pneumonia_classification____final (2)

October 4, 2022

1 M507 Methods of Prediction

1.1 Predicting pneumonia using CNN

1.1.1 GH1017889

2 Business Problem Undestanding

2.1 Introduction

Pneumonia, or lung infection, is a viral disease that affects lung tissue and interferes with the normal oxygen exchange between air and blood. Inflammatory secretions that enter the air sacs of the lungs do not allow the body to properly supply oxygen. And when the disease grabs most of the lungs, acute respiratory failure develops. Pneumonia affects people with weak immunity, children and the elderly. In Germany alone, 1.5 million people get the disease each year, 30% of whom are infants and people over the age of 70. But pneumonia can be treated and needs to be treated! And it's a good idea to do this with an integrated approach. Treatment of the disease begins with a correct diagnosis. Therefore, seek expert advice at the first signs of pathological development.Learn what the first symptoms of pneumonia are, how to properly and comprehensively treat them, and what you need to do to diagnose the disease. ## What is the aim of this notebook? I was hired by a "Medtronic" as a Data Scientist. Medtronic is a company that produces medical equipment that ease the life of doctors by digital solutions. In July we start to develop a x-ray machine that will be able to detect pneumonia. Now we are trying to develop a machine that makes the x-ray and gives a diagnosis at the same time. By doing so we are planning to eliminate a human factor.

3 Data Collection

3.1 Importing the libraries

At first lets import libraries that we will need during our pipeline creation process.

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import os
from tensorflow.keras.metrics import Recall, Precision
```

```
import glob
from sklearn.utils import class_weight
from tensorflow.keras.layers import Input, Conv2D, Dense
from tensorflow.keras.layers import GlobalAveragePooling2D, MaxPool2D, Dropout
from tensorflow.keras.models import Model
import tensorflow.keras.backend as K
from keras.layers.advanced_activations import LeakyReLU
from keras.layers import Flatten
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from keras.applications.vgg19 import VGG19
import opendatasets as od
from livelossplot import PlotLossesKeras
```

3.2 Loading the Dataset

[3]:
```
!pip install opendatasets
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: opendatasets in /usr/local/lib/python3.7/dist-
packages (0.1.22)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages
(from opendatasets) (4.64.0)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages
(from opendatasets) (7.1.2)
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages
(from opendatasets) (1.5.12)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-
packages (from kaggle->opendatasets) (2.23.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-
packages (from kaggle->opendatasets) (1.15.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-
packages (from kaggle->opendatasets) (6.1.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-
packages (from kaggle->opendatasets) (2.8.2)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages
(from kaggle->opendatasets) (1.24.3)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages
(from kaggle->opendatasets) (2022.6.15)
Requirement already satisfied: text-unidecode>=1.3 in
/usr/local/lib/python3.7/dist-packages (from python-
slugify->kaggle->opendatasets) (1.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests->kaggle->opendatasets) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from requests->kaggle->opendatasets)
```

```
(3.0.4)
```

[4]:
```
od.download("https://www.kaggle.com/datasets/paultimothymooney/
↪chest-xray-pneumonia/download")
```

```
Skipping, found downloaded files in "./chest-xray-pneumonia" (use force=True to
force download)
```

3.3 Setting Path to directories

3.3.1 Setting the Parameters

[5]:
```
BATCH_SIZE = 64
IMAGE_SIZE = (180, 180)
EPOCHS = 100
```

[6]:
```
dataset_directory = "/content/chest-xray-pneumonia/chest_xray"
```

[7]:
```
train_files = glob.glob(os.path.join(dataset_directory, "train/*/*"))
val_files = glob.glob(os.path.join(dataset_directory, "val/*/*"))
test_file = glob.glob(os.path.join(dataset_directory, "test/*/*"))
```

[8]:
```
TRAIN_DIRECTORY = os.path.join(dataset_directory, "train")
VALIDATION_DIRECTORY = os.path.join(dataset_directory, "val")
TEST_DIRECTORY = os.path.join(dataset_directory, "test")
```

[9]:
```
len(train_files), len(val_files), len(test_file)
```

[9]:
```
(5216, 16, 624)
```

[10]:
```
COUNT_NORMAL = len([filename for filename in train_files if "NORMAL" in␣
↪filename])
print("Normal images count in training set: " + str(COUNT_NORMAL))

