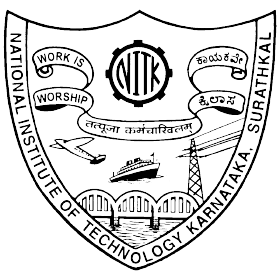
CS 351 - Machine Learning



National Institute of Technology, Karnataka

**Covid-19 Prediction using Neural Network**

Submitted By

**Atul Kailas Patil (181CO112)**

**Mohit Ajit Patil (181CO133)**

Mentor

**Nitesh Naik**

Submitted to

**Dr. M Venkatesh**

**Covid-19 prediction using CNN**

**INTRODUCTION**

This project utilizes several open-source or public datasets to present an open-source dataset of COVID-19 CXRs, named COVID-19-CXR-Dataset, and introduces a deep convolutional neural network model.

A deep convolutional neural network is an intuitive and powerful network architecture in deep learning and is widely used in pattern recognition and image classification tasks. In recent years, with the further development of deep learning technology, more and more efficient DCNN models have been proposed one after another, like VGG, ResNet, DenseNet, EfficientNet, etc. These DCNNs perform well in image classification tasks, making it possible for computers to perform better than humans in visual classification.

**Visual Geometry Group(VGG)**

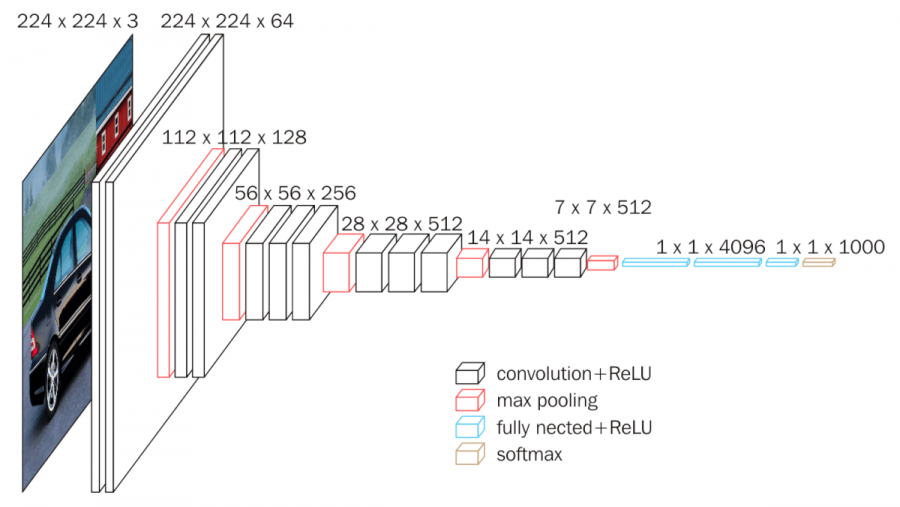
* 

Fig 1 . VGG Architecure

Visual Geometry Group (VGG) is a convolutional neural network architecture.

Very good at classifying image data.

