INHA UNIVERSITY TASHKENT
DEPARTMENT OF CSE & ICE

FALL SEMESTER 2022

SOC-4080: MULTIMEDIA APPLICATION

TERM PROJECT

Pneumonia Recognition using Chest X-Ray Images

Submitted by

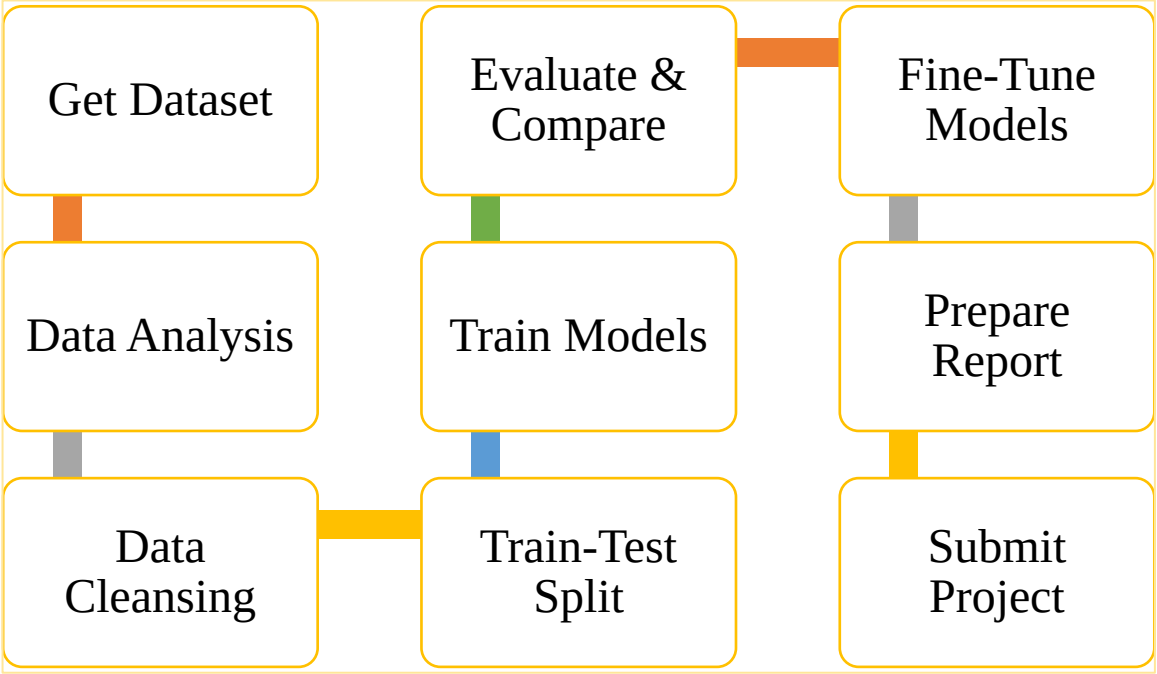
Student Name	Student ID	Section#
Rustamov Sunatillo	U1910136	002
Pak Anna	U1910243	002
Sultanov Azamat	U1910256	002

Group : SENIOR Team #35

Contents

Pneumonia Recognition using Chest X-Ray Images

Project Flowchart



Overview

Pneumonia is an infection that inflames the air sacs in one or both lungs. The air sacs may fill with fluid or pus (purulent material), causing cough with phlegm or pus, fever, chills, and difficulty breathing. A variety of organisms, including bacteria, viruses and fungi, can cause pneumonia.

Pneumonia can range in seriousness from mild to life-threatening. It is most serious for infants and young children, people older than age 65, and people with health problems or weakened immune systems.

Key Facts

- Pneumonia accounts for 14% of all deaths of children under 5 years old, killing 740 180 children in 2019

- Pneumonia can be caused by viruses, bacteria, or fungi.
- Pneumonia can be prevented by immunization, adequate nutrition, and by addressing environmental factors.
- Pneumonia caused by bacteria can be treated with antibiotics, but only one third of children with pneumonia receive the antibiotics they need.

Why is it a problem?

Pneumonia is the world's leading cause of death among children under 5 years of age, accounting for 16% of all deaths of children under 5 years old killing approximately 2,400 children a day in 2015.

For US adults, pneumonia is the most common cause of hospital admissions other than women giving birth.

While young healthy adults have less risk of pneumonia than the age extremes, it is always a threat.

Our changing interactions with the microbial world mean constantly developing new pneumonia risks.

Why it is important to be detected early?

Early identification of pneumonia and appropriate treatment saves lives. By initiating antibiotic therapy soon after onset of symptoms such as fast breathing in a child with cough, the progression of a pneumonia infection is blunted. In the absence of early intervention, pneumonia progresses to a state where even

intravenous antibiotics have limited impact, leading to the high hospital case–fatality rates for children.

The application of computer-assisted diagnostic models to help radiologists to more quickly and accurately interpret chest X-ray images, to screen and classify patients, is highly required. The integration of this algorithm into the clinical system could help health institutions to advance patient care by reducing the time to diagnosis and increasing access to chest radiograph interpretation, as well as mitigating the shortage of expert radiologists in remote areas.

About Dataset

Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children’s Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients’ routine clinical care.

For the analysis of chest x-ray images, all chest radiographs were initially screened for quality control by removing all low quality or unreadable scans. The diagnoses for the images were then graded by two expert physicians before being cleared for training the AI system. In order to account for any grading errors, the evaluation set was also checked by a third expert.

