

Pneumonia Classifier

Richard Scott McNew

A02077329

CS 6600: Intelligent Systems

Objective

Develop and train an image classification network to predict a pneumonia diagnosis when given a chest X-ray image. Use tools other than tflearn in order to explore and learn how to employ other machine learning tools.

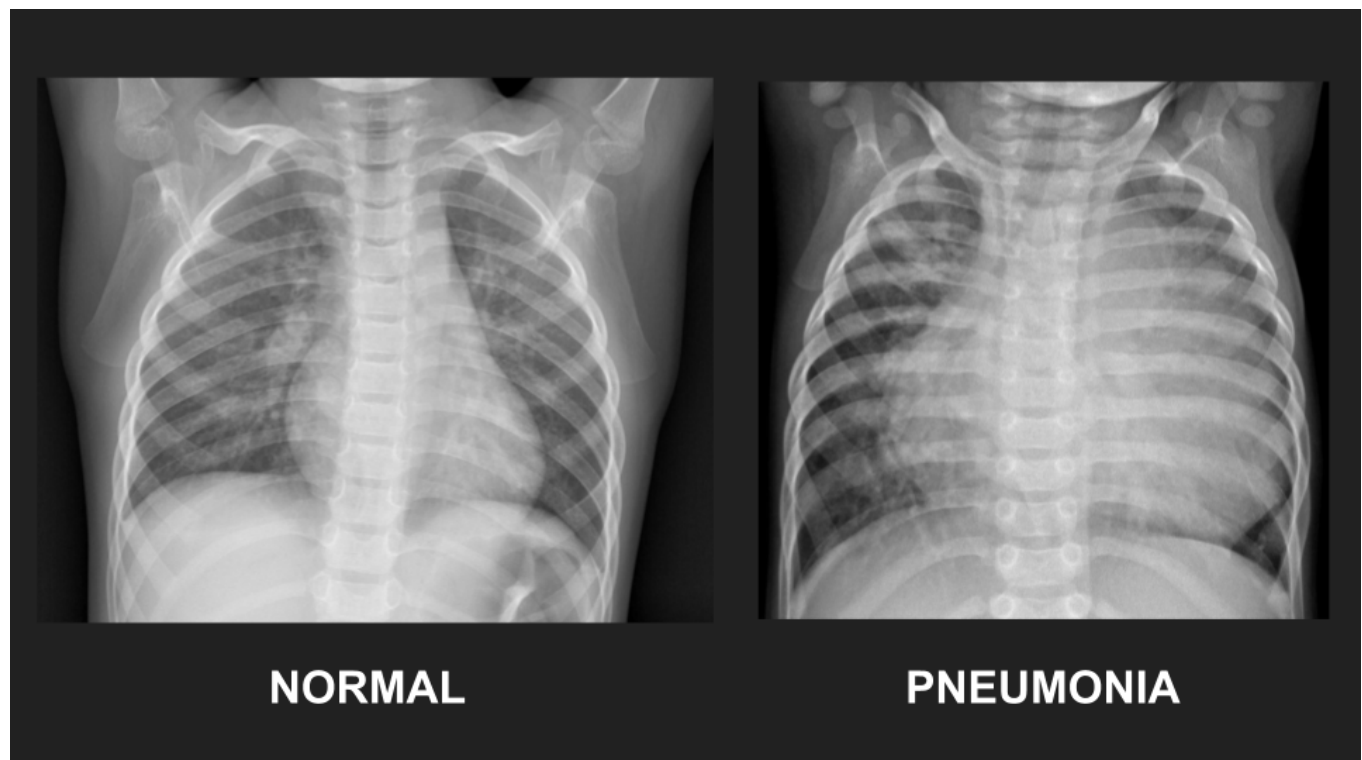
Motivation

Pneumonia is the number one killer of children under 5 years old world-wide, killing about 2 million children every year (Rudan et al., 2008). It is estimated that pediatric pneumonia kills more children than HIV/AIDS, malaria, and measles combined (Adegbola, 2012).

Practically all of these pediatric pneumonia cases occur in developing countries throughout Southeast Asia and Africa. Pediatric pneumonia can be caused by both bacterial or viral pathogens (Mcluckie, 2009). Timely and accurate diagnosis and follow-on treatment are essential to ensure the survival of affected children.

Chest X-rays are a primary means of diagnosing pneumonia, but rapid interpretation of chest X-rays are frequently not available in the developing nations where pediatric pneumonia is common and mortality rates are high.

This Pneumonia Classifier takes a chest X-ray image as input and provides a high-confidence automated diagnosis as to whether the patient has pneumonia or not.



Dataset

The chest X-ray dataset consists of 5856 chest X-ray images of various dimensions. Images are labelled as one of two sets: NORMAL or PNEUMONIA.

