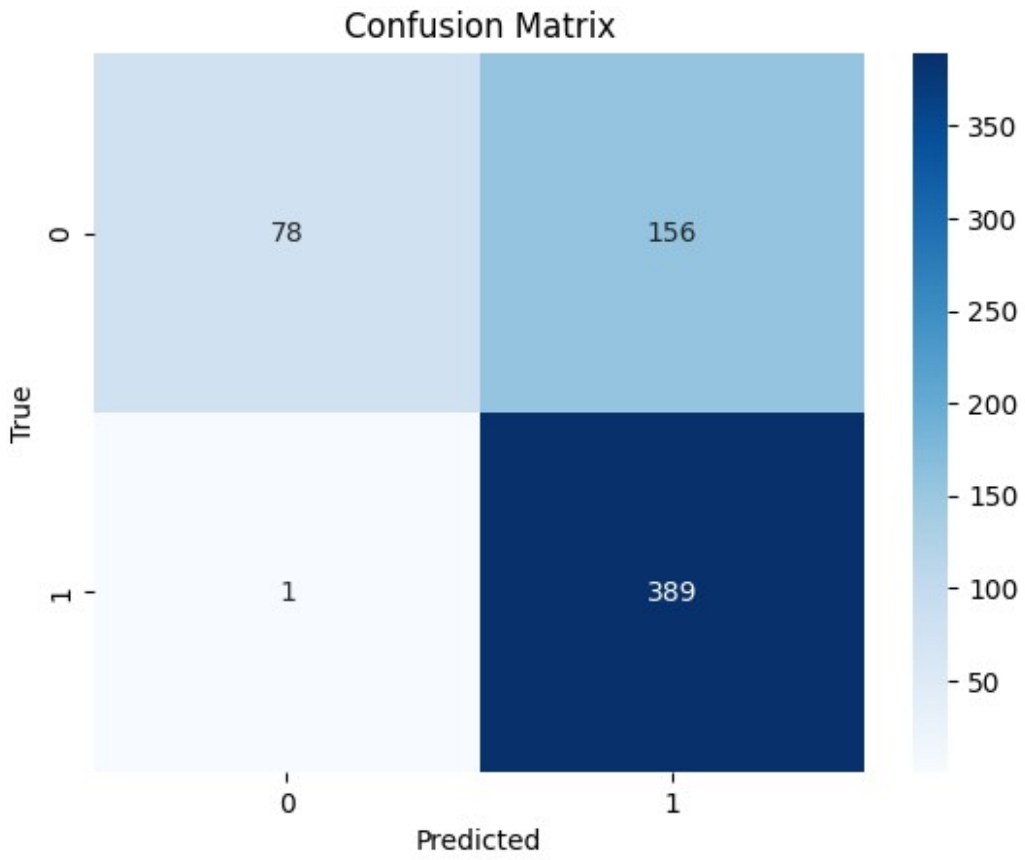
Accuracy

```
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.7483974358974359
```

