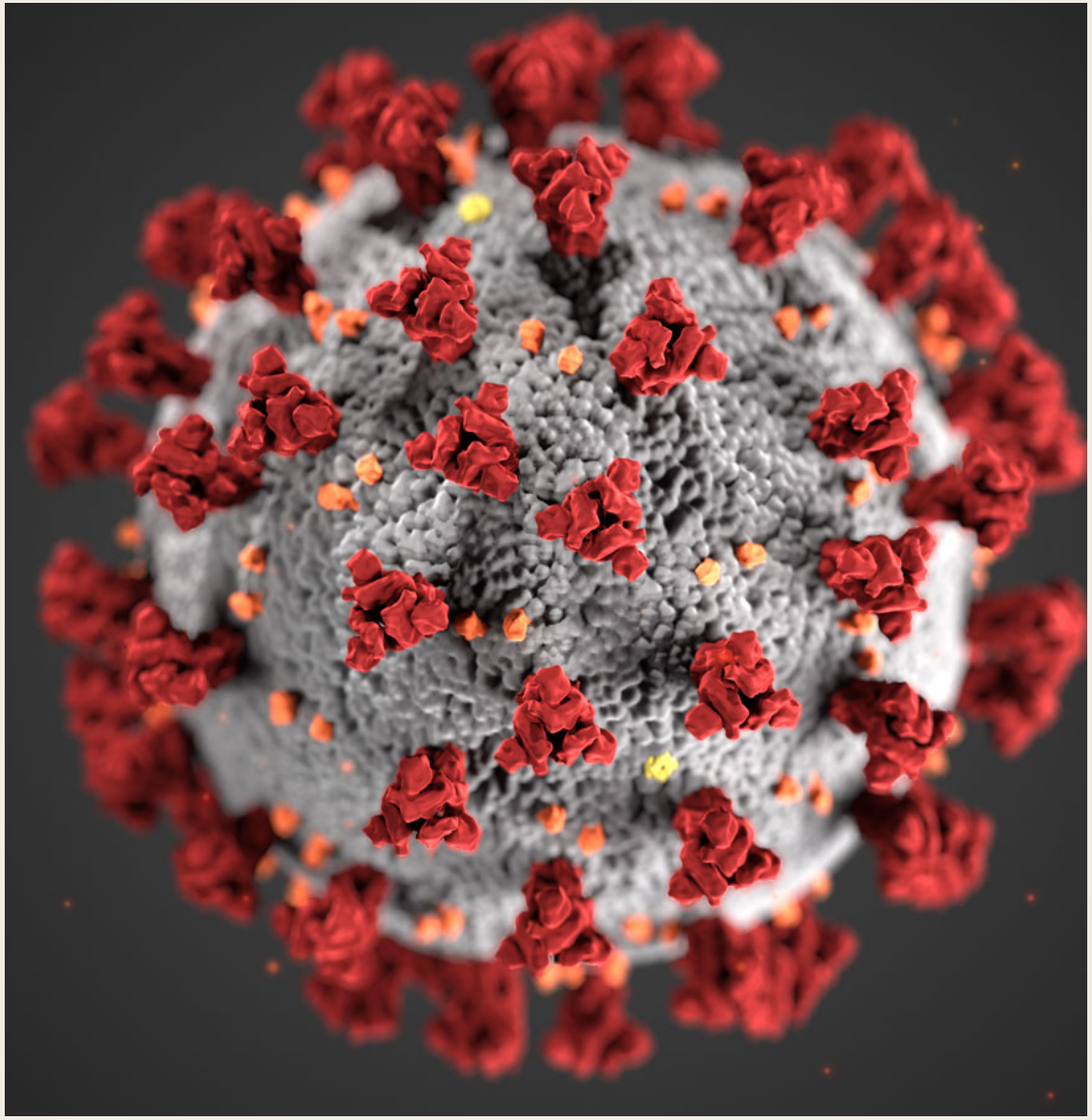