Dataset Source URL: <https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia/>

Dataset Image Counts

<u>Dataset Subset</u>	<u>NORMAL</u>	<u>PNEUMONIA</u>
Train	1341	3875
Test	234	390
Validate	8	8

Dataset Image Preprocessing

The dataset images must be preprocessed before they are ready for use in training the convolution networks. The following preprocessing actions are taken for all of the images before they are processed:

- Converted to grayscale
- Resized to 450 x 450 dimensions
- Numpy array values for each pixel are rescaled to [0, 1] range

The Train dataset subset images also have the following transformations randomly applied to compensate for the relatively small number of dataset images (dataset augmentation) and to help prevent overfitting:

- Images may be randomly rotated up to plus or minus 10 degrees (this is given by the *rotation_range* parameter)
- Images may be randomly translated horizontally or vertically up to 10% of the image width or height (this is given in the *width_shift* and *height_shift* parameters)
- Images may be randomly shear transformed up to 5% (this is given in the *shear_range* parameter)
- Images may be randomly zoomed to only show up to 20% portion of the original image (this is given in the *zoom_range* parameter)

The following images show the how randomly applying the above transformations to training images adds variety to the input data:



Image source: https://blog.keras.io/img/imgclrf/cat_data_augmentation.png

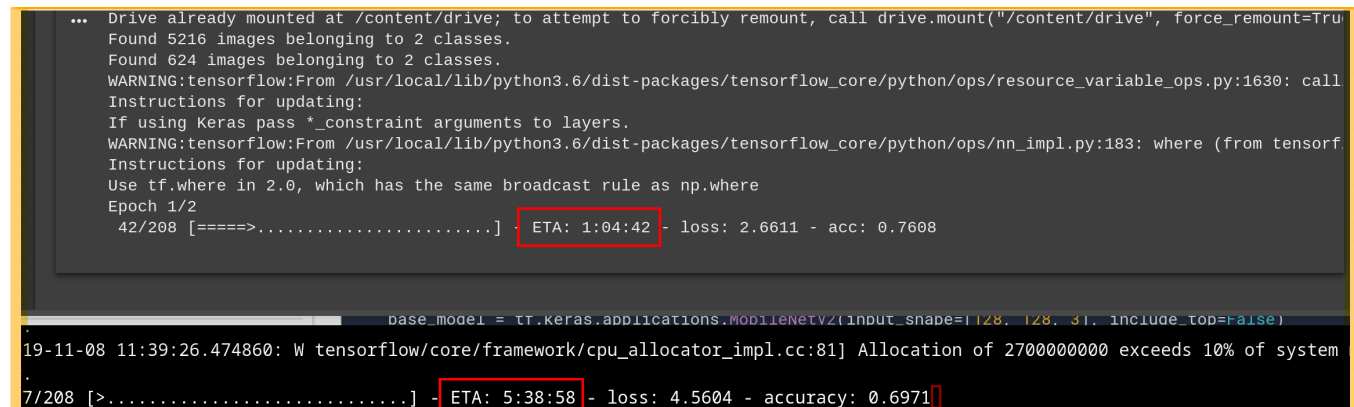
Tooling

After some exploration, I choose to use Google Colaboratory as the primary tool to train the Pneumonia Classifier convolution neural networks. This was largely motivated by the limited hardware that I have available to perform convolution net training.

Google Colaboratory Advantages

Google Colaboratory offers a no-cost, moderately powerful, GPU-accelerated virtual machine with more processing power and RAM than my laptop (25 GB RAM on the virtual machine versus 16 GB RAM on my laptop). The virtual machine is accessed through a Jupyter Notebook web page with Google additions that make it ideal for machine learning and scientific computing exploration.

Google Colaboratory enabled me to train the Pneumonia Classifier convolution neural networks much faster and get feedback on my design decisions more rapidly than I could with my laptop. This screenshot shows a training run with the same dataset and convolution neural network hyperparameters running on a Google Colaboratory instance and my laptop. (The Google Colaboratory instance is above. My laptop is below.)



```
... Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
Found 5216 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: call
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/nn_impl.py:183: where (from tensorf
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Epoch 1/2
42/208 [====>.....] - ETA: 1:04:42 - loss: 2.6611 - acc: 0.7608

base_model = tf.keras.applications.MobileNetV2(input_shape=(128, 128, 3), include_top=False)
19-11-08 11:39:26.474860: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 2700000000 exceeds 10% of system
7/208 [>.....] - ETA: 5:38:58 - loss: 4.5604 - accuracy: 0.6971
```

Note that the Colaboratory instance is completing training epoch 1 much faster than my laptop. The Colaboratory instance estimates one hour and four minutes left compared to the five hours and thirty-eight minutes left on my laptop. This is due to a faster virtual machine processor, more RAM, and GPU-acceleration.