Confusion matrix

```
# Tạo confusion matrix
cm = sklearn.metrics.confusion_matrix(y_test, y_pred)

# Vẽ confusion matrix
plt.figure()
sn.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

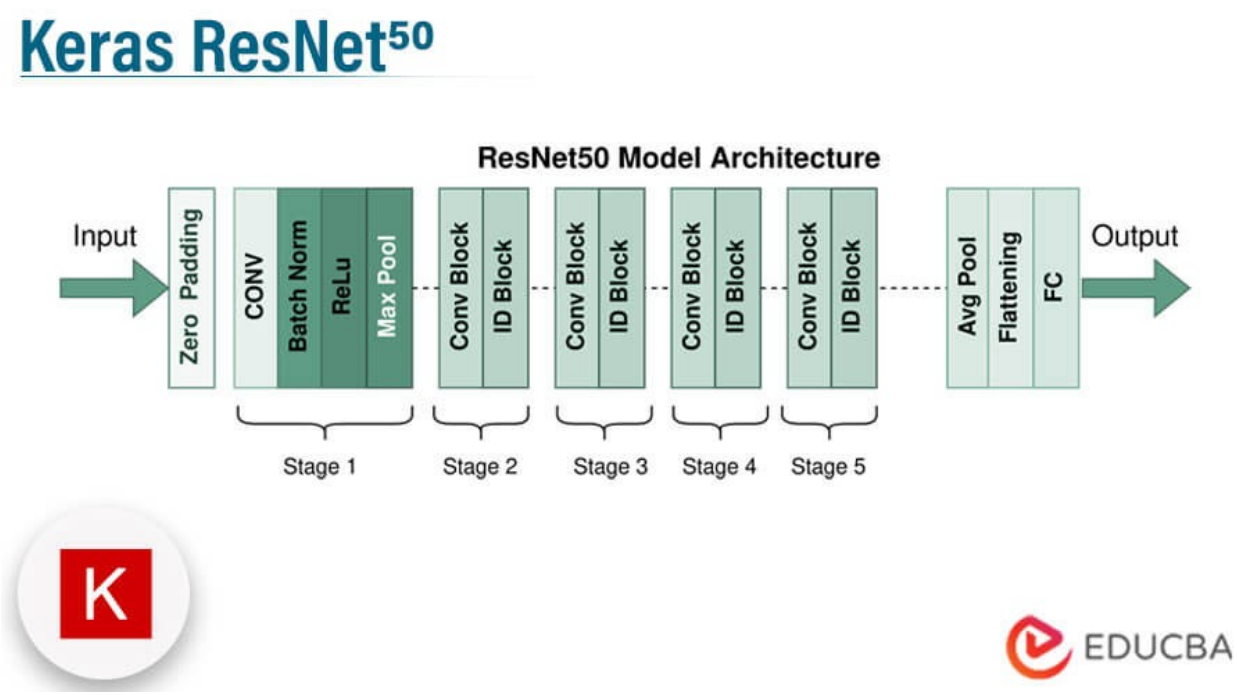


4. Tìm hiểu một mạng kiến trúc mạng Pre-train model để giải quyết bài toán này (Ví dụ: ResNet50).

Chuẩn bị input, output

```
X_train = np.array(X_train_list, dtype = 'float32')
y_train = np.array(X_train_label, dtype = 'int32').reshape(-1,1)
X_test = np.array(X_test_list, dtype = 'float32')
y_test1 = np.array(X_test_label, dtype = 'int32').reshape(-1,1)

X_train_scaled1 = np.array(X_train)/255.
X_test_scaled1 = np.array(X_test)/255.

X_train_scaled1, y_train = shuffle(X_train_scaled1, y_train,
random_state=42)
X_train_scaled1, X_val_scaled1, y_train, y_val =
train_test_split(X_train_scaled1, y_train, test_size=0.3)
```

Kiến trúc của mạng ResNet50:



Kết nối ResNet50

```
resnet50 = ResNet50V2(weights = "imagenet", input_shape = (128,128,3), include_top = False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5
94668760/94668760 [==============================] - 6s 0us/step
```

Xây dựng mô hình với ResNet50

```
model = Sequential()

model.add(resnet50)

for layer in resnet50.layers:
    layer.trainable = False

model.add(Flatten())

model.add(Dense(units = 128, activation = "relu"))
model.add(Dense(units = 16, activation = "relu"))
model.add(Dropout(0.5))