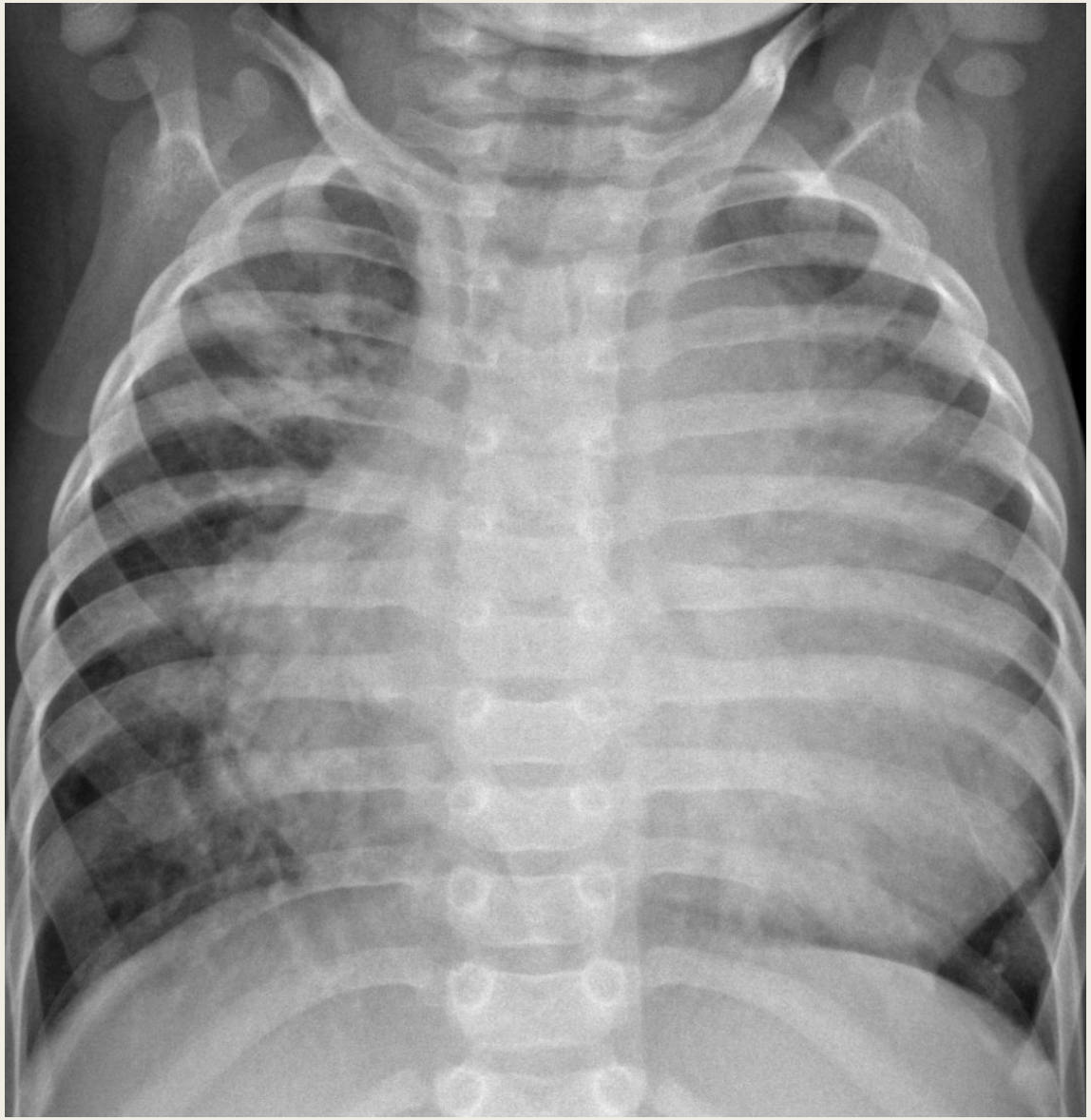
ML PNEUMONIA DETECTION