COUNT_PNEUMONIA = len([filename for filename in train_files if "PNEUMONIA" in␣
↪filename])
print("Pneumonia images count in training set: " + str(COUNT_PNEUMONIA))
```

```
Normal images count in training set: 1341
Pneumonia images count in training set: 3875
```

After applying count function on our train dataset, we can see numbers of each case.

As we can notice the number of pneumonia cases are almost 3 times more than normal cases. It is obvious that our value distribution is imbalanced.

3.4 Data Overview

This dataset is an open source, and it can be directly accessed via Kaggle. It was uploaded by Paul Monley and contains fluorography images of patients that were recorded in different times. It contains 3 folders that represents 'training' , 'test' and 'validation' datasets. All 3 folders contain subfolders with 'normal' and 'pneumonia' labels.

The original dataset contains 5,863 observations, split into training (1,341 normal cases, 3,875 pneumonia cases), validation (8 normal cases, 8 pneumonia cases) and testing (234 normal cases, 390 pneumonia cases) folders.

CNN Explanation
Before we start Implementing CNN let's try to understand how it works

14 years ago, when 1st research in the topic of 'Convolutional neural networks' were published it could not attract attention around it. Since people were not related with the topic and had some difficulties in understanding it, the technique was underestimated.

How everything changed?

Everything changed in 2012, when the computer vision contest was held based on the ImageNet database. Alex Krichevsky and his team have developed a convolutional neural network that can classify millions of images from thousands of different categories with an error of only 15.8%.
Today, convolutional networks have evolved beyond human capabilities!

Now let's discover the most interesting part

How convolutional neural networks are trained?

Convolutional neural networks work on the basis of filters that recognize certain characteristics of an image (for example, straight lines). A filter is a collection of cores; sometimes a single core is used in a filter. A kernel is a regular matrix of numbers called weights that are "trained" in order to search for certain characteristics in images. The filter moves along the image and determines whether some desired characteristic is present in a particular part of it. To get an answer of this kind, a convolution operation is performed, which is the sum of the products of the filter elements and the input signal matrix.

4 Feature Engineering

4.1 Generating the Images

[11]:
```
class Generators:
    def __init__(self):
        self.batch_size=BATCH_SIZE
        self.img_size=(IMAGE_SIZE[0], IMAGE_SIZE[1])

        _train_datagen = ImageDataGenerator(
            featurewise_center=False,
            samplewise_center=False,
            featurewise_std_normalization=False,
```

```
            samplewise_std_normalization=False,
            zca_whitening=False,
            rotation_range = 30,
            zoom_range = 0.2,
            width_shift_range=0.1,
            height_shift_range=0.1,
            horizontal_flip = True,
            vertical_flip=False
        )

        self.train_generator = _train_datagen.flow_from_directory(
            directory=TRAIN_DIRECTORY,
            batch_size=self.batch_size,
            shuffle=True,
            class_mode="binary",
            seed=1,
            target_size=self.img_size)
        print('Train generator created')

        _val_datagen = ImageDataGenerator(
            featurewise_center=False,
            samplewise_center=False,
            featurewise_std_normalization=False,
            samplewise_std_normalization=False,
            zca_whitening=False,
            rotation_range = 30,
            zoom_range = 0.2,
            width_shift_range=0.1,
            height_shift_range=0.1,
            horizontal_flip = True,
            vertical_flip=False
        )
        self.val_generator = _val_datagen.flow_from_directory(
            directory=VALIDATION_DIRECTORY,
            batch_size=self.batch_size,
            shuffle=True,
            class_mode="binary",
            seed=1,
            target_size=self.img_size)
        print('Validation generator created')

        _test_datagen=ImageDataGenerator(
        )
        self.test_generator = _test_datagen.flow_from_directory(
            directory=TEST_DIRECTORY,
            batch_size=self.batch_size,
            shuffle=False,
```