Google Colaboratory offers a ready-to-go Python-powered machine learning platform based on Jupyter Notebook that requires practically no setup and runs in the web browser. The pre-installed libraries are Tensorflow-centric, but also include popular Python numerical (numpy, scipy), image processing (opencv, scikit-image), graph visualization (matplotlib, plotly), and machine learning (tensorflow, keras, tflearn, pandas, scikit-learn, pytorch) libraries. This makes it possible to do a wide range of data transformation and visualization without extensive setup.

Google Colaboratory Disadvantages

Google Colaboratory instances on the no-cost tier are intended primarily for interactive use and have a timeout of 90 minutes, meaning that the Colaboratory notebook will be automatically disconnected from a backend virtual machine if no activity occurs in the notebook. This effectively prevents brute-force searches of the hyperparameters since it means that long-running training of multiple epochs is more difficult to accomplish without "baby-sitting" the notebook. I found this to not be too disadvantageous since it encouraged me to monitor the training more closely and notice immediately when signs of overfitting began to be manifest.

Although it does not occur very often, sometimes the Google Colaboratory virtual machine instances can be flaky and will exhibit transitory failures. This usually manifests as unexpected out-of-memory failures or much slower training times. In these cases, it was best to either restart the runtime ("Runtime" -> "Restart Runtime" in the Menu) or disconnect from the "lemon" virtual machine backend and let the session timeout. Following

the restart and connecting to a different virtual machine backend the problems disappear and no changes are needed to the code being run. If "Out of Memory" errors continue to occur, this usually means that the model or dataset input data is too large and needs to be scaled down.

Machine Learning Framework Choice

The Keras APIs that are included with Tensorflow 2.X are used for image processing, model creation, and model training. Tensorflow 2.x and Keras are well-supported on Google Colaboratory and take can advantage of GPU acceleration for rapid model training.

Convolution Neural Network Architecture

I studied the architecture of several high performance image classification convolution neural networks to find ideas about how to design a suitable convolution network that would be effective a pneumonia classification and still be able to train on the Colaboratory virtual machine and ideally run on commodity hardware.

Many of the high performance image classification convolution neural networks (VGG16, VGG19, Xception, and many others) use several convolution layers followed by a single max pooling layer. I decided to follow a similar design and iterated until I found designs that were a good match for the chest X-ray image dataset and be able to train without encountering "Out of Memory" errors. The following diagram shows the final convolution neural network architecture that I arrived at after several iterations. The corresponding model code is given in the `create_model` function below.

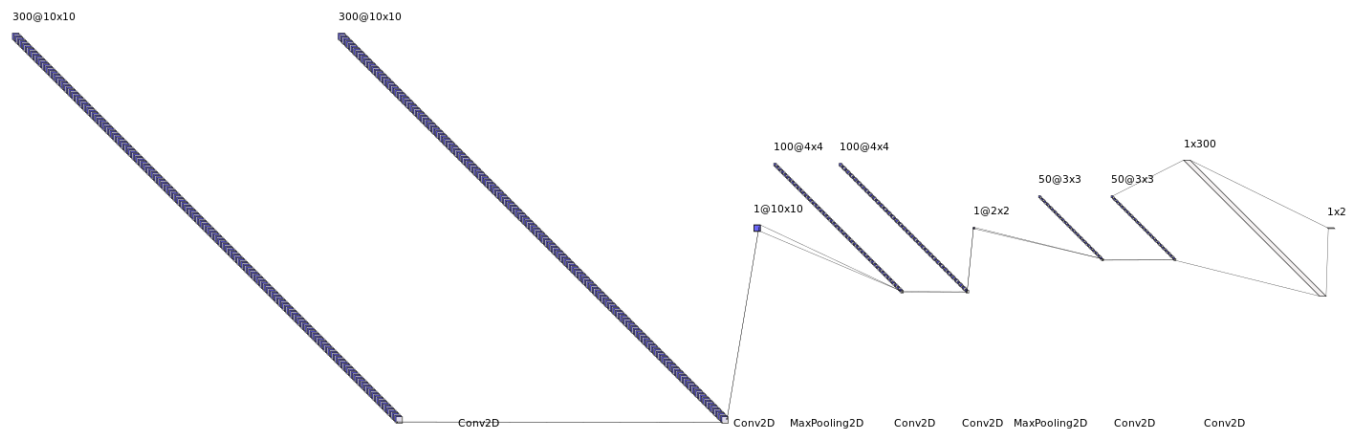


Diagram created using [NN-SVG](#).