Sujeethan Vigneswaran
3rd Year Environmental Engineering Student at UW
QHacks 2021

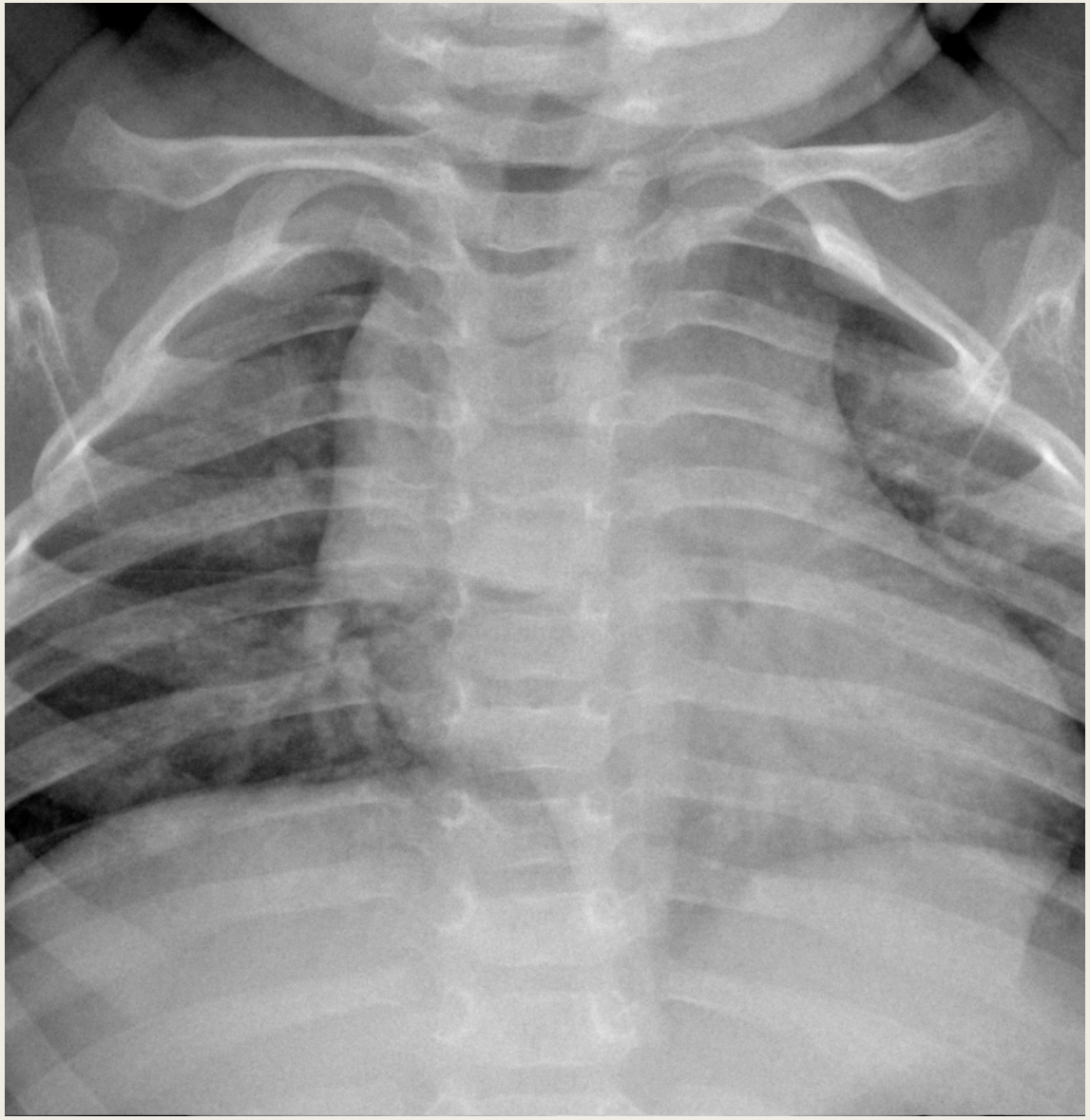


COVID-19 CHANGED THE WORLD

- Hospitals packed
- Medical workers over worked
- Pneumonia is a common comorbidity
- Opportunity to use technology to help medical workers!



Pneumonia X-Ray



Normal X-Ray

DID YOU CATCH THAT?

Yeah… I didn’t either

TensorFlow CNN
X-Ray Classification
Powered by DCL

DEMO TIME!

Setup DCP

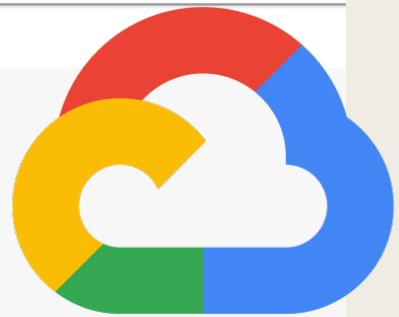
```
%%capture
!npm install -g n && n 10.20.1
!pip install git+https://github.com/Kings-Distributed-Systems/Bifrost
from bifrost import npm, node
```

```
%%capture
npm.install('dcp-client')
```

```
# UPLOAD DCP KEYSTORE FILE
# from google.colab import files
# KEYSTORE_NAME = list(files.upload().keys())[0]
# !mkdir -p ~/.dcp && cp /content/$KEYSTORE_NAME ~/.dcp/id.keystore && cp ~/.dcp/id.keystore
```

```
%%node

require('dcp-client').initSync();
const compute = require('dcp/compute');
const dcpCli = require('dcp/dcp-cli');
```