model.add(Dense(units = 1, activation = "sigmoid"))
```

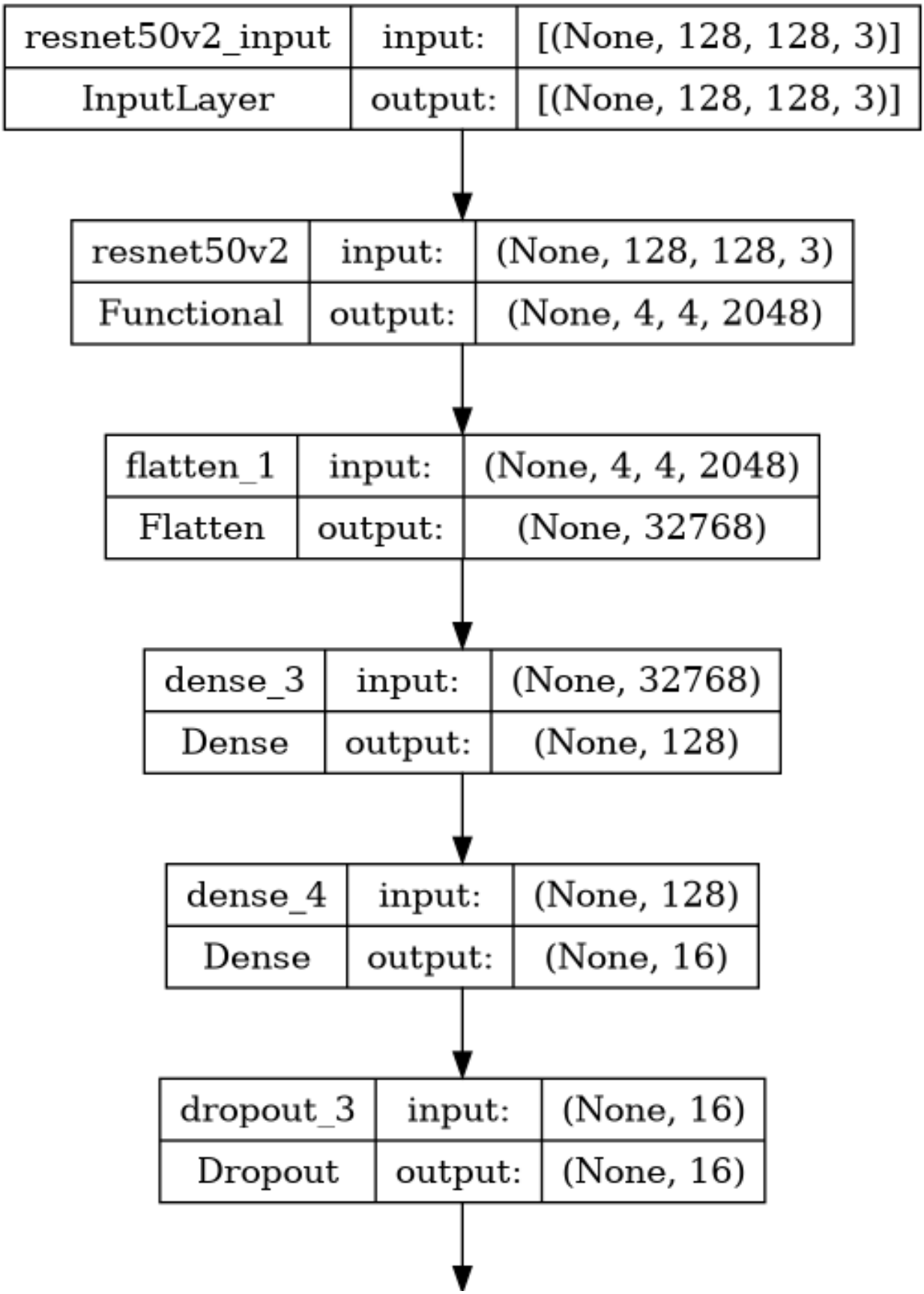
Tối ưu hoá bằng Adam

```
optimizer1 = tensorflow.keras.optimizers.Adam(learning_rate = 0.005)
model.compile(optimizer = optimizer1, loss = "binary_crossentropy",
metrics = ["accuracy"])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50v2 (Functional)     (None, 4, 4, 2048)        23564800

 flatten_1 (Flatten)         (None, 32768)             0

 dense_3 (Dense)             (None, 128)               4194432

 dense_4 (Dense)             (None, 16)                2064

 dropout_3 (Dropout)         (None, 16)                0

 dense_5 (Dense)             (None, 1)                 17

=================================================================
Total params: 27761313 (105.90 MB)
Trainable params: 4196513 (16.01 MB)
Non-trainable params: 23564800 (89.89 MB)
_________________________________________________________________
```

Trực quan hoá mô hình pre-train