```
            class_mode="binary",
            seed=1,
            target_size=self.img_size)
        print('Test generator created')
```

Using Image Generator library, I am extracting all the data (Images in our case) from the folders. However, there are some techniques that I decided to implement before extracting the data.

First of all, I decided to rescale the images to it´s default size which is 1./255. By setting featurewise_center=False,
samplewise_center=False,
featurewise_std_normalization=False,
samplewise_std_normalization=False,
zca_whitening=False

I prevent any changes in the architecture of the pictures itself. I also set 'Flips' to false so no displacement might occur. Moreover, I decided to zoom the pictures by 20% randomly and stretch them from left to right then from top to bottom by 10%.

4.1.1 Getting general info

[12]:
```
generators = Generators()
```

```
Found 5216 images belonging to 2 classes.
Train generator created
Found 16 images belonging to 2 classes.
Validation generator created
Found 624 images belonging to 2 classes.
Test generator created
```

4.2 Assigning class weights

[13]:
```
class_weights = class_weight.compute_class_weight('balanced',
                                                  classes=list(generators.
 ↪train_generator.class_indices.values()),
                                                  y=generators.train_generator.
 ↪classes)

class_weights = {k: v for k,v in enumerate(class_weights)}
class_weights
```

[13]:
```
{0: 1.9448173005219984, 1: 0.6730322580645162}
```

I assigned class weights in order to balance the class distrubtion and to ease the work of ml alghoritm.

5 Model Building

5.1 Pretrained model as a base

[14]:
```
base_model = VGG19(weights='imagenet', include_top=False,␣
 ↪input_shape=(224,224,3))

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

Experiments showed that without a pretrained model in the base our pipeline is overfitting.

Another solve to this problem was using a really complicated architecture.

So, what is VGG-19?

VGG-19 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 19 layers deep and can classify images into 1,000 categories of objects such as keyboard, mouse, pencil and many animals. So, in our case with pneumonia looks like a really good choice.

layer.trainable = False by setting it to false I am moving all the trainable weights to non-trainable.

'weights' = was choosing them by default.

5.2 Building the Layers

[16]:
```
def build_densenet(input_shape, n_classes, filters = 32):
    input = Input (input_shape)
    x=(base_model)
    x = Conv2D(64, 7, strides = 2, padding = 'same')(input)
    x = MaxPool2D(3, strides = 2, padding = 'same')(x)
    x = GlobalAveragePooling2D()(x)
    x = (LeakyReLU(alpha=0.1))(x)
    x = (Dropout(0.25)) (x) ####     Adding Dropout into the Network
    x = (Dense(128, activation='linear'))(x)
    x = (Flatten()) (x)
    output = Dense(1, activation = 'sigmoid')(x)
    model = Model(input, output)
    return model
```

In this section I am trying to build a simple model with few layers.

Before this model I was trying to build a complicated model with a lot of layers and at the end the architecture of the model was turning to be beyond my understanding.

In this case as we can see everything is pretty simple. 1 conventional layer, max pooling, Global average Pooling,
Moreover, I added a small dropout to prevent overfitting. Dense of 128 was added to pack the data points tighter to each other.

During the experiments was revealed that batch Normalization decreases efficiency of the model

5.3 Function to return the scores

[17]:
```
def f1_score(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
```

5.4 Compiling the model

[18]:
```
input_shape = IMAGE_SIZE[0], IMAGE_SIZE[1], 3
n_classes = len(generators.train_generator.class_indices)

filters = 32
clf = build_densenet(input_shape, n_classes, filters = 32)
clf.compile(optimizer="rmsprop", loss='binary_crossentropy',␣
 ↪metrics=['accuracy', Precision(), Recall(), f1_score])

clf.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 180, 180, 3)]     0

 conv2d (Conv2D)             (None, 90, 90, 64)        9472

 max_pooling2d (MaxPooling2D  (None, 45, 45, 64)       0
 )

 global_average_pooling2d (G  (None, 64)               0
 lobalAveragePooling2D)

 leaky_re_lu (LeakyReLU)     (None, 64)                0

 dropout (Dropout)           (None, 64)                0

 dense (Dense)               (None, 128)               8320
```

```
 flatten (Flatten)           (None, 128)               0

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 17,921
Trainable params: 17,921
Non-trainable params: 0
_________________________________________________________________
```

The model that we built consist only of 17.000 hyperparameters. For comparasion my previous models were consisted of approximately 2,116,448-3,000,000 parameters.