Code for Training and Testing the Convolution Nets

```

In [0]: # Richard Scott McNew
        # A02077329
        # CS 6600: Intelligent Systems

        # use tensorflow 2.x
        %tensorflow_version 2.x
        from __future__ import absolute_import, division, print_function, unicode_literals
        from google.colab import drive, files
        import datetime
        import io
        import os
        import pathlib
        from time import sleep
        import tensorflow as tf
        import numpy as np
        from tensorflow import keras
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
        from tensorflow.keras.preprocessing.image import ImageDataGenerator
        import matplotlib.pyplot as plt

        # enable accelerated linear algebra
        tf.config.optimizer.set_jit(True)
        # enable tensorflow AUTOTUNE
        AUTOTUNE = tf.data.experimental.AUTOTUNE

        # Load the TensorBoard notebook extension for training metrics graphs
        %load_ext tensorboard

        ##### Constants #####
        BATCH_SIZE = 8 # Use small batches to allow current batch to fit in GPU memory for faster training
        IMAGE_SIDE = 450 # We have to resize the dataset images to reduce memory consumption
        SHUFFLE_SIZE = 25 # We shuffle the images to ensure a better fit
        EPOCHS = 1 # Only run a few epochs at a time since Colaboratory times out without interactive use

        # ensure we are in the correct working directory
        os.chdir("/content")

        ##### Dataset Download and Path Construction #####
        ##
        # Local path to the dataset
        DATASET_PATH = "/content/Pneumonia_Classifier/dataset"

        # There is a copy of the Pneumonia dataset in my Pneumonia_Classifier Github repo
        # We can clone the 'dataset_only' branch to get a local copy
        def get_dataset_files_from_github():
            if not os.path.isdir(DATASET_PATH):
                print("Downloading the dataset from Github . . .")

```

```

        !git clone -b dataset_only https://github.com/rmcnew/Pneumonia_Classifier.git
    else:
        print("Using previously downloaded dataset")

# There is a tarball of the Pneumonia dataset available as a publicly shared link
# from my Google Drive account. This is probably the fastest way to download a
# local copy of the dataset since it should be all within Google's networks
def get_dataset_files_from_google_drive_shared():
    if not os.path.isdir(DATASET_PATH):
        print("Downloading the dataset from Google Drive shared link . . .")
        !wget https://drive.google.com/uc?id=1u2_Ap4r0xHuEKnSb5te070skuoXcJTX9
        print("Download completed! Untarring the dataset . . .")
        !tar xjf Pneumonia_Classifier_dataset.tar.bz2
        print("Dataset is ready!")
    else:
        print("Using previously downloaded dataset")

# Download a local copy of the dataset and then build paths
# to the different dataset subsets: 'train', 'test', and 'validate'
get_dataset_files_from_github()
get_dataset_files_from_google_drive_shared()
dataset = pathlib.Path(DATASET_PATH)
test = dataset.joinpath("test")
test_count = len(list(test.glob('**/*.jpeg')))
train = dataset.joinpath("train")
train_count = len(list(train.glob('**/*.jpeg')))
validate = dataset.joinpath("validate")
validate_count = len(list(validate.glob('**/*.jpeg')))

##### Dataset Preprocessing #####
###
# The train_image_generator applies random transformations to the
# Train dataset subset to provide dataset augmentation since the dataset
# is small and we want to avoid overfitting
def create_train_image_generator():
    train_image_generator = ImageDataGenerator(
        rescale=1./255,
        rotation_range=10,
        width_shift_range=0.1,
        height_shift_range=0.1,
        zoom_range=0.2,
        shear_range=0.05,
        fill_mode='nearest')
    train_data_gen = train_image_generator.flow_from_directory(
        batch_size=BATCH_SIZE,
        directory=str(train),
        shuffle=True,
        target_size=(IMAGE_SIDE, IMAGE_SIDE),
        color_mode="grayscale",

```

```

        class_mode='categorical')
    return train_data_gen

# Prepare images in the Test dataset subset for use
def create_test_image_generator():
    test_image_generator = ImageDataGenerator(rescale=1./255)
    test_data_gen = test_image_generator.flow_from_directory(
        batch_size=BATCH_SIZE,
        directory=str(test),
        target_size=(IMAGE_SIDE, IMAGE_SIDE),
        color_mode="grayscale",
        class_mode='categorical')
    return test_data_gen

# Prepare images in the Validate dataset subset for use
def create_validate_image_generator():
    validate_image_generator = ImageDataGenerator(rescale=1./255)
    validate_data_gen = validate_image_generator.flow_from_directory(
        batch_size=BATCH_SIZE,
        directory=str(validate),
        target_size=(IMAGE_SIDE, IMAGE_SIDE),
        color_mode="grayscale",
        class_mode='categorical')
    return validate_data_gen

##### Model Creation, Loading, and Saving ###
#####
### This model tends to approach the upper limit of available memory
of the Colaboratory virtual machine.
### Larger models with more layers or greater numbers of features ru
n into "Out of Memory" errors
def create_model():
    model = Sequential([
        Conv2D(300, 10, padding='same', activation='relu', kernel_re
gularizer='l2',
            input_shape=(IMAGE_SIDE, IMAGE_SIDE, 1)),
        Conv2D(300, 10, padding='same', activation='relu', kernel_re
gularizer='l2'),
        MaxPooling2D(10),
        Dropout(0.2),
        Conv2D(100, 4, padding='same', activation='relu', kernel_reg
ularizer='l2'),
        Conv2D(100, 4, padding='same', activation='relu', kernel_reg
ularizer='l2'),
        MaxPooling2D(),
        Dropout(0.2),
        Conv2D(50, 3, padding='same', activation='relu', kernel_regu
larizer='l2'),
        Conv2D(50, 3, padding='same', activation='relu', kernel_regu
larizer='l2'),
        MaxPooling2D(),
        Flatten(),
        Dense(300, activation='relu'),
        Dense(2, activation='softmax')
    ])