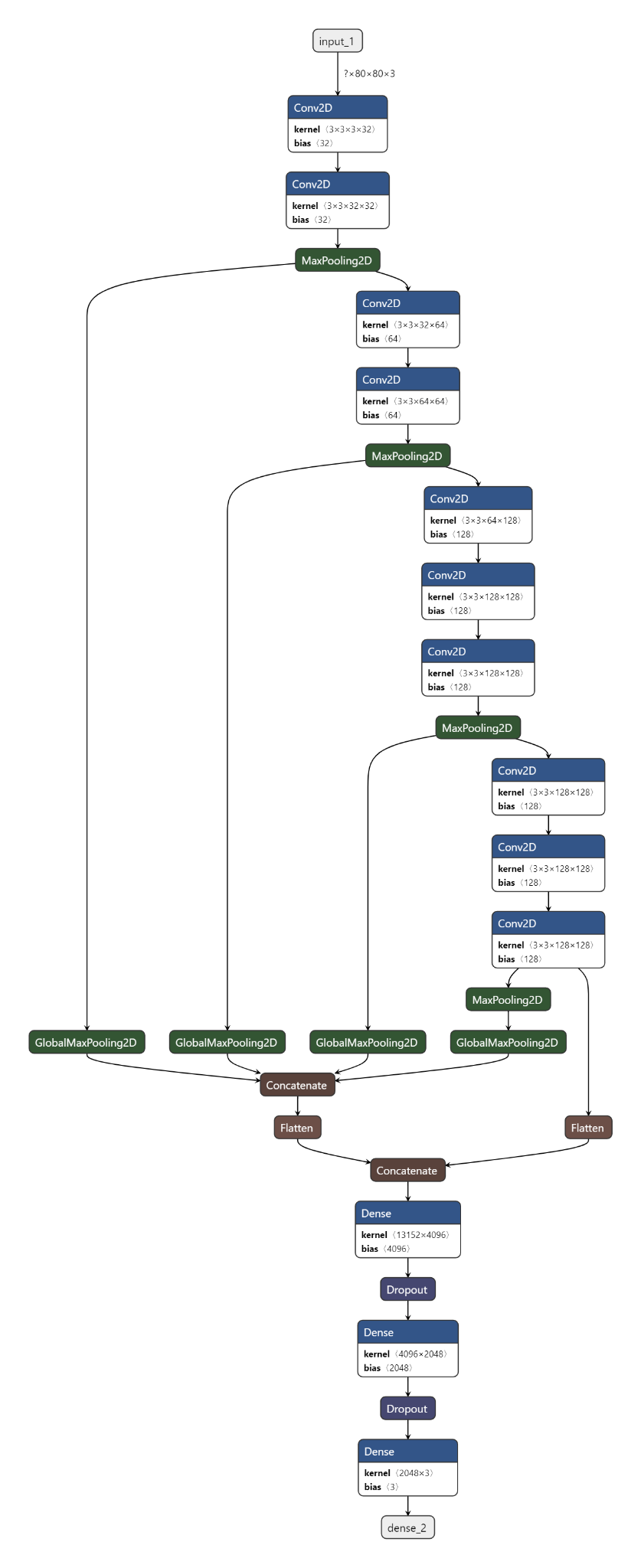
* VGG networks only use small convolution kernels and increase the number of network layers to improve the accuracy in the classification tasks.
* No doubt increasing the depth of the network allows the model to better fit complex data with higher dimensions, and the use of small convolution kernels can greatly reduce the number of parameters.
* Shallow networks usually have rich semantic information, but VGG does not make the most of this semantic information

VGG consists of 16 layers, where there are 3 different layers which are,

* Conv Layer
* Pooling Layer
* Fully Connected / Dense Layer

**Modified Visual Geometry Group(MVGG)**

We used VGG but modified it. The initialization in this Architecture is the same starting with a Convolutional layer of stripes = 2 and 32 filers. A 64 initial filter layer could be possible but it would increase the time required to make the model by multiple times. After 2 Conv2d layers, we took max-pooling then take global-max-pooling. Similar thing in consecutive layers. Hence finally we concatenated all the max-pooling layers before flattening and then concatenated its output with the last flattened convolutional layer. Then the remaining is the same as VGG.



**DATA**

We can see that the data used in the paper is not rich enough. There are obvious data imbalances in the three categories of CXR images. Hence we used “covid19-radiography-database”.

* Here we have (Covid, Normal, Pneumonia) in quantity (552-708-222) in Test,

(2041- 2724-618) in Train and (500-695-203) in Validation.

* Because of the lack of proper data, some data enhancement methods were used in the project.

**IMPLEMENTATION**

During Implementation to get better results we used few optimization techniques.

|  |
| --- |
| import time  import tensorflow as tf  from tensorflow import keras  from tensorflow.keras import datasets  from tensorflow.keras import models, optimizers  from tensorflow.keras.layers import Input,Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Concatenate,GlobalMaxPooling2D  from tensorflow.keras.metrics import CategoricalAccuracy, Recall, Precision |

While loading the image, we added Labeling Data, then read images, apply normalization, and finally applied image enhancement to tackle overfitting.