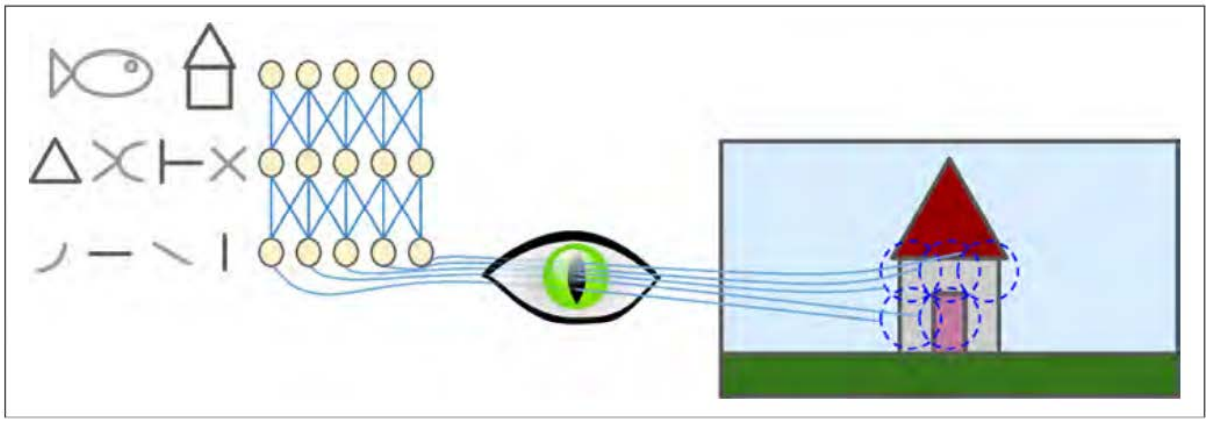
Algorithm applied

Convolutional Neural Networks (CNNs)

Convolutional neural networks (CNNs) emerged from the study of the brain's **visual cortex**, and they have been used in image recognition since the 1980s.

The architecture of visual cortex

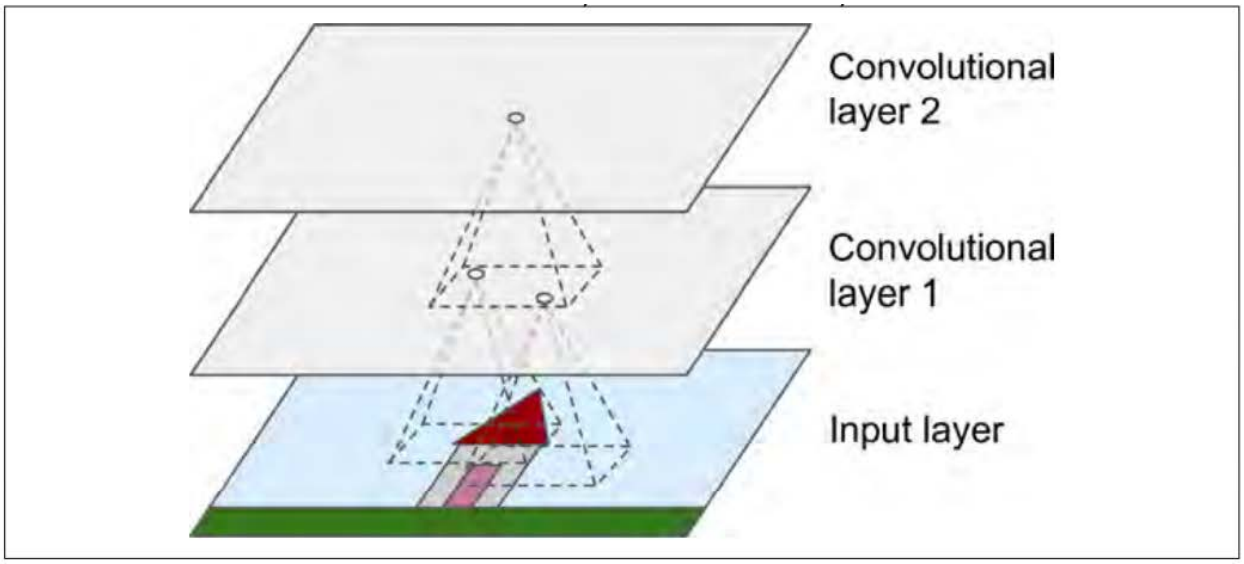
Many neurons in the visual cortex have a small local receptive field, meaning they react only to visual stimuli located in a limited region of the visual field. The receptive fields of different neurons may overlap, and together they tile the whole visual field. Some neurons react only to images of horizontal lines, while others react only to lines with different orientations (two neurons may have the same receptive field but react to different line orientations). Some neurons have larger receptive fields, and they react to more com- plex patterns that are combinations of the lower-level patterns, so the higher-level neurons are based on the outputs of neighboring lower-level neurons. This powerful architecture is able to detect all sorts of complex patterns in any area of the visual field.



These studies of the visual cortex inspired the neocognitron, which gradually evolved into what we now call convolutional neural networks. This architecture has some building blocks that you already know, such as fully connected layers and sigmoid activation functions, but it also introduces two new building blocks: convolutional layers and pooling layers.