```



```

        model.compile(optimizer='SGD', loss='categorical_crossentropy',
metrics=['accuracy'])
        return model

##### Trained model from my Google Drive shared link #####
# There is the trained persisted model that is shared from my Google
Drive account.
# This model can be downloaded by anyone who has the shared link UR
L.
# The 'gdown' command line tool is used to perform non-interactive d
ownloads.
# This is the 'final' trained model that is trained to be as accurat
e as possible.

# Download a copy of the trained model to evaluate it against the Te
st or Validate
# dataset subsets or to predict using an uploaded chest X-ray image
def download_trained_model_from_google_drive_shared(download_anyway=
False):
    MODEL_PATH = "/content/pneumonia_classifier_model.h5"
    if not os.path.exists(MODEL_PATH) or download_anyway:
        print("Downloading trained model from Google Drive shared li
nk . . .")
        !gdown https://drive.google.com/uc?id=1cux0s6RXWNT3eGjqSbGiy
Mvp_jhBekq8
        print("Download completed! Loading the model . . .")
    else:
        print("Using previously downloaded model")
    model = tf.keras.models.load_model(MODEL_PATH, custom_objects=No
ne, compile=True)
    print("Model loaded.")
    return model

# Save the trained model and then create a web browser dialogue to d
ownload it
# This can take a bit to run and download since the model occupies a
bout 700 MB on disk
def download_trained_model_via_browser(model):
    TMP_PATH = '/tmp/pneumonia_classifier_model.h5'
    model.save(TMP_PATH, overwrite=True, include_optimizer=True, sav
e_format='h5')
    print("Preparing model for browser download. Please wait as it
could take a while . . . .")
    sleep(2) # pause for two seconds for the file to save
    files.download(TMP_PATH)

##### Google Drive functions #####
#####
# These functions are only needed if you want to load a trained netw
ork from your
# Google Drive storage or save a trained network to your Google Driv
e storage.

# Note that these functions will require interactive steps (opening
a page in a
# web browser and copying / pasting the authoriztaion code) to creat

```

```

e OAuth
# tokens that will be used to authorize access to your Google Drive
  account.
# The URL to authorize access is given in the output of this cell (a
  t the bottom)
# when these functions are called to access Google Drive.

# mount Google drive
def mount_drive():
    drive.mount('/content/drive', force_remount=True)

# unmount Google drive
def unmount_drive():
    drive.flush_and_unmount()

# Google Drive path to the saved model
# Note that you may encounter an error if this path does not exist i
  n your Google Drive account
GOOGLE_DRIVE_MODEL_PATH = "/content/drive/My Drive/USU/intelligent_s
  ystems/Pneumonia_Classifier/pneumonia_classifier_model.h5"

def save_model_to_google_drive(model):
    print("Saving model to Google Drive")
    mount_drive()
    model.save(GOOGLE_DRIVE_MODEL_PATH, overwrite=True, include_opti
  mizer=True, save_format='h5')
    unmount_drive()

def load_trained_model_from_google_drive():
    print("Loading saved model from Google Drive")
    mount_drive()
    model = tf.keras.models.load_model(GOOGLE_DRIVE_MODEL_PATH, cust
  om_objects=None, compile=True)
    return model

##### Model Training and Evaluation #####
#####
# Create the initial version of the model and run some training epoc
  hs
# After training epochs run, save the trained model to Google Drive
  or download it
def train_model():
    print("Training model from scratch")
    print("Preparing Train and Test dataset subsets")
    train_data_gen = create_train_image_generator()
    test_data_gen = create_test_image_generator()
    model = create_model()
    log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%
  H%M%S")
    tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=lo
  g_dir, histogram_freq=1)
    print("Training model . . .")
    history = model.fit(
        train_data_gen,
        steps_per_epoch=train_count // BATCH_SIZE,
        epochs=EPOCHS,