5.4.1 Installing chart package

[19]:
```
!pip install livelossplot
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: livelossplot in /usr/local/lib/python3.7/dist-
packages (0.5.5)
Requirement already satisfied: bokeh in /usr/local/lib/python3.7/dist-packages
(from livelossplot) (2.3.3)
Requirement already satisfied: numpy<1.22 in /usr/local/lib/python3.7/dist-
packages (from livelossplot) (1.21.6)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-
packages (from livelossplot) (3.2.2)
Requirement already satisfied: ipython==7.* in /usr/local/lib/python3.7/dist-
packages (from livelossplot) (7.34.0)
Requirement already satisfied: pygments in /usr/local/lib/python3.7/dist-
packages (from ipython==7.*->livelossplot) (2.6.1)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.7/dist-
packages (from ipython==7.*->livelossplot) (4.8.0)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.7/dist-
packages (from ipython==7.*->livelossplot) (5.1.1)
Requirement already satisfied: setuptools>=18.5 in
/usr/local/lib/python3.7/dist-packages (from ipython==7.*->livelossplot)
(57.4.0)
Requirement already satisfied: matplotlib-inline in
/usr/local/lib/python3.7/dist-packages (from ipython==7.*->livelossplot) (0.1.3)
Requirement already satisfied: backcall in /usr/local/lib/python3.7/dist-
packages (from ipython==7.*->livelossplot) (0.2.0)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.7/dist-
packages (from ipython==7.*->livelossplot) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from ipython==7.*->livelossplot)
(3.0.30)
```

```
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.7/dist-
packages (from ipython==7.*->livelossplot) (0.18.1)
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-
packages (from ipython==7.*->livelossplot) (4.4.2)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in
/usr/local/lib/python3.7/dist-packages (from
jedi>=0.16->ipython==7.*->livelossplot) (0.8.3)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.7/dist-
packages (from pexpect>4.3->ipython==7.*->livelossplot) (0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-packages
(from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython==7.*->livelossplot)
(0.2.5)
Requirement already satisfied: typing-extensions>=3.7.4 in
/usr/local/lib/python3.7/dist-packages (from bokeh->livelossplot) (4.1.1)
Requirement already satisfied: tornado>=5.1 in /usr/local/lib/python3.7/dist-
packages (from bokeh->livelossplot) (5.1.1)
Requirement already satisfied: PyYAML>=3.10 in /usr/local/lib/python3.7/dist-
packages (from bokeh->livelossplot) (3.13)
Requirement already satisfied: packaging>=16.8 in /usr/local/lib/python3.7/dist-
packages (from bokeh->livelossplot) (21.3)
Requirement already satisfied: Jinja2>=2.9 in /usr/local/lib/python3.7/dist-
packages (from bokeh->livelossplot) (2.11.3)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.7/dist-packages (from bokeh->livelossplot) (2.8.2)
Requirement already satisfied: pillow>=7.1.0 in /usr/local/lib/python3.7/dist-
packages (from bokeh->livelossplot) (7.1.2)
Requirement already satisfied: MarkupSafe>=0.23 in
/usr/local/lib/python3.7/dist-packages (from Jinja2>=2.9->bokeh->livelossplot)
(2.0.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/usr/local/lib/python3.7/dist-packages (from
packaging>=16.8->bokeh->livelossplot) (3.0.9)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-
packages (from python-dateutil>=2.1->bokeh->livelossplot) (1.15.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-
packages (from matplotlib->livelossplot) (0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib->livelossplot) (1.4.3)
```