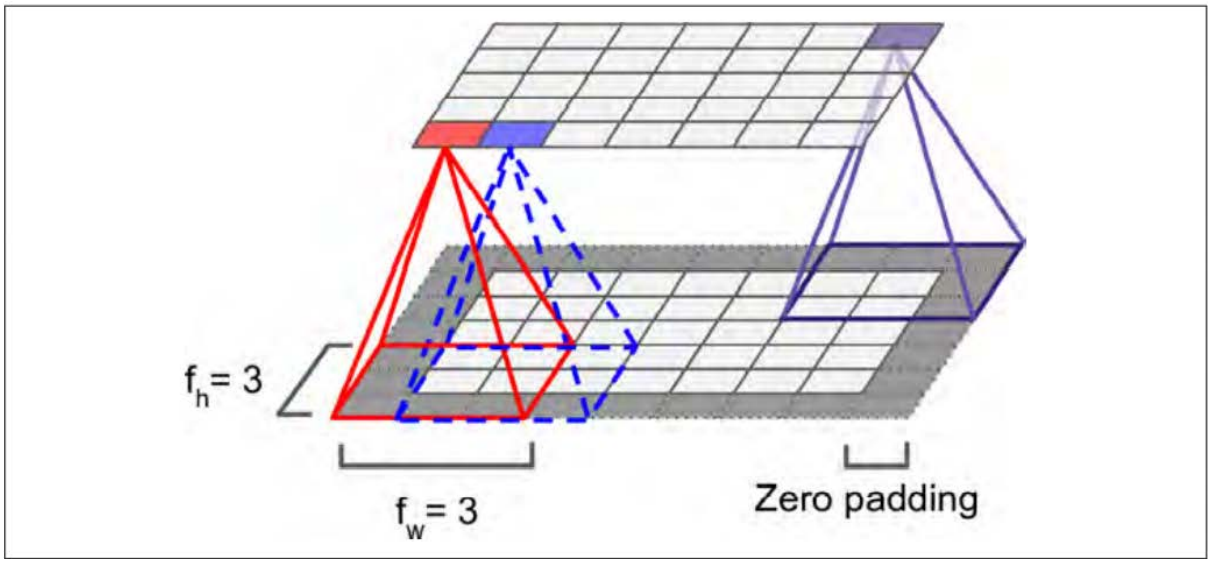
Convolutional Layer

Neurons in the first convolutional layer are not connected to every single pixel in the input image, but only to pixels in their receptive fields.



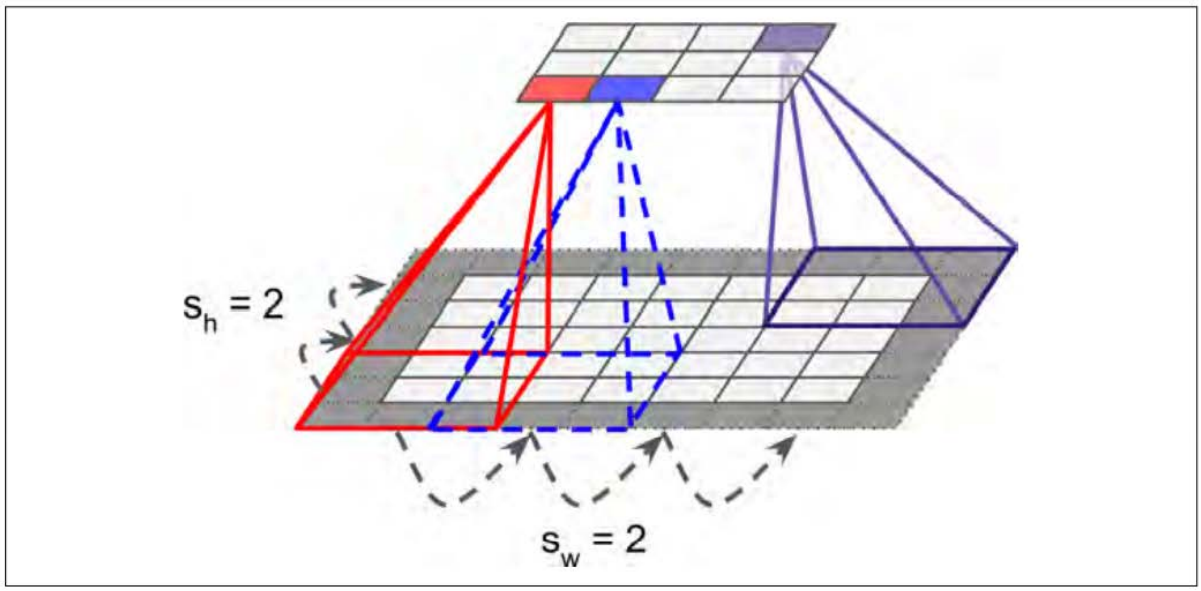
This architecture allows the network to concentrate on low-level features in the first hidden layer, then assemble them into higher-level features in the next hidden layer, and so on.

A neuron located in row i, column j of a given layer is connected to the outputs of the neurons in the previous layer located in rows i to i + fh – 1, columns j to j + fw – 1, where fh and fw are the height and width of the receptive field.



In order for a layer to have the same height and width as the previous layer, it is common to add zeros around the inputs, as shown in the diagram. This is called zero pad- ding.