```

```

        validation_data=test_data_gen,
        validation_steps=test_count // BATCH_SIZE,
        callbacks=[tensorboard_callback]
    )
    save_model_to_google_drive(model)

# Load the previously trained model from Google Drive, run more training
# epochs, and then save the more trained model back to Google Drive
def train_model_more():
    print("Training model more")
    print("Preparing training and testing datasets")
    train_data_gen = create_train_image_generator()
    test_data_gen = create_test_image_generator()
    model = load_trained_model_from_google_drive()
    log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
    tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
    print("Training model more . . .")
    history = model.fit(
        train_data_gen,
        steps_per_epoch=train_count // BATCH_SIZE,
        epochs=EPOCHS,
        validation_data=test_data_gen,
        validation_steps=test_count // BATCH_SIZE,
        callbacks=[tensorboard_callback]
    )
    save_model_to_google_drive(model)

# Download the trained model from the Google Drive shared link
# and then evaluate the trained model's accuracy against the
# Test dataset subset
def test_trained_model():
    print("Running model against Test dataset subset")
    test_data_gen = create_test_image_generator()
    model = download_trained_model_from_google_drive_shared()
    model.evaluate(test_data_gen)

# Download the trained model from the Google Drive shared link
# and then evaluate the trained model's accuracy against the
# Validate dataset subset
def validate_trained_model():
    print("Running model against Validate dataset subset")
    validate_data_gen = create_validate_image_generator()
    model = download_trained_model_from_google_drive_shared()
    model.evaluate(validate_data_gen)

##### Main Section #####
# These function calls will need to be
# commented out or uncommented depending
# on what you want to do

#train_model()          # <-- Run to train the model from scratch and save the trained model to YOUR Google Drive

```

```

#train_model_more()      # <-- Run to load a trained model from YOUR
Google Drive and train it more
test_trained_model()     # <-- Run to see accuracy against Test datas
et subset; MY trained model is used
validate_trained_model() # <-- Run to see accuracy against Validate
dataset subset; MY trained model is used
The tensorboard extension is already loaded. To reload it, use:
%reload_ext tensorboard
Using previously downloaded dataset
Running model against Test dataset subset
Found 624 images belonging to 2 classes.
Downloading trained model from Google Drive shared link . . .
Downloading...
From: https://drive.google.com/uc?id=1cux0s6RXWNT3eGjqSbGiyMVp_jhBek
q8
To: /content/pneumonia_classifier_model.h5
46.3MB [00:00, 53.4MB/s]
Download completed! Loading the model . . .
Model loaded.
78/78 [=====] - 55s 701ms/step - loss: 0.27
91 - accuracy: 0.9151
Running model against Validate dataset subset
Found 16 images belonging to 2 classes.
Using previously downloaded model
Model loaded.
2/2 [=====] - 2s 1s/step - loss: 0.3412 - a
ccuracy: 0.9375

```

TensorBoard Graphs

TensorBoard graphs can be used to visualize the convolution network training metrics and determine if any changes need to be made. The graphs are derived from training metric logs which are enabled via *tensorboard_callback* functions above.

Running the cells below will build dynamic graphs from the previous training runs.

```

In [0]: # Delete training metric logs from previous training runs
# These logs are used by TensorBoard to create the training metrics
graphs
# so DO NOT run this unless you want to delete the old logs.
!rm -rf ./logs/

```

```

In [0]: # Run me to see the training metric graphs for the convolution
# net training that you just ran. Note that the logs are created
# on the virtual machine backend and are not persisted unless you
# do something to keep the training metric logs.

# For some reason TensorBoard needs to be run in its own cell, so
# this cell is apart from the rest of the convolution net training
# code above
%tensorboard --logdir logs/fit

```

Download Non-dataset Chest X-Ray Images to Run Against the Trained Model

[other_xrays.zip](#)

Upload and Run a Chest X-Ray Image Against the Trained Model

Run the following code block to upload a chest X-ray image from your web browser and get a pneumonia prediction.

Use the "Choose Files" button in the output pane to select an image file to upload. If you get the error message, "Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable." then re-run the code block and it should succeed.

```

In [0]: %tensorflow_version 2.x
from __future__ import absolute_import, division, print_function, unicode_literals
from google.colab import files
import os
import shutil
import tempfile
from time import sleep
import tensorflow as tf
import numpy as np
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator

IMAGE_SIDE = 450 # We have to resize the dataset images to reduce memory consumption

def download_trained_model_from_google_drive_shared(download_anyway=False):
    MODEL_PATH = "/content/pneumonia_classifier_model.h5"
    if not os.path.exists(MODEL_PATH) or download_anyway:
        print("Downloading trained model from Google Drive shared link . . .")
        !gdown https://drive.google.com/uc?id=1cux0s6RXWNT3eGjqSbGiyMvp_jhBekq8
        print("Download completed! Loading the model . . .")
    else:
        print("Using previously downloaded model")
    model = tf.keras.models.load_model(MODEL_PATH, custom_objects=None, compile=True)
    print("Model loaded.")
    return model

def upload_images_and_save_files():
    files.upload()

# create an image generator for the uploaded images
def create_image_generator(dir_name):
    print("Creating image generator for images in directory: ", dir_name)
    image_generator = ImageDataGenerator(rescale=1./255)
    data_gen = image_generator.flow_from_directory(
        directory=str(dir_name),
        target_size=(IMAGE_SIDE, IMAGE_SIDE),
        color_mode="grayscale",
        class_mode=None) # class_mode has to be None to indicate there are no data labels
    return data_gen

# Round the prediction to a human-readable interpretation
def interpret_prediction(prediction):
    print("Prediction is: {}".format(prediction))
    pred_list = prediction[0].tolist()
    max_index = pred_list.index(max(pred_list))
    if max_index == 0:
        print("NORMAL")
    else:

```

```
print("PNEUMONIA")

##### Main section #####
model = download_trained_model_from_google_drive_shared()
# create temporary directory
temp_dir_name = tempfile.mkdtemp()
print('Created temporary directory: ', temp_dir_name)
# cd to temporary directory
os.chdir(temp_dir_name)
# make subdirectory for image files; per the Keras documentation:
# "Please note that in case of class_mode None, the data still needs
# to reside in a subdirectory of directory for it to work correctly."
image_path = os.path.join(temp_dir_name, "images")
os.mkdir(image_path)
os.chdir(image_path)
# upload image files from user's web browser into image_path
upload_images_and_save_files()
sleep(2) # pause to save files to disk
# preprocess uploaded images
image_gen = create_image_generator(temp_dir_name)
# run prediction
prediction = model.predict(image_gen)
# interpret and print out result
interpret_prediction(prediction)
# return to starting directory
os.chdir("/content")
# clean-up temporary directory
shutil.rmtree(temp_dir_name)
```

Performance Results

The following table shows how the trained models performed against the Test and Validate dataset subsets:

<u>Dataset Subset</u>	<u>Accuracy</u>
Test	91.51%
Validate	93.75%

Download Trained Model

The final trained model is available on GitHub: https://github.com/rmcnew/Pneumonia_Classifier/raw/master/pneumonia_classifier_model.h5

Or you can run this cell to download the trained model from my Google Drive shared link.

```
In [0]: ##### Trained model from my Google Drive shared link #####
# There is the trained persisted model that is shared from my Google
# Drive account.
# This model can be downloaded by anyone who has the shared link UR
# L.
# The 'gdown' command line tool is used to perform non-interactive d
# ownloads.
# This is the 'final' trained model that is trained to be as accurat
# e as possible.

# Download a copy of the trained model to evaluate it against the Te
# st or Validate
# dataset subsets or to predict using an uploaded chest X-ray image
def download_trained_model_from_google_drive_shared(download_anyway=
False):
    MODEL_PATH = "/content/pneumonia_classifier_model.h5"
    if not os.path.exists(MODEL_PATH) or download_anyway:
        print("Downloading trained model from Google Drive shared li
nk . . .")
        !gdown https://drive.google.com/uc?id=1cux0s6RXWNT3eGjqSbGiy
MVp_jhBekq8
        print("Download completed! Loading the model . . .")
    else:
        print("Using previously downloaded model")
    model = tf.keras.models.load_model(MODEL_PATH, custom_objects=No
ne, compile=True)
    print("Model loaded.")
    return model

# Save the trained model and then create a web browser dialogue to d
# ownload it
# This can take a bit to run and download since the model occupies a
# bout 700 MB on disk
def download_trained_model_via_browser(model):
    TMP_PATH = '/tmp/pneumonia_classifier_model.h5'
    model.save(TMP_PATH, overwrite=True, include_optimizer=True, sav
e_format='h5')
    print("Preparing model for browser download. Please wait as it
could take a while . . . .")
    sleep(2) # pause for two seconds for the file to save
    files.download(TMP_PATH)

# Main section
model = download_trained_model_from_google_drive_shared(download_any
way=True)
download_trained_model_via_browser(model)
```

Ideas for Fielding the Pneumonia Classifier

The trained Pneumonia Classifier convolution network is about 45 megabytes on disk and uses less than a gigabyte of RAM when run under Python. This means that it can run on commodity hardware and deployed for field use as part of a mobile radiography lab to assist in "on-the-spot" diagnoses. For example, it could easily be included on the radiography laptop computer that is a component of the Mobile Diagnostic Unit described by Dhoot et al (Dhoot R, et al., 2018).