This package are used to plot lifetime charts of val,accuracy and etc

5.5 Loading the weights

[20]:
```
model_weights_file_path = "best_model_weights.h5"
checkpoint = ModelCheckpoint(filepath=model_weights_file_path,␣
 ↪monitor="val_accuracy", verbose=1, save_best_only=True, mode="max",␣
 ↪save_weights_only=True)
```

```
early_stopping = EarlyStopping(monitor="val_accuracy", mode="max", verbose=1,␣
 ↪patience=20)
lr_reduce = ReduceLROnPlateau(monitor='val_accuracy', factor=0.5, patience=10,␣
 ↪verbose=0, mode='max', min_delta=0.0001, cooldown=0, min_lr=0)
plotlosses = PlotLossesKeras()

call_backs = [checkpoint, early_stopping, plotlosses, lr_reduce]
```

By using ModelCheckpoint I am making conjunction with training using model. fit() to save a model or weights (in a checkpoint file) at some interval.

Ealy stopping is used to prevent overfitting again.

LR_reduce is used to reduce the learning rate. All other parameters were explained before.

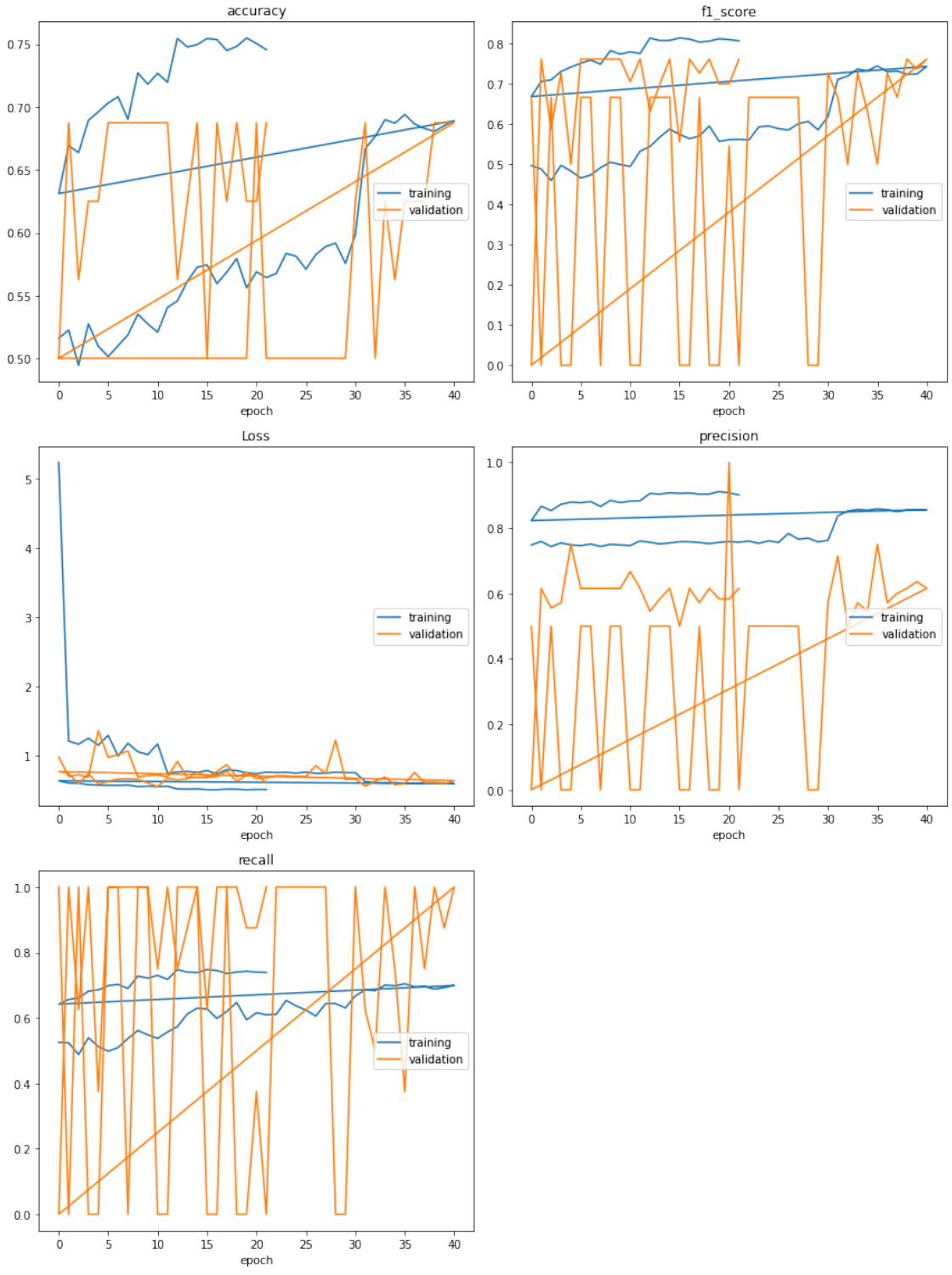
5.6 Architecture of the model

plot_model(clf, to_file='model_plot.png', show_shapes=True, show_layer_names=True)

By plotting this architecture we can see overall amount of layer. Since I explained all off the layers in model building part.

5.7 Fitting the Model

[37]:
```
history = clf.fit(generators.train_generator,
                  validation_data=generators.val_generator,
                  epochs=EPOCHS,
                  batch_size=BATCH_SIZE,
                  callbacks=call_backs,
                  class_weight=class_weights,
                  verbose=1)
```




```
accuracy
        training                (min:    0.494, max:    0.755, cur:    0.746)
        validation              (min:    0.500, max:    0.688, cur:    0.688)
```

```
f1_score
        training                (min:    0.460, max:    0.814, cur:    0.807)
        validation              (min:    0.000, max:    0.762, cur:    0.762)
Loss