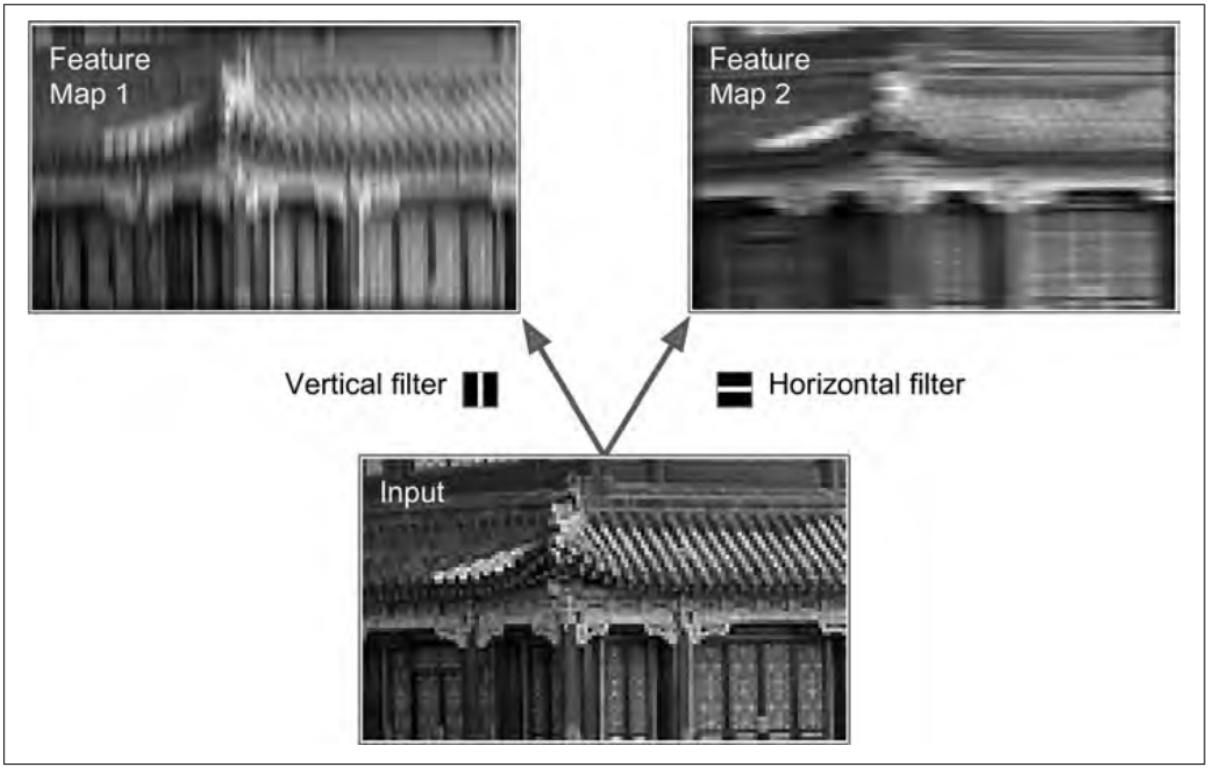
It is also possible to connect a large input layer to a much smaller layer by spacing out the receptive fields. The distance between two consecutive receptive fields is called the stride.



A neuron located in row i, column j in the upper layer is connected to the outputs of the neurons in the previous layer located in rows i × sh to i × sh + fh – 1, columns j × sw + fw – 1, where sh and sw are the vertical and horizontal strides.

Filters

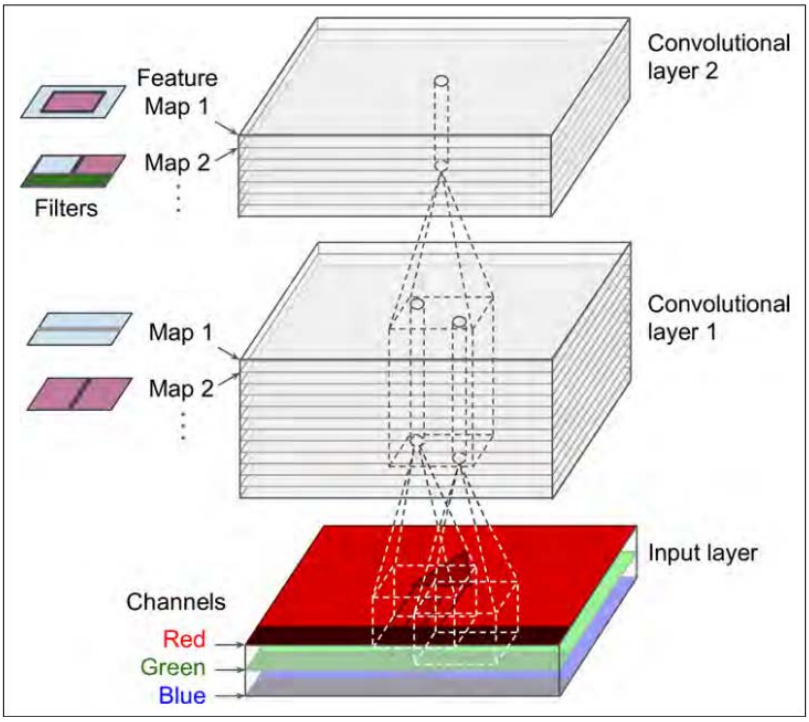
A neuron's weights can be represented as a small image the size of the receptive field. Figure shows two possible sets of weights, called filters (or convolution kernels).



During training, a CNN finds the most useful filters for its task, and it learns to combine them into more complex patterns (e.g., a cross is an area in an image where both the vertical filter and the horizontal filter are active).

Stacking Multiple Feature Maps

Up to now, for simplicity, we have represented each convolutional layer as a thin 2D layer, but in reality, it is composed of several feature maps of equal sizes, so it is more accurately represented in 3D.