```
tensorflow.keras.utils.plot_model(model, show_layer_names=True,
show_shapes=True)
```


resnet50v2_input | input: | [(None, 128, 128, 3)]
InputLayer | output: | [(None, 128, 128, 3)]
resnet50v2 | input: | (None, 128, 128, 3)
Functional | output: | (None, 4, 4, 2048)
flatten_1 | input: | (None, 4, 4, 2048)
Flatten | output: | (None, 32768)
dense_3 | input: | (None, 32768)
Dense | output: | (None, 128)
dense_4 | input: | (None, 128)
Dense | output: | (None, 16)
dropout_3 | input: | (None, 16)
Dropout | output: | (None, 16)

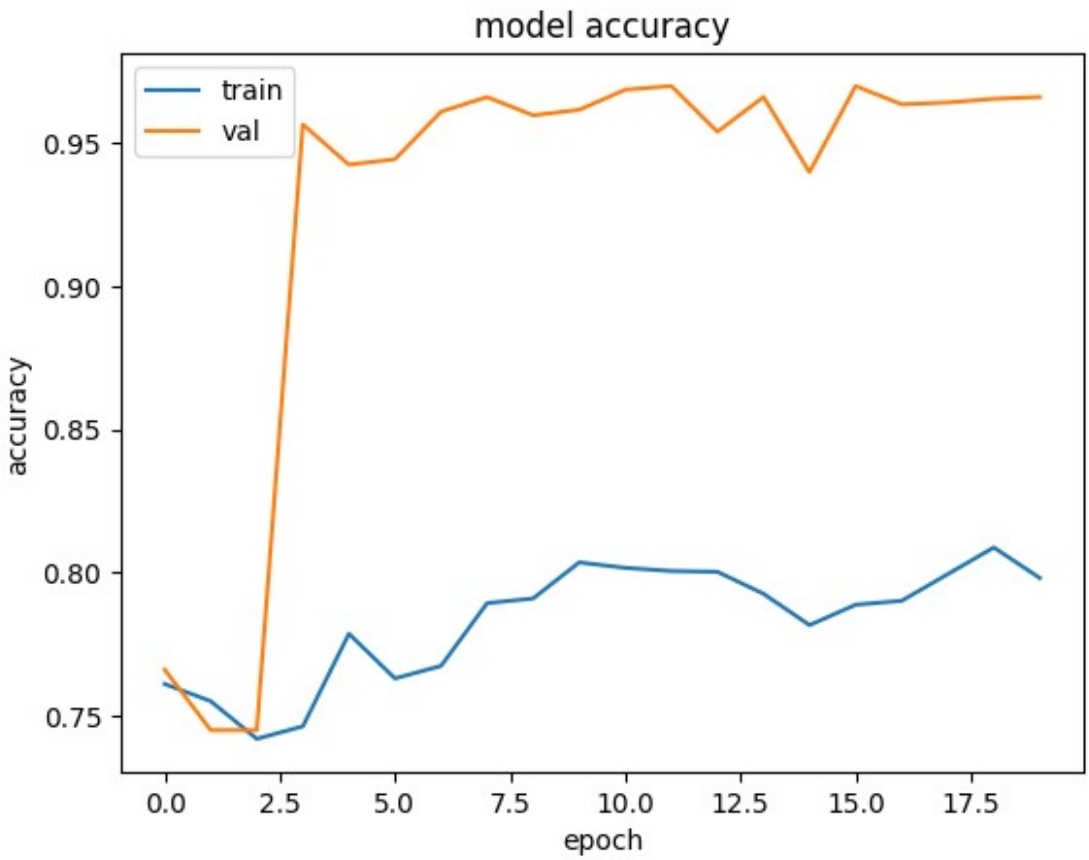
Huấn luyện mô hình

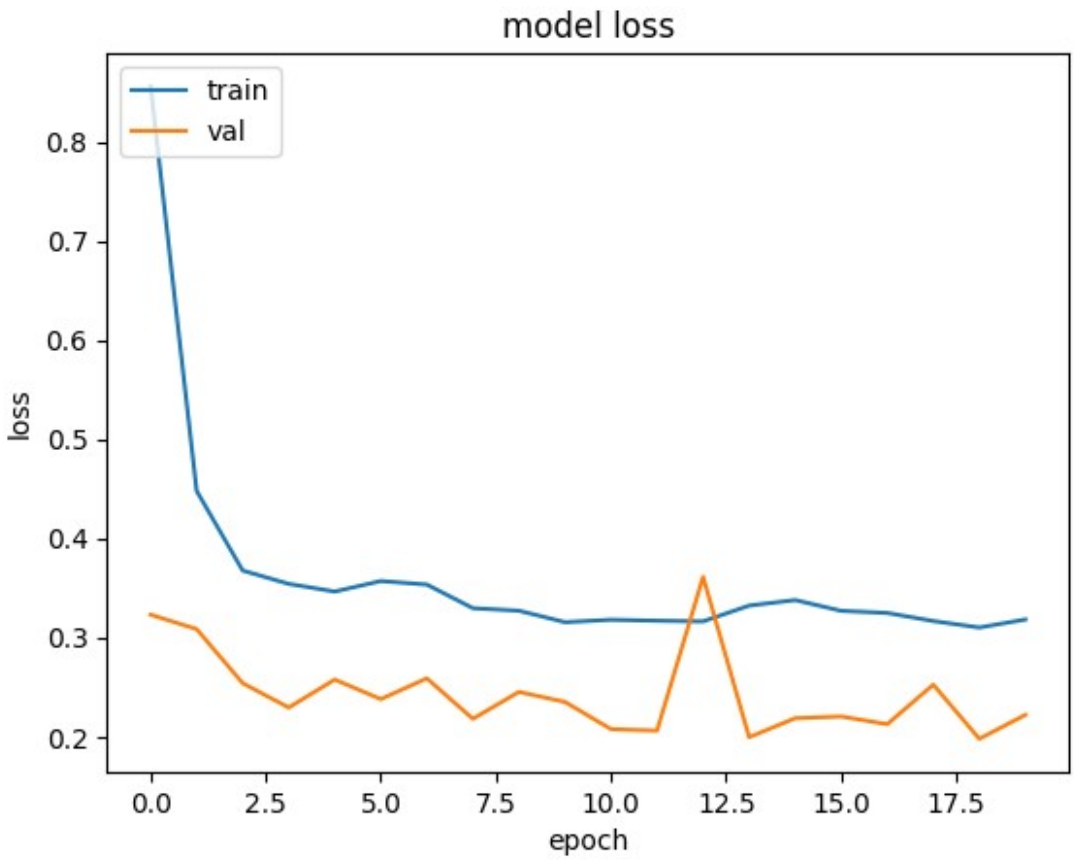
```
history = model.fit(X_train_scaled1,y_train,batch_size=64,
epochs = 20, validation_data = (X_val_scaled1, y_val))