We used a three-stage learning rate when training the model, that is, a smaller learning rate is used in the initial stage, and a larger preset learning rate is used in the intermediate stage after the model is stabilized, and at the later stage of training, to help the model further converge, a smaller learning rate is used.

|  |
| --- |
| def load\_image(img\_path,size=(80,80)):    if tf.strings.regex\_full\_match(img\_path, ".\*COVID19.\*"):  label = tf.constant(0, tf.uint8)  elif tf.strings.regex\_full\_match(img\_path, ".\*NORMAL.\*"):  label = tf.constant(1, tf.uint8)  else:  label = tf.constant(2, tf.uint8)  label = tf.one\_hot(label,depth=3, on\_value=1,off\_value=0 )  img = tf.io.read\_file(img\_path)  img = tf.io.decode\_jpeg(img,channels=3)  img = tf.cast(img, tf.float32) / 255.0  if tf.strings.regex\_full\_match(img\_path, ".\*train.\*"):  img = tf.image.random\_flip\_left\_right(img)  img = tf.image.random\_brightness(img,0.1)  img = tf.image.random\_contrast(img,0.1,0.2)  img = tf.image.random\_saturation(img,0,5)  img = tf.image.resize(img,size=(96,96))  img = tf.image.random\_crop(img,size=[80,80,3])  img = tf.image.resize(img, size)  return (img, label)  def scheduler(epoch, lr=0.0001):  if epoch < 3:  lr=0.0004  elif epoch <6:  lr=0.0002  else:  lr=0.0001  return lr |

This is our model as explained in modified VGG.

|  |
| --- |
| def myModel():  inputs = Input(shape=(80,80,3))  conv1 = Conv2D(32, (2, 2), padding='same', activation='elu')(inputs)  conv1 = Conv2D(32, (2, 2), padding='same', activation='elu')(conv1)  pool1 = MaxPooling2D((2, 2),strides=(2,2))(conv1)  conv2 = Conv2D(64, (2, 2), padding='same', activation='elu')(pool1)  conv2 = Conv2D(64, (2, 2), padding='same', activation='elu')(conv2)  pool2 = MaxPooling2D((2, 2),strides=(2,2))(conv2)  conv3 = Conv2D(128, (2, 2),padding='same', activation='elu')(pool2)  conv3 = Conv2D(128, (2, 2),padding='same', activation='elu')(conv3)  conv3 = Conv2D(128, (2, 2),padding='same', activation='elu')(conv3)  pool3 = MaxPooling2D((2, 2),strides=(2,2))(conv3)  conv4 = Conv2D(128, (2, 2),padding='same', activation='elu')(pool3)  conv4 = Conv2D(128, (2, 2),padding='same', activation='elu')(conv4)  conv4 = Conv2D(128, (2, 2),padding='same', activation='elu')(conv4)  pool4 = MaxPooling2D((2, 2),strides=(2,2))(conv4)  concat = Concatenate()(  [GlobalMaxPooling2D()(pool1),  GlobalMaxPooling2D()(pool2),  GlobalMaxPooling2D()(pool3),  GlobalMaxPooling2D()(pool4)])  flatten1 = Flatten()(concat)  flatten2 = Flatten()(conv4)  concat2 = Concatenate()([flatten1,flatten2])  x = Dense(4096, activation='elu')(concat2)  x = Dropout(0.5)(x)  x = Dense(2048, activation='elu')(x)  x = Dropout(0.4)(x)  outputs = Dense(3, activation='softmax')(x)  return models.Model(inputs=inputs,outputs=outputs) |

This is our main function. Since the weights of the model in the initial epochs of training are randomly initialized, the model may be unstable if using a large learning rate. Therefore, we used the Warmup method, using a small learning rate in the first few epochs or steps of the training, and then using the preset learning rate for training after the model is relatively stable, which makes the model convergence speed faster and more effective. In the later stage of training, the loss of the model is reduced to a low level. At this time, if the original larger learning rate is still used, it may cause "oscillation". The learning rate should be gradually attenuated to reduce the model gradient to the local minimum as soon as possible.