Once the CNN has learned to recognize a pattern in one location, it can recognize it in any other location.

Equation: Computing the output of a neuron in a convolutional layer

$$z_{i,j,k} = b_k + \sum_{u=1}^{f_h} \sum_{v=1}^{f_w} \sum_{k'=1}^{f_{n'}} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with} \begin{cases} i' = u \,.\, s_h + f_h - 1 \\ j' = v \,.\, s_w + f_w - 1 \end{cases}$$

All it does is calculate the weighted sum of all the inputs, plus the bias term.

- $z_{i,j,k}$ is the output of the neuron located in row i, column j in feature map k of the convolutional layer (layer l).
- As explained earlier, s_h and s_w are the vertical and horizontal strides, f_h and f_w are the height and width of the receptive field, and $f_{n'}$ is the number of feature maps in the previous layer (layer $l - 1$).
- $x_{i',j',k'}$ is the output of the neuron located in layer $l - 1$, row i', column j', feature map k' (or channel k' if the previous layer is the input layer).
- b_k is the bias term for feature map k (in layer l). You can think of it as a knob that tweaks the overall brightness of the feature map k.
- $w_{u,v,k',k}$ is the connection weight between any neuron in feature map k of the layer l and its input located at row u, column v (relative to the neuron's receptive field), and feature map k'.

Sequential model from Keras

The Sequential model is a linear stack of layers. Sequential model can be created by passing a list of layer instances to the constructor:

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_dim=784),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

Or add layers via the .add() method:

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

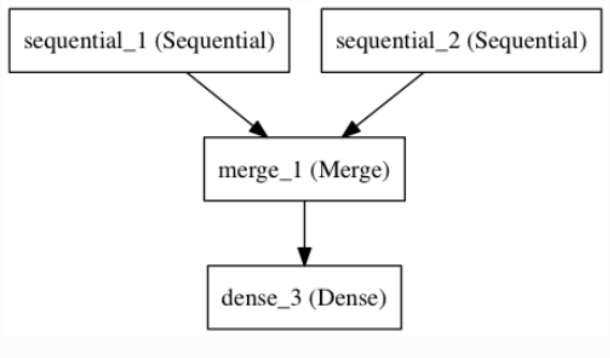
The model needs to know what input shape it should expect. For this reason, the first layer in a Sequential model (and only the first, because following layers can do

automatic shape inference) needs to receive information about its input shape. There are several possible ways to do this:

- pass an input_shape argument to the first layer. This is a shape tuple (a tuple of integers or None entries, where None indicates that any positive integer may be expected). In input_shape, the batch dimension is not included.
- pass instead a batch_input_shape argument, where the batch dimension is included. This is useful for specifying a fixed batch size (e.g. with stateful RNNs).
- some 2D layers, such as Dense, support the specification of their input shape via the argument input_dim, and some 3D temporal layers support the arguments input_dim and input_length.

Multiple Sequential instances can be merged into a single output via a Merge layer. The output is a layer that can be added as first layer in a new Sequential model. For instance, here's a model with two separate input branches getting merged:

```
from keras.layers import Merge

left_branch = Sequential()
left_branch.add(Dense(32, input_dim=784))

right_branch = Sequential()
right_branch.add(Dense(32, input_dim=784))

merged = Merge([left_branch, right_branch], mode='concat')

final_model = Sequential()
final_model.add(merged)
final_model.add(Dense(10, activation='softmax'))
```



Such a two-branch model can then be trained via e.g.:

```
final_model.compile(optimizer='rmsprop',
loss='categorical_crossentropy')
final_model.fit([input_data_1, input_data_2], targets)  # we
pass one data array per model input
```

The Merge layer supports a number of pre-defined modes:

- sum (default): element-wise sum
- concat: tensor concatenation. You can specify the concatenation axis via the argument concat_axis.
- mul: element-wise multiplication
- ave: tensor average
- dot: dot product. You can specify which axes to reduce along via the argument dot_axes.
- cos: cosine proximity between vectors in 2D tensors.

Before training a model, there is a need to configure the learning process, which is done via the compile method. It receives three arguments:

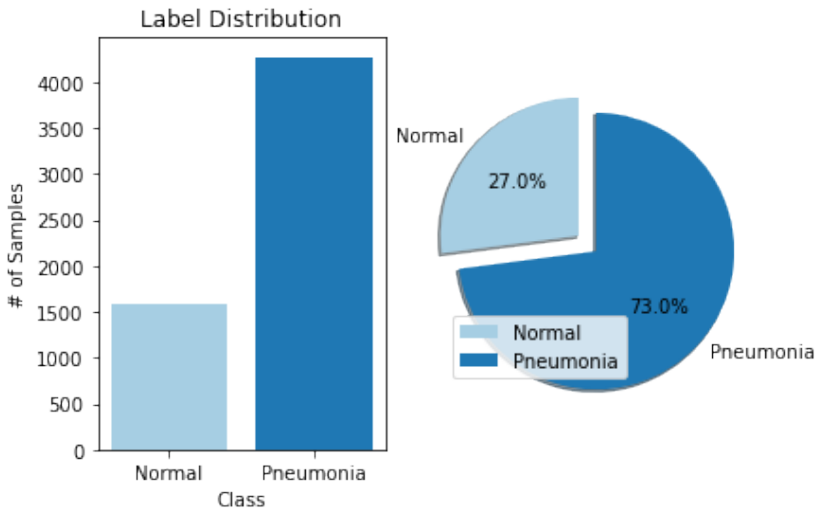
- an optimizer. This could be the string identifier of an existing optimizer (such as rmsprop or adagrad), or an instance of the Optimizer class. See: optimizers.