Epoch 1/20
58/58 [==============================] - 11s 102ms/step - loss: 0.8561
- accuracy: 0.7612 - val_loss: 0.3230 - val_accuracy: 0.7661
Epoch 2/20
58/58 [==============================] - 3s 59ms/step - loss: 0.4478 -
accuracy: 0.7551 - val_loss: 0.3085 - val_accuracy: 0.7450
Epoch 3/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3675 -
accuracy: 0.7420 - val_loss: 0.2539 - val_accuracy: 0.7450
Epoch 4/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3541 -
accuracy: 0.7464 - val_loss: 0.2293 - val_accuracy: 0.9565
Epoch 5/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3464 -
accuracy: 0.7787 - val_loss: 0.2576 - val_accuracy: 0.9425
Epoch 6/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3569 -
accuracy: 0.7631 - val_loss: 0.2378 - val_accuracy: 0.9444
Epoch 7/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3534 -
accuracy: 0.7675 - val_loss: 0.2588 - val_accuracy: 0.9610
Epoch 8/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3295 -
accuracy: 0.7894 - val_loss: 0.2180 - val_accuracy: 0.9661
Epoch 9/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3270 -
accuracy: 0.7910 - val_loss: 0.2451 - val_accuracy: 0.9597
Epoch 10/20
58/58 [==============================] - 3s 60ms/step - loss: 0.3154 -
accuracy: 0.8036 - val_loss: 0.2350 - val_accuracy: 0.9617
Epoch 11/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3179 -
accuracy: 0.8017 - val_loss: 0.2076 - val_accuracy: 0.9687
Epoch 12/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3169 -
accuracy: 0.8006 - val_loss: 0.2061 - val_accuracy: 0.9700
Epoch 13/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3163 -
accuracy: 0.8003 - val_loss: 0.3611 - val_accuracy: 0.9540
Epoch 14/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3322 -
accuracy: 0.7927 - val_loss: 0.1995 - val_accuracy: 0.9661
Epoch 15/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3377 -
accuracy: 0.7817 - val_loss: 0.2186 - val_accuracy: 0.9399
```

```
Epoch 16/20
58/58 [==============================] - 3s 60ms/step - loss: 0.3269 -
accuracy: 0.7888 - val_loss: 0.2204 - val_accuracy: 0.9700
Epoch 17/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3248 -
accuracy: 0.7902 - val_loss: 0.2125 - val_accuracy: 0.9636
Epoch 18/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3166 -
accuracy: 0.7995 - val_loss: 0.2525 - val_accuracy: 0.9642
Epoch 19/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3101 -
accuracy: 0.8088 - val_loss: 0.1978 - val_accuracy: 0.9655
Epoch 20/20
58/58 [==============================] - 3s 59ms/step - loss: 0.3181 -
accuracy: 0.7981 - val_loss: 0.2220 - val_accuracy: 0.9661
```

Vẽ đồ thị accuracy và loss vừa train

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','val'],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()
```


model accuracy
train
val
accuracy
epoch
0.95
0.90
0.85
0.80
0.75
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5



Lưu mô hình, load mô hình ResNet50 vừa train

```
model.save('model2.h5')
model2 = load_model('/kaggle/working/model2.h5')
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/engine/
training.py:3000: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

Tính y dự đoán từ mô hình đã lưu