Create Training & Validation Datasets



Google Cloud

[1]
```
import re
import os
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
strategy = tf.distribute.get_strategy()
GCS_PATH = "gs://dcp-dataset"
IMAGE_SIZE = [180, 180]
BATCH_SIZE = 16 * strategy.num_replicas_in_sync
EPOCHS = 25
```

[3]
```
filenames = tf.io.gfile.glob(str(GCS_PATH + '/chest_xray/train/*/*'))
filenames.extend(tf.io.gfile.glob(str(GCS_PATH + '/chest_xray/val/*/*')))
# Split training and validation data 80:20
train_filenames, val_filenames = train_test_split(filenames, test_size=0.2)
```

[4]
```
train_list_ds = train_filenames
val_list_ds = val_filenames
```

```
[ ]
```

```
#@markdown ### DCP Job Parameters

#@markdown Maximum runtime allowed for the job, in minutes:
max_runtime = 30 #@param {type:"slider", min:10, max:120, step:10}

train_ds = [];
val_ds = [];
```

DCP Job Parameters

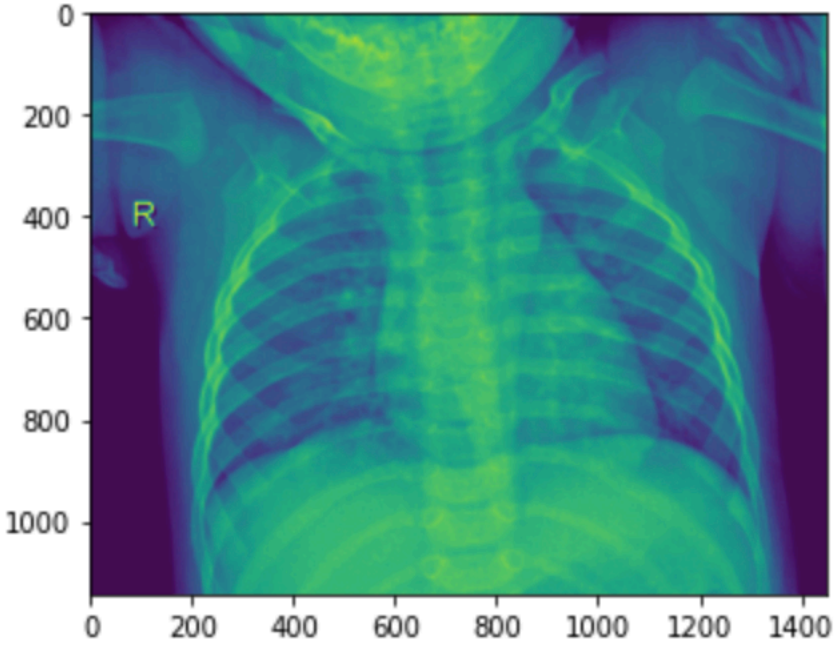
Maximum runtime allowed for the job, in minutes:

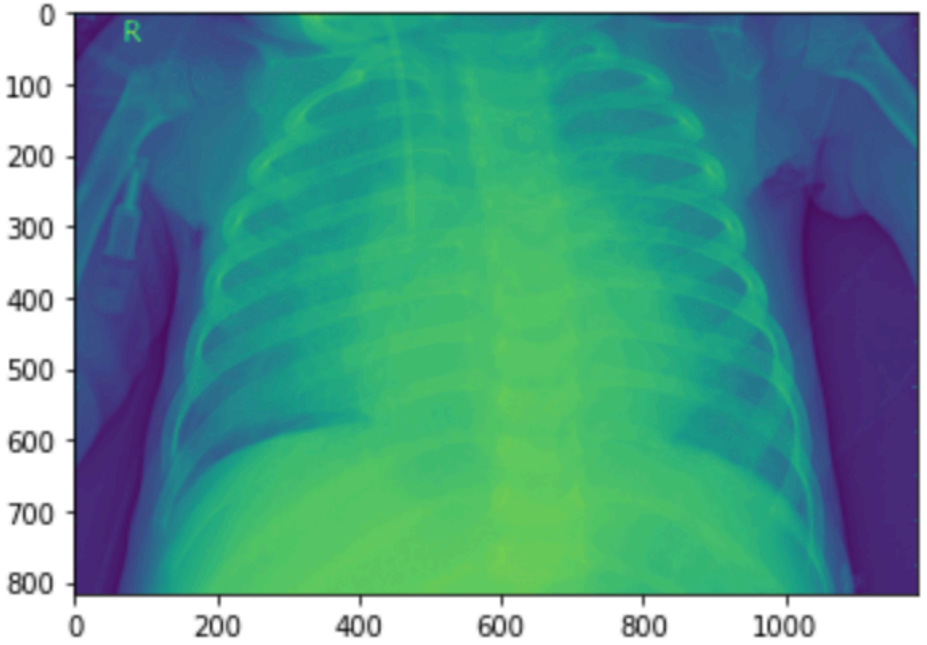
max_runtime:

```
%%node
// # Training Dataset
// # Call functions to decode images, and deploy them to the Distributed Computer for processing in parallel

deployTime = Date.now();

postTrainJob(train_list_ds, max_runtime).then((value) => {
    console.log('Job complete.');

    let finalTime = Date.now() - deployTime;

    console.log('Total time to compute:');
    console.log((finalTime / 1000).toFixed(2) + ' seconds.');
});

postTrainJob(val_list_ds, max_runtime).then((value) => {
    console.log('Job complete.');

    let finalTime = Date.now() - deployTime;

    console.log('Total time to compute:');
    console.log((finalTime / 1000).toFixed(2) + ' seconds.');
```