- a loss function. This is the objective that the model will try to minimize. It can be the string identifier of an existing loss function (such as categorical_crossentropy or mse), or it can be an objective function. See: objectives.
- a list of metrics. For any classification problem you will want to set this to metrics=['accuracy']. A metric could be the string identifier of an existing metric or a custom metric function. Custom metric function should return either a single tensor value or a dict metric_name -> metric_value.

Keras models are trained on Numpy arrays of input data and labels. For training a model, fit function is used.

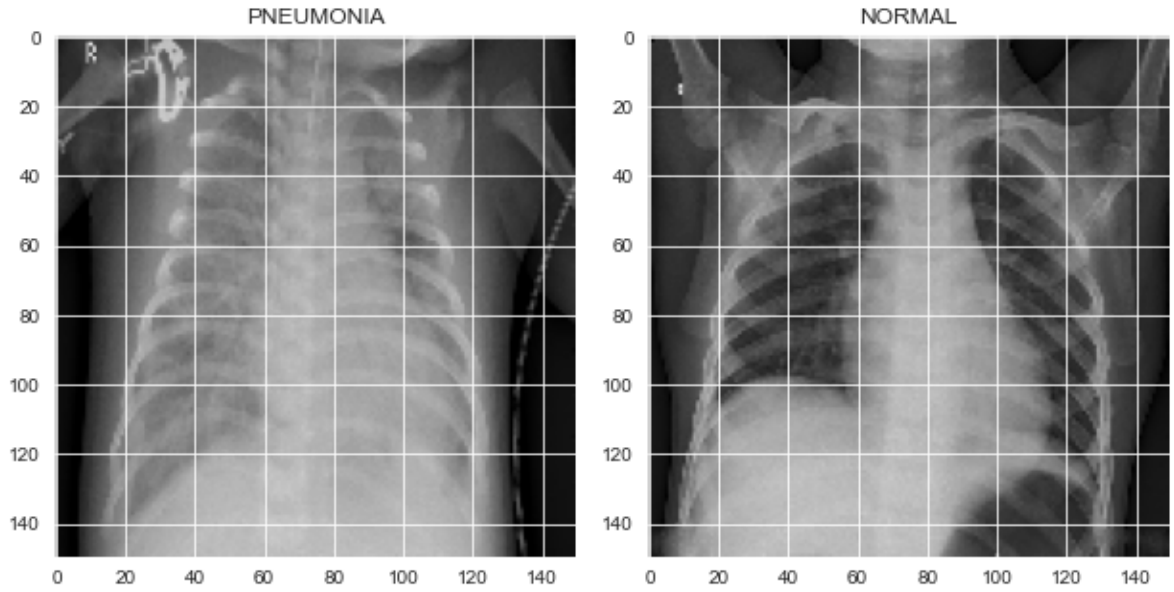
About Dataset

For this problem we used a dataset from a Medical Center located in China. Overall, we have more than 5.000 chest x ray images of children between 1 and 5 years old. Our dataset is also imbalanced, since we have almost 3/4 of pneumonia positive images.



Data Augmentation

Before training our model, let us take a look on some sample images:



For better experiments results, we had to preprocess train images. In our case, preprocessing included several operations:

- Normalize images
- Resize to 150*150 pixels
- Set input and each sample mean to 0 (standardize)
- Apply ZCA Whitening
- Rotate randomly in range 0-180 degrees
- Zoom randomly
- Shift and flip images

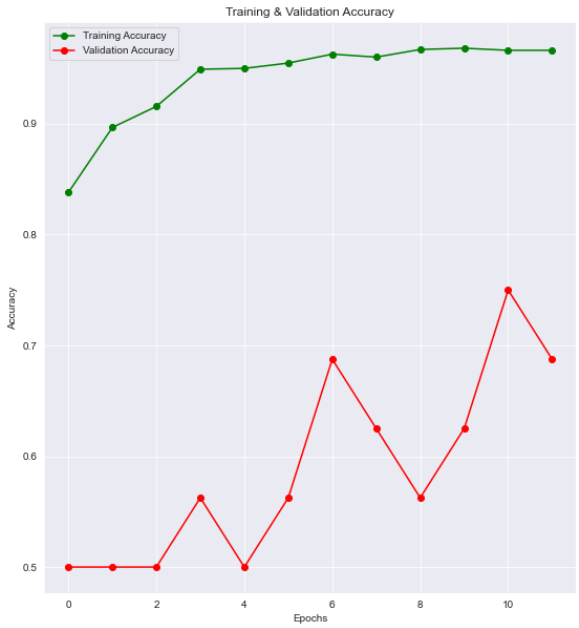
Data Augmentation was done using ImageDataGenerator from keras preprocessing image library.