```
y_hat = model2.predict(X_test_scaled1)
```

```
20/20 [==============================] - 2s 46ms/step
```

Threshold binary y dự đoán

```
def predict(y_hat):
    y_hat[y_hat >= 0.5] =1
    y_hat[y_hat < 0.5] =0
```

```
    return y_hat
y_pred1 = predict(y_hat)
```

Đánh giá mô hình pre-train

Precision. recall, F1-score

```
from sklearn.metrics import classification_report
target_names = ['NORMAL', 'PNEUMONIA']
print(classification_report(y_test1,
y_pred1,target_names=target_names))
```

```
              precision    recall  f1-score   support

      NORMAL       0.94      0.53      0.68       234
   PNEUMONIA       0.78      0.98      0.87       390

    accuracy                           0.81       624
   macro avg       0.86      0.75      0.77       624
weighted avg       0.84      0.81      0.80       624
```

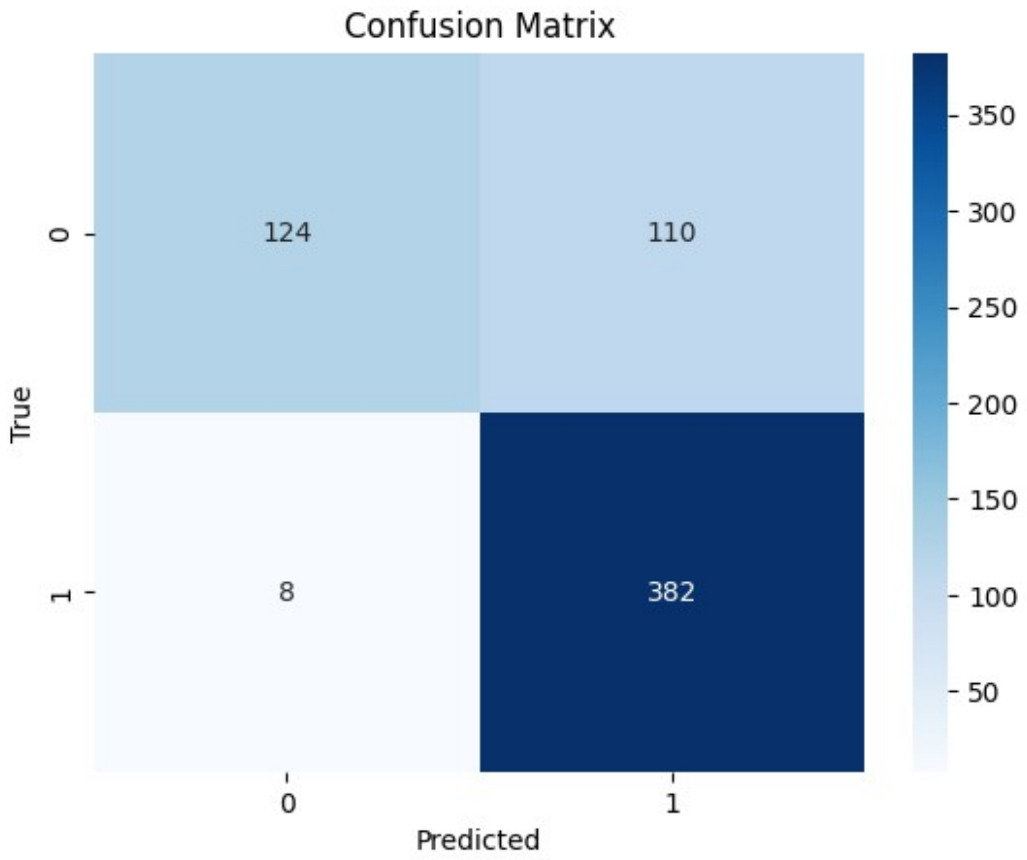
Accuracy

```
from sklearn.metrics import accuracy_score
accuracy1 = accuracy_score(y_test1, y_pred1)
print('Accuracy:', accuracy1)
```

```
Accuracy: 0.8108974358974359
```

Confusion matrix

```
import sklearn.metrics
import seaborn as sn

# Tạo confusion matrix
cm = sklearn.metrics.confusion_matrix(y_test1, y_pred1)

# Vẽ confusion matrix
plt.figure()
sn.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



So sánh độ chính xác của 2 mô hình

```
print('Accuracy của mô hình CNN tự xây dựng: ', accuracy)
print('Accuracy của mô hình pre-train ResNet50: ', accuracy1)
```