We also used label smoothing during training. It is a regularization strategy.

|  |
| --- |
| if \_\_name\_\_ == "\_\_main\_\_":  BATCH\_SIZE = 32  tensorboard\_callback = tf.keras.callbacks.TensorBoard(log\_dir="./logs")  train\_data = tf.data.Dataset.list\_files("./train/\*/\*.\*")\  .map(load\_image, num\_parallel\_calls=tf.data.experimental.AUTOTUNE) \  .shuffle(buffer\_size=1280).batch(BATCH\_SIZE) \  .prefetch(tf.data.experimental.AUTOTUNE)  validation\_data = tf.data.Dataset.list\_files("./validation/\*/\*.\*")\  .map(load\_image, num\_parallel\_calls=tf.data.experimental.AUTOTUNE) \  .shuffle(buffer\_size=1280).batch(BATCH\_SIZE) \  .prefetch(tf.data.experimental.AUTOTUNE)  test\_data = tf.data.Dataset.list\_files("./test/\*/\*.\*")\  .map(load\_image, num\_parallel\_calls=tf.data.experimental.AUTOTUNE) \  .batch(BATCH\_SIZE) \  .prefetch(tf.data.experimental.AUTOTUNE)  # new\_model = tf.keras.models.load\_model('./covid19.h5') #same file path  # new\_model.evaluate(covid\_data)  # new\_model.summary()    model = myModel()  loss = tf.keras.losses.CategoricalCrossentropy(label\_smoothing=0.12)  adam = tf.keras.optimizers.Adam(learning\_rate=0.0004)  scheduler\_callback = tf.keras.callbacks.LearningRateScheduler(scheduler)  start = time.time()  model.compile(  loss=loss,  optimizer=adam,  metrics=[CategoricalAccuracy(name="accuracy"),Recall(),Precision()]  )  history=model.fit(train\_data,  epochs=15,  class\_weight={0:2.5,1:3.18,2:1},  validation\_data=validation\_data,  callbacks=[tensorboard\_callback,scheduler\_callback]  )  # model.evaluate(test\_data)  # model.evaluate(covid\_data)  end = time.time()  print(f"Runtime of the program is {end - start}")  model.summary()  model.save('covid19-32f.h5') |

The Model is saved in /covid19-32f.h5 file, which is then used to evaluate test data

|  |
| --- |
| new\_model = tf.keras.models.load\_model('./covid19-32f.h5')  model.evaluate(test\_data) |

Here we extract the output of each layer. Then Creates a model that will return these outputs, given the model input. Returns a list of five Numpy arrays: one array per layer activation. And try to print a single image of one-layer filters.

|  |
| --- |
| BATCH\_SIZE= 32  my\_data = tf.data.Dataset.list\_files("./my/\*.\*")\  .map(load\_image, num\_parallel\_calls=tf.data.experimental.AUTOTUNE) \  .batch(BATCH\_SIZE) \  .prefetch(tf.data.experimental.AUTOTUNE) |

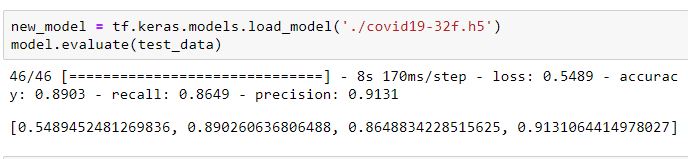
|  |
| --- |
| from keras.callbacks import ModelCheckpoint  layer\_outputs = [layer.output for layer in new\_model.layers[:12]]  activation\_model = models.Model(inputs=new\_model.input, outputs = layer\_outputs)  activations = activation\_model.predict(my\_data)  first\_layer\_activation = activations[0]  plt.matshow(first\_layer\_activation[0, :, :,1], cmap='viridis') |