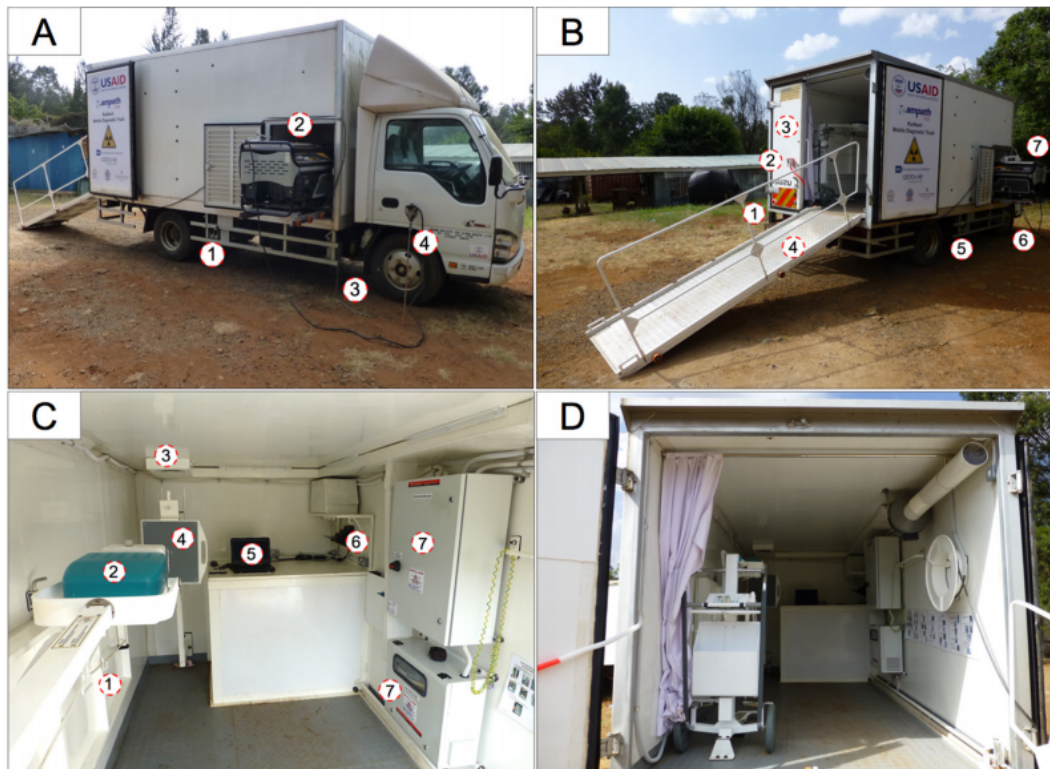


Figure 2 Mobile diagnostic unit equipment placement and descriptions. (A) Anterior side view: (1) fuel can; (2) slide-out electrical generator; (3) grounding chain; (4) grounding rod (hammered into the ground at each site). (B) Posterior side view: (1) standing platform for radiologists; (2) power factor correction device; (3) shielded door for radiologists to stand behind; (4) patient access ramp; (5) onboard generator fuel can; (6) redundant grounding devices (installed at each site); (7) slide-out electrical generator. (C) (interior view, detailed): (1) fold-down bed for supine radiographs; (2) X-ray generator; (3) negative pressure ventilation; (4) digital radiography cassette; (5) radiography laptop computer; (6) onboard 230 V power and charging; (7) power control, safety and distribution systems. (D) Interior view, general layout, shown with shielding door open.

Image source: <https://gh.bmj.com/content/bmjgh/3/5/e000947.full.pdf>

The Pneumonia Classifier would help to fill a critical shortage in the rapid interpretation of chest X-Ray results. Dhoot et al state: "A primary radiologist reading, which is the standard of care throughout the USA, is challenging in Kenya due to multiple factors including poor internet connectivity, too few radiologists and delivery time for paper reports. Poor internet connectivity in rural areas creates delays in archiving images for radiologist review and precludes clinicians' access to the online results portal." If the Pneumonia Classifier were to be deployed with the Mobile Diagnostic Unit or a similar mobile field radiography system it could provide an immediate preliminary field screening that would help patients get the care they need more rapidly.

References

- Kermany et al., 2018. "Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning", Cell 172, p. 1122–1131, February 22, 2018. Elsevier Inc. <https://doi.org/10.1016/j.cell.2018.02.010>
- Rudan, I., Boschi-Pinto, C., Biloglav, Z., Mulholland, K., and Campbell, H. (2008). "Epidemiology and Etiology of Childhood Pneumonia" Bulletin of the World Health Organization 86, 408–416.
- Adegbola, R.A. (2012). "Childhood Pneumonia as a Global Health Priority and the Strategic Interest of the Bill & Melinda Gates Foundation". Clinical Infectious Diseases 54 (Supplement 2), S89–S92.
- McLuckie, A. (2009). "Respiratory disease and its management", Volume 57 (Springer).
- Dhoot R, et al. (2018). "Implementing a Mobile Diagnostic Unit to Increase Access to Imaging and Laboratory Services in Western Kenya" BMJ Glob Health 2018; 3:e000947. doi:10.1136/bmjgh-2018-000947 Available at <https://gh.bmj.com/content/bmjgh/3/5/e000947.full.pdf>

License

Pneumonia_Classifier: a machine learning image classifier for pediatric chest X-rays to help detect pneumonia

Copyright (C) 2019 Richard Scott McNew.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.