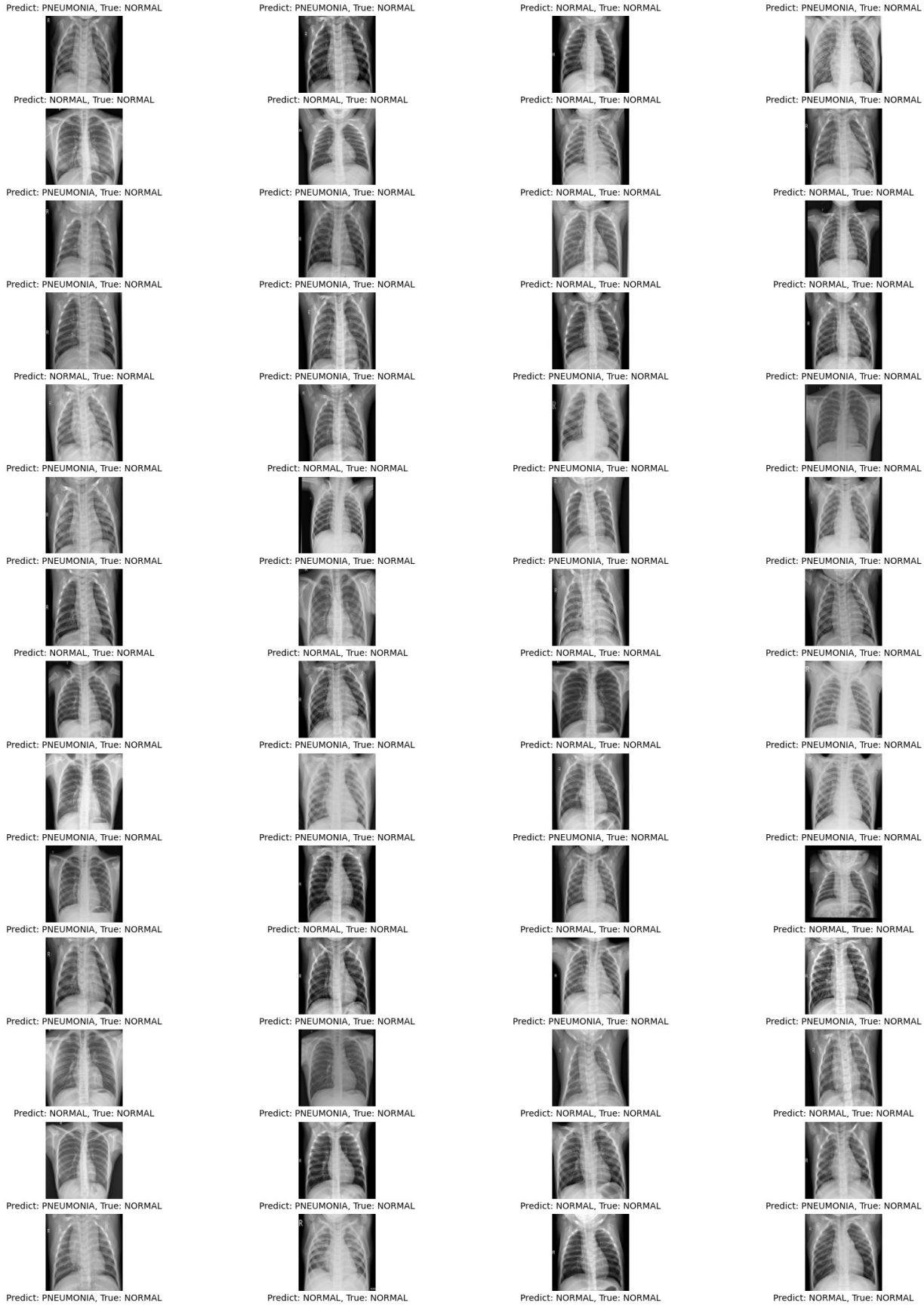
```
Accuracy của mô hình CNN tự xây dựng:  0.7483974358974359
Accuracy của mô hình pre-train ResNet50:  0.8108974358974359
```