|  |
| --- |
| layer\_names = []  for layer in new\_model.layers[:12]:  layer\_names.append(layer.name)  images\_per\_row = 14  for layer\_name, layer\_activation in zip(layer\_names, activations):  n\_features = layer\_activation.shape[-1]  size = layer\_activation.shape[1]  n\_cols = n\_features // images\_per\_row  for col in range(n\_cols):  for row in range(images\_per\_row):  channel\_image = layer\_activation[0,  : , :,  col \* images\_per\_row + row]  channel\_image -= channel\_image.mean()  channel\_image /= channel\_image.std()  channel\_image \*= 64  channel\_image += 128  channel\_image = np.clip(channel\_image, 0, 255).astype('uint8')  display\_grid[col \* size : (col + 1) \* size, # Displays the grid  row \* size : (row + 1) \* size] = channel\_image  scale = 2.0 / size  plt.figure(figsize=(scale \* display\_grid.shape[1],  scale \* display\_grid.shape[0]))  plt.title(layer\_name)  plt.grid(False)  plt.imshow(display\_grid, aspect='auto', cmap='viridis') |

**RESULT**

We used VGG Model to predict whether a person has Covid or Not using an X-ray image. We have modified the VGG model and also added Optimisation code as a Scheduler where we are changed the learning rate as no. of epochs are increasing.

We have got 89.30 % accuracy 86.49% recall and 91.31% precision. Also, we found that accuracy becomes constant after 10 epochs.



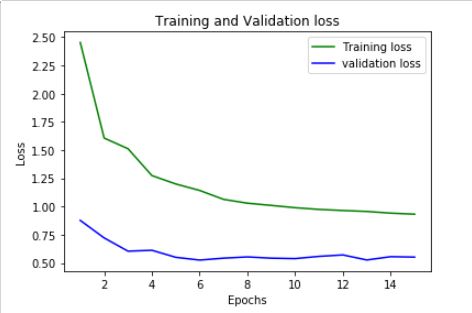
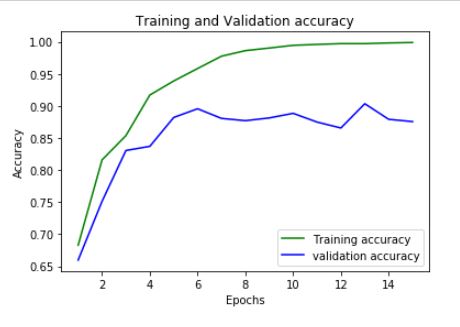
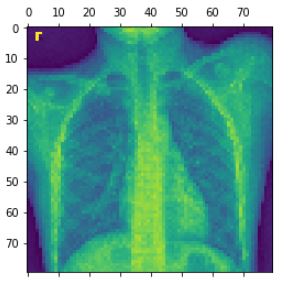


Fig 2. Graphs showing Accuracy and Loss of training and validation data.