Dataset Viewing

```
train_list_ds = tf.data.Dataset.from_tensor_slices(train_filenames)
val_list_ds = tf.data.Dataset.from_tensor_slices(val_filenames)

def get_label(file_path):
    # convert the path to a list of path components
    parts = tf.strings.split(file_path, os.path.sep)
    # The second to last is the class-directory
    return parts[-2] == "PNEUMONIA"

def decode_img(img):
  # convert the compressed string to a 3D uint8 tensor
  img = tf.image.decode_jpeg(img, channels=3)
  # Use `convert_image_dtype` to convert to floats in the [0,1] range.
  img = tf.image.convert_image_dtype(img, tf.float32)
  # resize the image to the desired size.
  return tf.image.resize(img, IMAGE_SIZE)

def process_path(file_path):
    label = get_label(file_path)
    # load the raw data from the file as a string
    img = tf.io.read_file(file_path)
    img = decode_img(img)
    return img, label

train_ds = train_list_ds.map(process_path, num_parallel_calls=AUTOTUNE)
val_ds = val_list_ds.map(process_path, num_parallel_calls=AUTOTUNE)
```





Build the CNN

```
# Big Chungus Model & Hyperparameter Tuning Created by Amy Jang from Google

def conv_block(filters):
    block = tf.keras.Sequential([
        tf.keras.layers.SeparableConv2D(filters, 3, activation='relu', padding='same'),
        tf.keras.layers.SeparableConv2D(filters, 3, activation='relu', padding='same'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPool2D()
    ]
    )
    return block

def dense_block(units, dropout_rate):
    block = tf.keras.Sequential([
        tf.keras.layers.Dense(units, activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(dropout_rate)
    ])
    return block

def build_model():
    model = tf.keras.Sequential([
        tf.keras.Input(shape=(IMAGE_SIZE[0], IMAGE_SIZE[1], 3)),

        tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same'),
        tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same'),
        tf.keras.layers.MaxPool2D(),
```

```
history = model.fit(
    train_ds,
    steps_per_epoch=TRAIN_IMG_COUNT // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=val_ds,
    validation_steps=VAL_IMG_COUNT // BATCH_SIZE,
    class_weight=class_weight
)
```

```
Epoch 1/25
261/261 [==============================] - 544s 2s/step -
Epoch 2/25
261/261 [==============================] - 311s 1s/step -
Epoch 3/25
261/261 [==============================] - 316s 1s/step -
Epoch 4/25
261/261 [==============================] - 312s 1s/step -
Epoch 5/25
261/261 [==============================] - 314s 1s/step -
Epoch 6/25
261/261 [==============================] - 312s 1s/step -
Epoch 7/25
261/261 [==============================] - 314s 1s/step -
Epoch 8/25
261/261 [==============================] - 311s 1s/step -
Epoch 9/25
261/261 [==============================] - 312s 1s/step -
Epoch 10/25
261/261 [==============================] - 312s 1s/step -
Epoch 11/25
261/261 [==============================] - 311s 1s/step -
Epoch 12/25
261/261 [==============================] - 312s 1s/step -
Epoch 13/25
```

PRETTY GOOD!

```
_________________________________________________________________
39/39 [==============================] - 41s 1s/step - loss: 0.8909 - accuracy: 0.6763 - precision: 0.9896 - recall: 0.48
Restored model, accuracy: 67.63%
```