        training                (min:    0.510, max:    5.238, cur:    0.514)
        validation              (min:    0.558, max:    1.362, cur:    0.607)
precision
        training                (min:    0.743, max:    0.911, cur:    0.901)
        validation              (min:    0.000, max:    1.000, cur:    0.615)
recall
        training                (min:    0.488, max:    0.748, cur:    0.739)
        validation              (min:    0.000, max:    1.000, cur:    1.000)
82/82 [==============================] - 82s 992ms/step - loss: 0.5143 -
accuracy: 0.7456 - precision: 0.9009 - recall: 0.7388 - f1_score: 0.8073 -
val_loss: 0.6071 - val_accuracy: 0.6875 - val_precision: 0.6154 - val_recall:
1.0000 - val_f1_score: 0.7619 - lr: 6.2500e-05
Epoch 22: early stopping
```

In fitting part of the model I used all the hyperparameters that were defined before as variables. Moreover, the liveplot is helping us to see live charts.

5.7.1 Loading the best weights

[38]:
```
clf.load_weights("./best_model_weights.h5")
```

Save and load Keras models is used to get the best weights

6 Model Assesment

[44]:
```
y_hat = clf.predict(generators.test_generator)
```

[47]:
```
y_hat = (y_hat > 0.5)+0
```

6.1 Defining the function

def print_score(y_pred, y_real): print("Accuracy: ", accuracy_score(y_real, y_pred))

```
print()
print("Macro precision_recall_fscore_support (macro) average")
print(precision_recall_fscore_support(y_real, y_pred, average="macro"))

print()
print("Macro precision_recall_fscore_support (micro) average")
print(precision_recall_fscore_support(y_real, y_pred, average="micro"))
```

```
print()
print("Macro precision_recall_fscore_support (weighted) average")
print(precision_recall_fscore_support(y_real, y_pred, average="weighted"))

print()
print("Confusion Matrix")
print(confusion_matrix(y_real, y_pred))

print()
print("Classification Report")
print(classification_report(y_real, y_pred))
```

6.2 Score Reports

[48]:
```
print_score(y_hat, generators.test_generator.classes.reshape((-1,1)))
```

```
Accuracy:  0.42467948717948717

Macro precision_recall_fscore_support (macro) average
(0.6683725442638855, 0.538034188034188, 0.3595425630762633, None)

Macro precision_recall_fscore_support (micro) average
(0.42467948717948717, 0.42467948717948717, 0.42467948717948717, None)

Macro precision_recall_fscore_support (weighted) average
(0.7369936939121998, 0.42467948717948717, 0.3084804517189622, None)

Confusion Matrix
[[232   2]
 [357  33]]

Classification Report
              precision    recall  f1-score   support

           0       0.39      0.99      0.56       234
           1       0.94      0.08      0.16       390

    accuracy                           0.42       624
   macro avg       0.67      0.54      0.36       624
weighted avg       0.74      0.42      0.31       624
```

6.2.1 Splitting train generators in variables

[42]:
```
key_list = list(generators.train_generator.class_indices.keys())
val_list = list(generators.train_generator.class_indices.keys())
```

[43]:
```
key_list
```

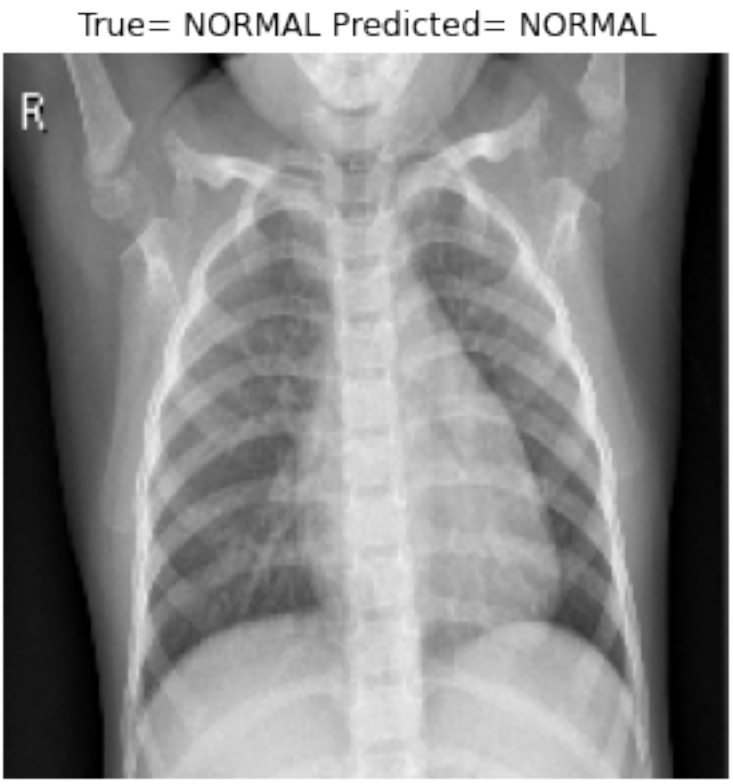
[43]:
```
['NORMAL', 'PNEUMONIA']
```

6.3 Predicting an actual photo

[36]:
```
gen_next = generators.test_generator.next()
smaple = gen_next[0]
y_hat = clf.predict(np.array([smaple[0]]))
y_hat = (y_hat > 0.5)+0
y_hat = y_hat[-1][-1]

plt.figure(figsize = (5,5))
plt.imshow(smaple[0]/255);
plt.title("True= {} Predicted= {}".format(
    key_list[int(gen_next[1][0])],
    key_list[y_hat]
    ))
plt.axis('off')
```

[36]:
```
(-0.5, 179.5, 179.5, -0.5)
```



7 Conclusion

The dataset that was chosen for this project was not the most appropriate one. The first reason was the it has big capacity and working with it is really slow, however normally setting minimum sample size has to helpful, but again not in my case because for some reason models are not performing well on a small dataset sample. The next issue was the complexity of architecture. Before this model I was trying to build a model with complicated architecture, with 14-16 layers that are cross connected to each other, but unfortunately, they were either copies from open sources, or the models that were built with the help of experienced data scientist. But unfortunately, they were beyond my understanding. So as a final decision I decided to keep this simple model that does not perform that well, but at least is understandable and was built by me.