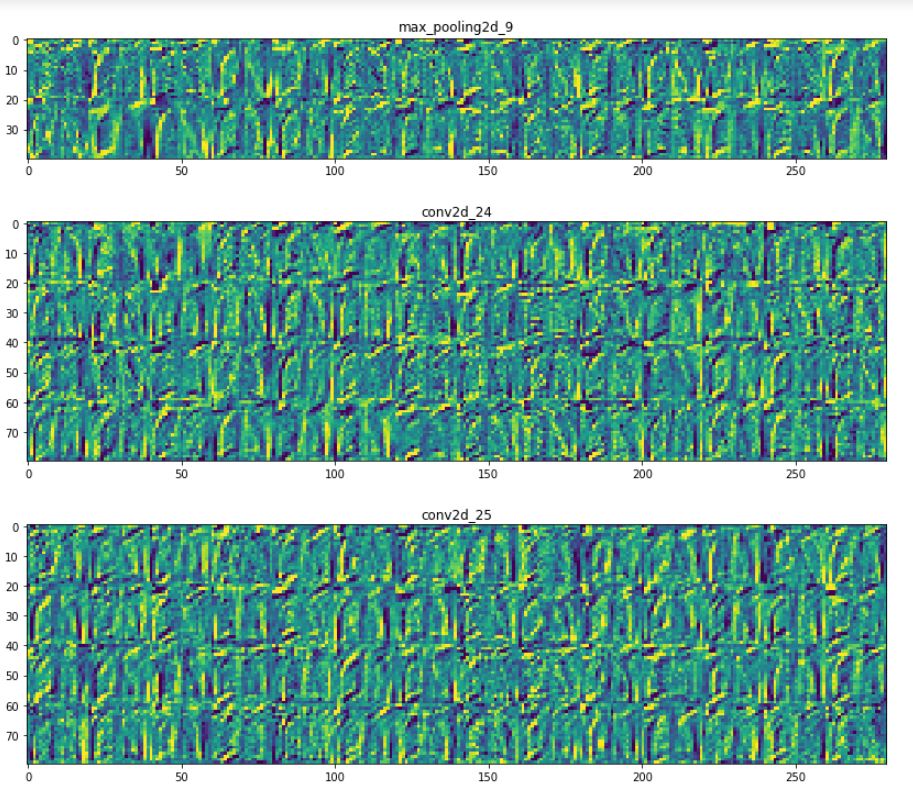
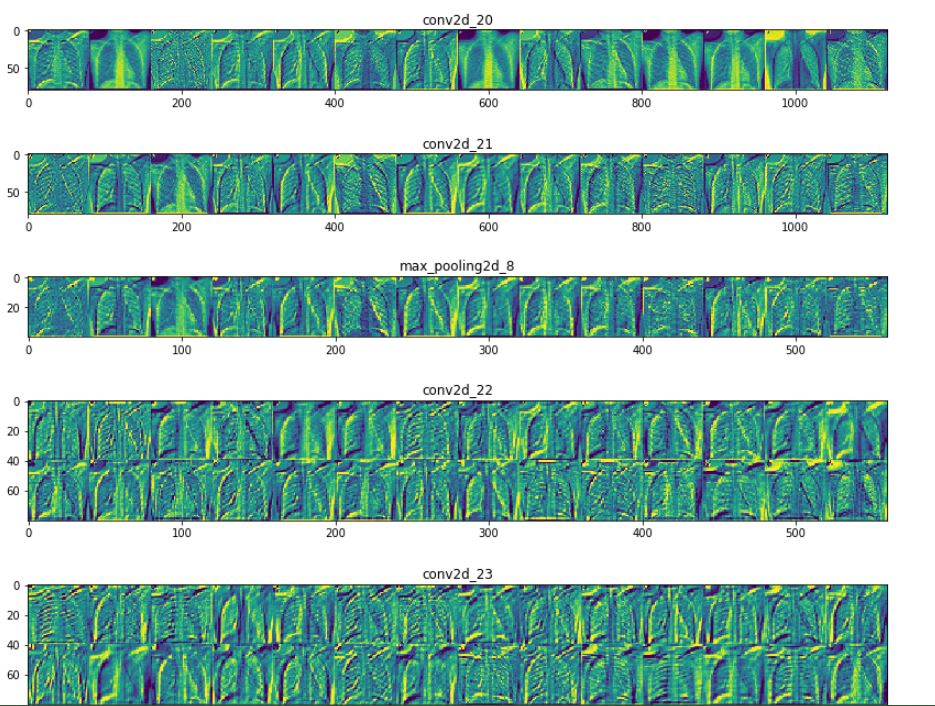


Fig 3. visualization of VGG layers.

CONCLUSION

This project may serve as a reference for other researchers aiming to advance the development of deep learning applications in medical imaging. Here we were able to successfully classify X-Ray images into Covid, normal, and pneumonia patients. This was done with good accuracy of about 89%. A better accuracy could be obtained if we used a bigger image in input as we used only 80x80 pixel images and more filters in convolutional layers. This was avoided as it would require a lot of time to train a huge dataset along with more filters.

References

1. <https://becominghuman.ai/what-is-the-vgg-neural-network-a590caa72643>
2. <https://arxiv.org/abs/2007.09695>
3. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53