5. So sánh kết quả thu được của 2 mô hình (1 & 4).

Kết quả của mô hình 1

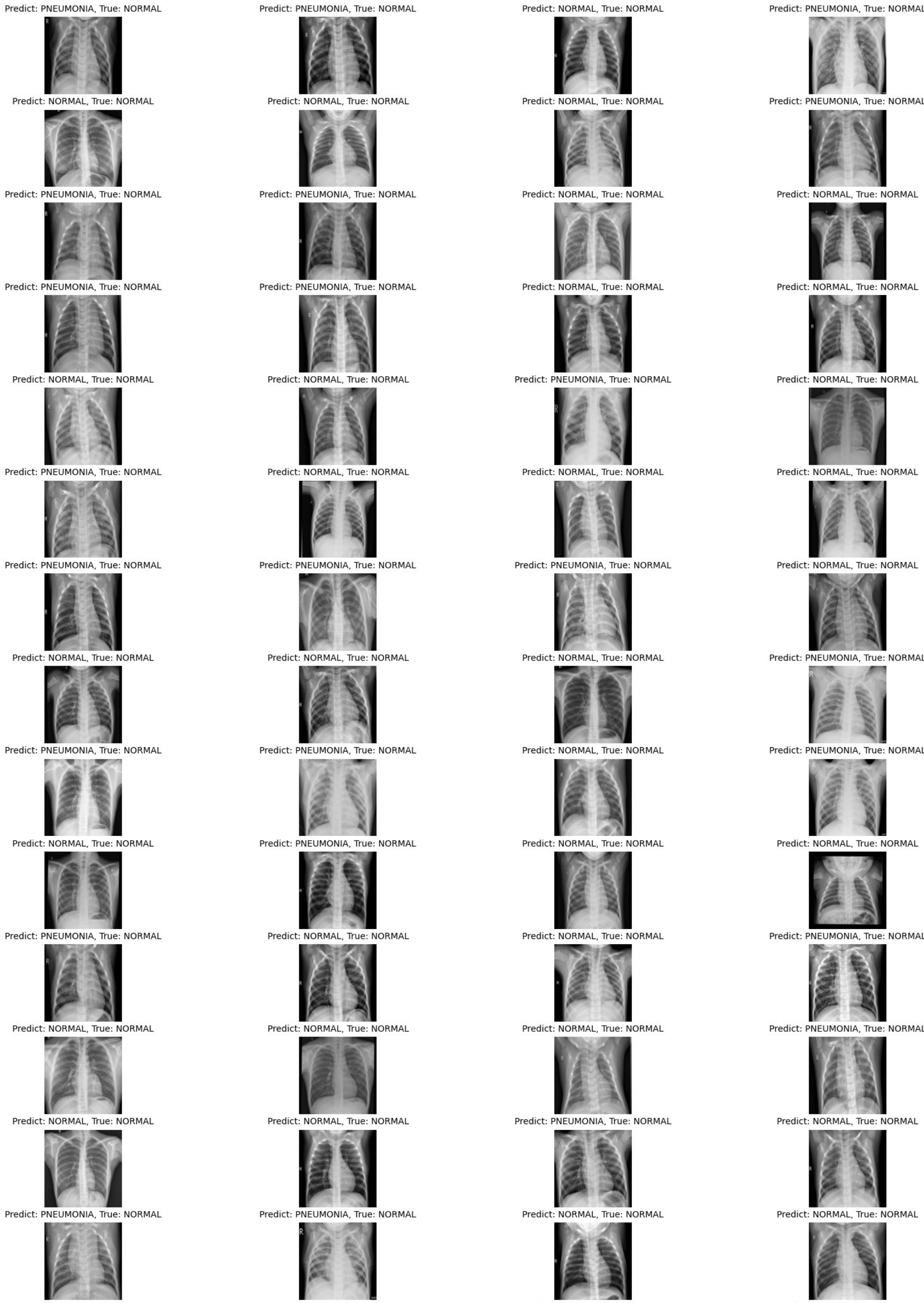
```
# Check with some set
plt.figure(figsize=(20,300))
for i in range(len(X_test_scaled)):
    plt.subplot(int(len(X_test_scaled)/4)+1,4,i+1)
    plt.title(f"Predict: {target_names[int(y_pred[i])]}, True:
{target_names[int(y_test[i])]}", fontsize=10)
    plt.imshow(X_test_scaled[i])
```

```
    plt.axis("off")
plt.show()
```


Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL

Kết quả của mô hình 4

```
# Check with some set
plt.figure(figsize=(20,300))
for i in range(len(X_test_scaled1)):
    plt.subplot(int(len(X_test_scaled1)/4)+1,4,i+1)
    plt.title(f"Predict: {target_names[int(y_pred1[i])]}, True:
{target_names[int(y_test1[i])]}", fontsize=10)
    plt.imshow(X_test_scaled1[i])
    plt.axis("off")
plt.show()
```


Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: PNEUMONIA, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL
Predict: NORMAL, True: NORMAL