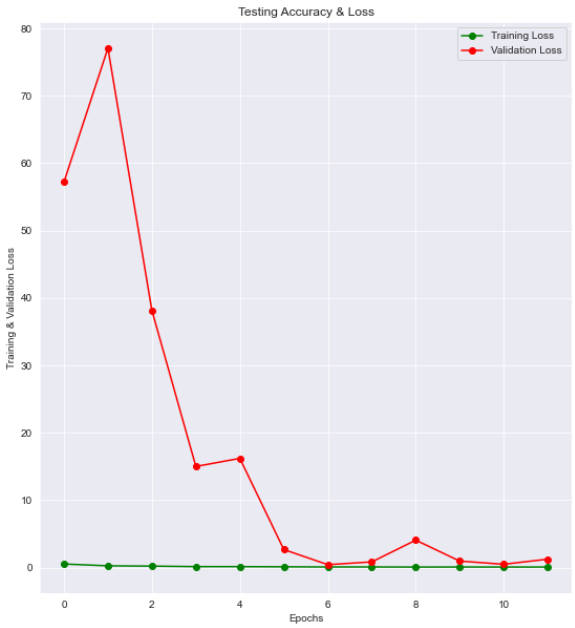
Implementation

CNN applied using Sequential from keras with following parameters:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 150, 150, 32)      320

batch_normalization (BatchN  (None, 150, 150, 32)      128
ormalization)

max_pooling2d (MaxPooling2D  (None, 75, 75, 32)        0
)

conv2d_1 (Conv2D)            (None, 75, 75, 64)        18496

dropout (Dropout)            (None, 75, 75, 64)        0

batch_normalization_1 (Batc  (None, 75, 75, 64)        256
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 38, 38, 64)        0
2D)

conv2d_2 (Conv2D)            (None, 38, 38, 64)        36928

batch_normalization_2 (Batc  (None, 38, 38, 64)        256
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 19, 19, 64)        0
2D)

conv2d_3 (Conv2D)            (None, 19, 19, 128)       73856

dropout_1 (Dropout)          (None, 19, 19, 128)       0

batch_normalization_3 (Batc  (None, 19, 19, 128)       512
hNormalization)

max_pooling2d_3 (MaxPooling  (None, 10, 10, 128)       0
2D)

conv2d_4 (Conv2D)            (None, 10, 10, 256)       295168

dropout_2 (Dropout)          (None, 10, 10, 256)       0

batch_normalization_4 (Batc  (None, 10, 10, 256)       1024
hNormalization)

max_pooling2d_4 (MaxPooling  (None, 5, 5, 256)         0
2D)

flatten (Flatten)            (None, 6400)              0

dense (Dense)                (None, 128)               819328

dropout_3 (Dropout)          (None, 128)               0

dense_1 (Dense)              (None, 1)                 129

=================================================================
Total params: 1,246,401
Trainable params: 1,245,313
Non-trainable params: 1,088
```

Model Accuracy and Loss on Train and Test set on 12 epochs:



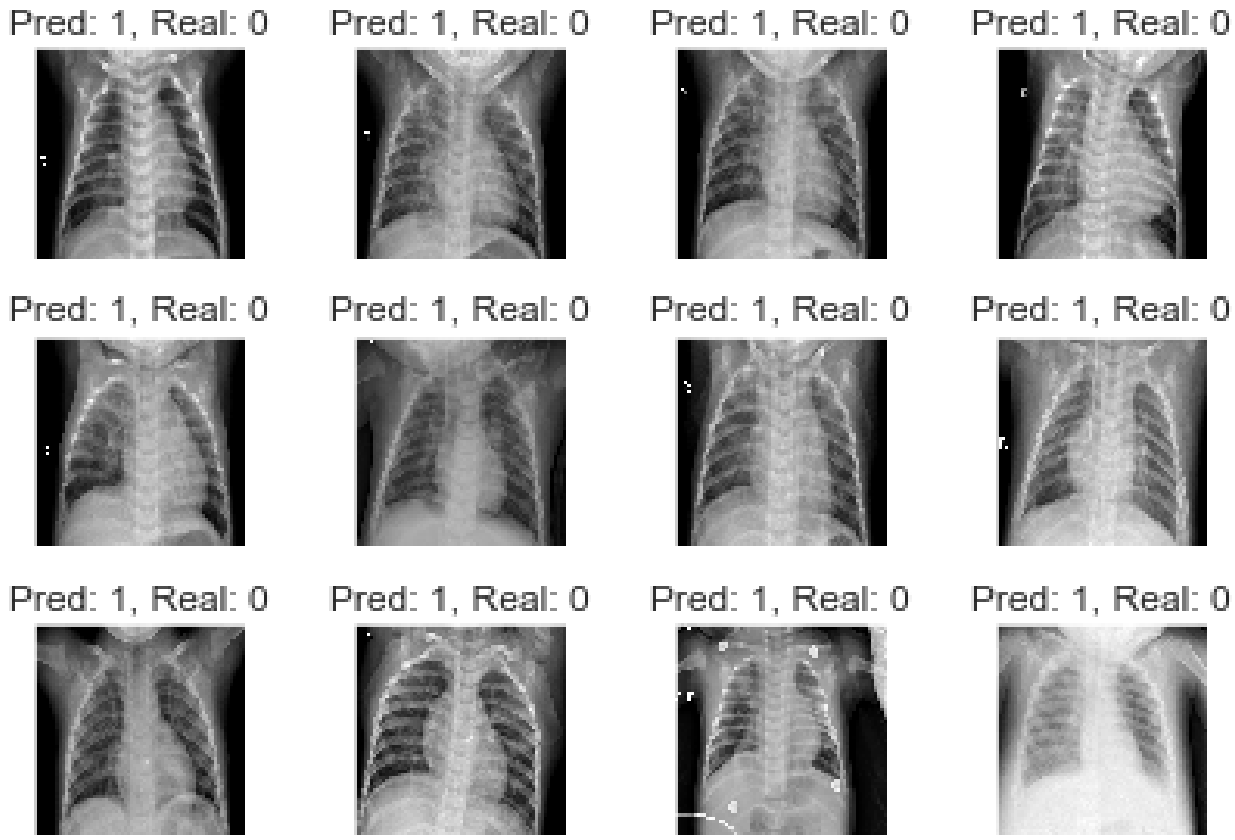


Experiment Results

Classification Report

	precision	recall	f1-score	support
Pneumonia (Class 0)	0.93	0.95	0.94	390
Normal (Class 1)	0.91	0.87	0.89	234
accuracy			0.92	624
macro avg	0.92	0.91	0.92	624
weighted avg	0.92	0.92	0.92	624

Since we are working with medical dataset, accuracy is less important than Recall. But High Recall means that the model will treat healthy people as the sick ones more often. It can be seen on the plot with incorrectly classified images, where 1 = pneumonia, and 0 = normal

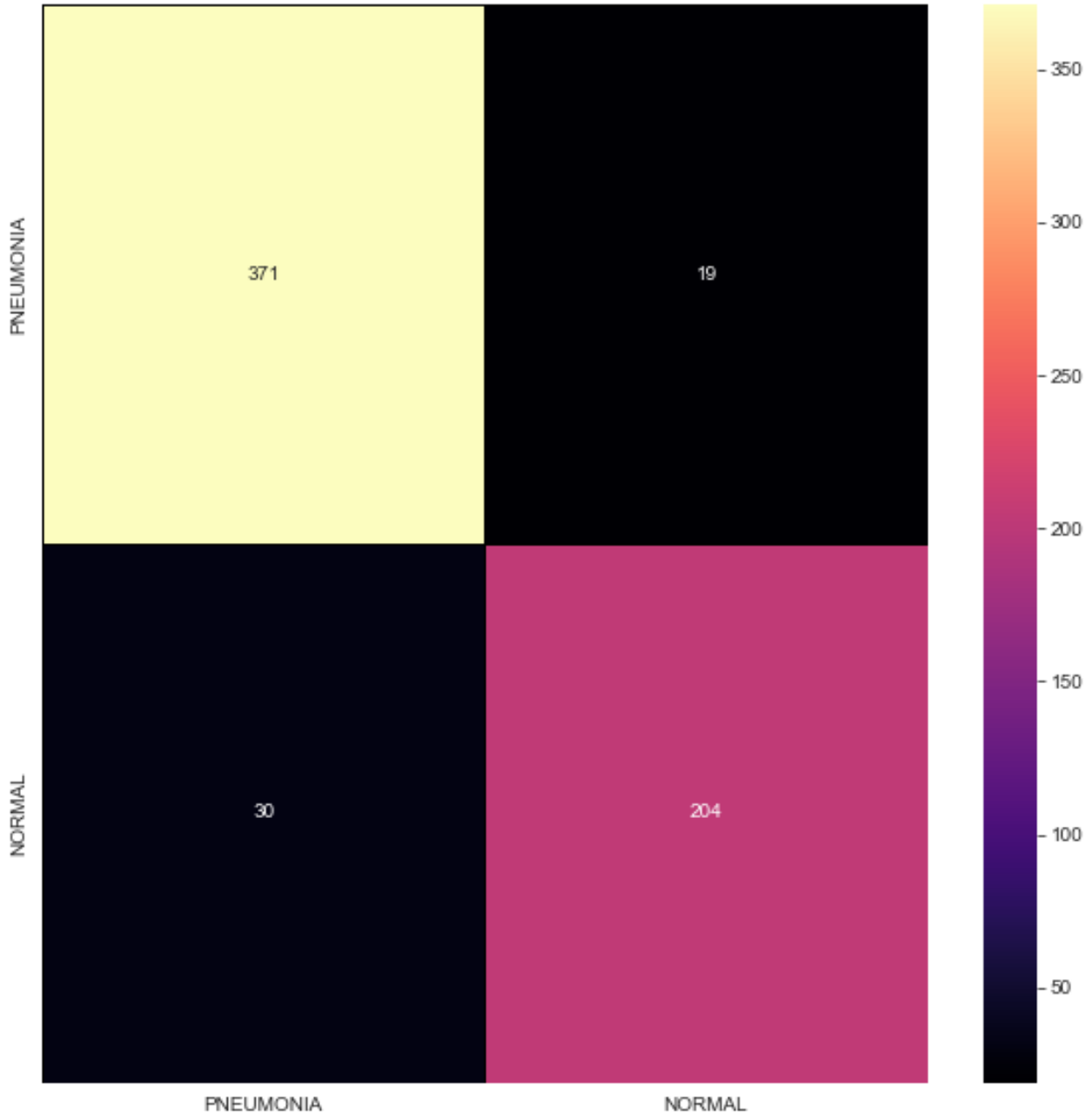


Performance Measures

- ✓ **Accuracy:** *91.82%*
- ✓ **Precision:** *91,85%*
- ✓ **Recall:** *95,38%*
- ✓ **F1 Score:** *93,58%*

From these scores, it can be seen that model performs pretty well. But probably, our model overfit the data, because we have just ~5000 training images, and only 624 test images. This problem can be fixed by increasing the number of training images.

Confusion Matrix



Links and References

Presentation Video: https://youtu.be/eB3dzgk4LRc

GitHub: https://github.com/PakAnna/Chest_x_ray_MA_Project

Dataset: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia