Run some Covid-19 lung X-Ray classification and CT detection demos

Article

Zhong Li · Apr 16, 2020 12m read

Run some Covid-19 lung X-Ray classification and CT detection demos

Keywords: COVID-19, Medical Imaging, Deep Learning, PACS Viewer, and HealthShare.

Purpose

We are all gripped by this unprecedented Covid-19 pandemic. While supporting our customers in battlefields by any means, we also observed various fighting fronts against Covid-19 by leveraging today's AI powers.

Last year I briefly touched a deep learning demo environment. Why don't we jump into some real-world images during this long Easter Weekend, to test run some deep learning models on Covid-19 lungs' X-Ray dataset for quick classifications, and to witness how such tools for X-Ray and even CTs are rapidly deployed into cloud via docker etc for prompt "AI Triage" as well as for "Radiologist's assistance".

This is just another 10-minute quick note, hoping to help get hands dirty by the simplest possible approach along the journey.

Scope

The following components are used in this demo environment. They are the simplest form I can find so far:

- A small **anonymised open dataset** of 3 types: Covid-19 lungs vs. Bacterial Pneumonia lungs vs. normal clear lungs.
- A set of **deep learning models** such as i.e. Inception V3 model(s) for X-Ray lung classification(s)
- A container of **Tensorflow 1.13.2** with Jupyter Notebook
- A container of **Nvidia-Docker2 for GPU** tooling
- An AWS **Ubuntu 16.04 VM** with a Nvidia T4 GPU (a laptop's GPU would be enough if no re-training of pre-trained models)

And,

- A demo container of "AI assisted CT detection".
- A demo container of a 3rd party Open PACS Viewer.
- A demo instance of HealthShare Clinical Viewer.

The following are NOT in demo scope:

- PyTorch is getting more popular (we will use it next time)
- Tensorflow 2.0 running much slower in the demo environment (so I revert back to version 1.13 for now)
- AutoML etc multi-model ensembles (they are getting popular in real-world but a single old-fashioned model here is enough for this small dataset)
- X-Ray and CTs data from any real-world sites.

Disclaimer

This demo is more about technology approaches than clinical trials in this specific domain. Covid-19 dictation based on CT & X-Ray etc evidences are widely available online now, with both positive and negative reviews, and it actually plays different roles in various countries and cultures during this pandemic. Also, the content and layout of this post might be revised along as needed. The content is a purely personal view as a "developer".

Data

The original images in this test came from a publicly available Covid-19 Lung X-Ray set by Joseph Paul Cohen, and some clean lungs from the open Kaggle Chest X-Ray sets, collected into a small test set by Adrian Yu at the GradientCrescent repository. I also uploaded test data here for quick test of anyone's interests. It so far only contains a small training set of:

- 60x Covid-19 lungs
- 70x normal clear lungs
- 70x Bacterial Pneumonia lungs

Test

For the tests below, I ran the following with slight change of my own flavors:

- Inception V3 model as base plus a couple of CNN layers as tops
- Transfer Learning with unfrozen weights of underlining Inception layers for re-training (if on a laptop GPU you can simply freeze the pre-trained inception layers)
- Slight augmentation to compensate the small data set collected so far.
- 3x categories instead of binary: Covid-19 vs. Normal vs Bacterial (or Viral) Pneumonia (I will explain why these 3 classes)
- Calculating a basic 3-class confusion matrix as a template for possible later steps.

Note: there is no particularly reason why InceptionV3 is chosen over other popular CNN based models such as VGG16 or ResNet50 etc. I happened to use it for demo run another bone fracture data set recently among other models so just re-use it here. You can use any such models you prefer to re-run the following Jupyter Notebook scripts.

I attached the following Jupyter Notebook file within this post too. Below is for quick illustration too.

1. Import the necessary libraries:

```
# import the necessary packages
from tensorflow.keras.layers import AveragePooling2D, Dropout, Flatten, Dense, Input
from tensorflow.keras.models import Model
```

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import optimizers, models, layers
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.resnet50 import ResNet50

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os
```

2. Load the sample image files provided here

```
# set learning rate, epochs and batch size
INIT_LR = 1e-5
# This value is specific to what model is chosen: Inception, VGG or ResNet etc.
EPOCHS = 50
BS = 8

print("Loading images...")
imagePath = "./Covid_M/all/train"  # change to your local path for the sample images
imagePaths = list(paths.list_images(imagePath))

data = []
labels = []

# read all X-Rays in the specified path, and resize them all to 256x256
for imagePath in imagePaths:
    label = imagePath.split(os.path.sep)[-2]
    image = cv2.imread(imagePath)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (256, 256))
    data.append(image)
    labels.append(label)

#normalise pixel values to real numbers between 0.0 - 1.0
data = np.array(data) / 255.0
labels = np.array(labels)

# perform one-hot encoding for a multi-class labeling
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(labels)
labels = to_categorical(integer_encoded)

print("... ... ", len(data), "images loaded in multiple classes:")
print(label_encoder.classes_)



Loading images...
... ...  200 images loaded in 3x classes:
['covid' 'normal' 'pneumonia_bac']
```

3. Add in basic data augment, re-compose the model, then train it

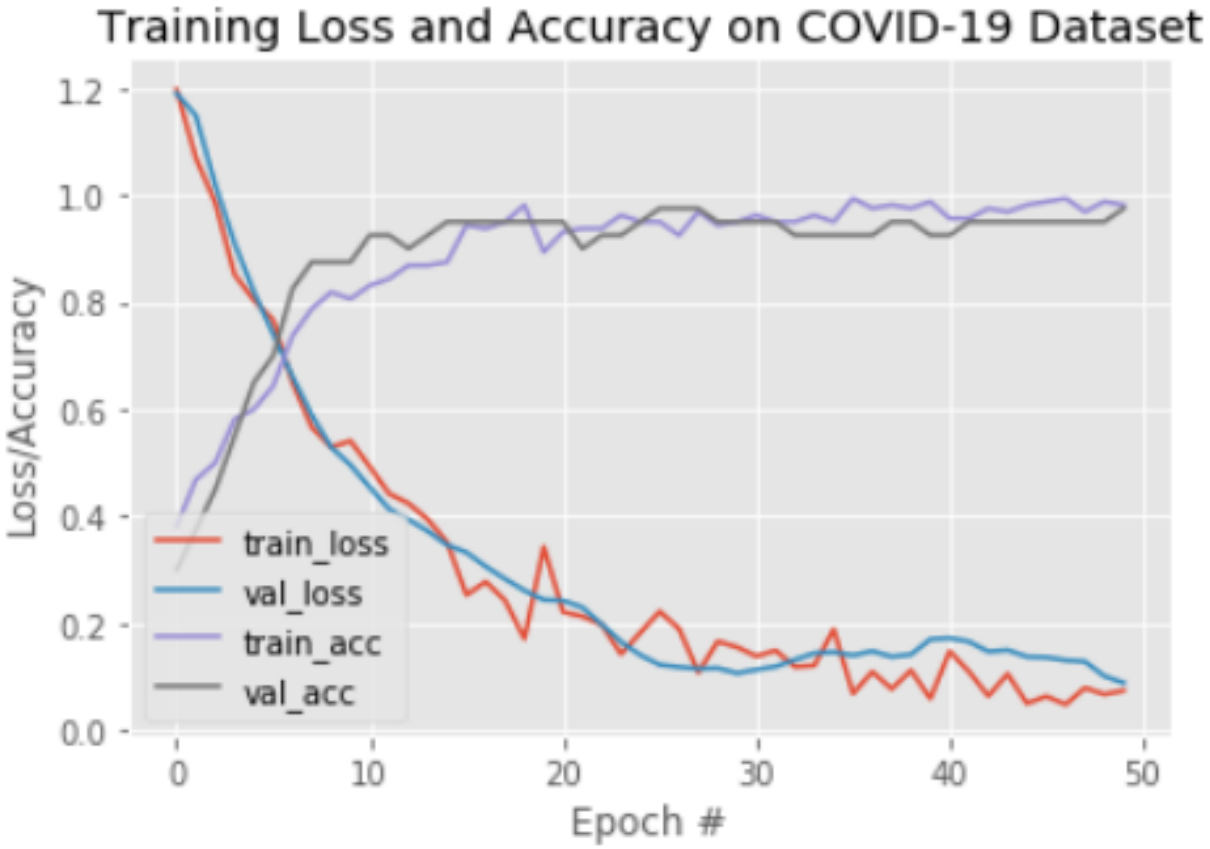
```
# split the data between train and validation.
(trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.20, strat
ify=labels, random_state=42)

# add on a simple Augmentation. Note: too many Augumentation doesn't actually help in
 this case - I found during the test.
trainAug = ImageDataGenerator(rotation_range=15, fill_mode="nearest")

#Use the InveptionV3 model with Transfer Learning of pre-
trained "ImageNet"'s weights.
#note: If you choose VGG16 or ResNet you may need to reset the initial learning rate
at the top.
baseModel = InceptionV3(weights="imagenet", include_top=False, input_tensor=Input(sha
pe=(256, 256, 3)))
#baseModel = VGG16(weights="imagenet", include_top=False, input_tensor=Input(shape=(2
56, 256, 3)))
#baseModel = ResNet50(weights="imagenet", include_top=False, input_tensor=Input(shape
=(256, 256, 3)))

#Add on a couple of custom CNN layers on top of the Inception V3 model.
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(3, activation="softmax")(headModel)

# Compose the final model
model = Model(inputs=baseModel.input, outputs=headModel)

# Unfreeze pre-trained Inception "ImageNet" weights for re-
training since I got a Navidia T4 GPU to play with anyway
#for layer in baseModel.layers:
#    layer.trainable = False

print("Compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])

# train the full model, since we unfroze the pre-trained weights above
print("Training the full stack model...")
H = model.fit_generator( trainAug.flow(trainX, trainY, batch_size=BS), steps_per_epoc
h=len(trainX) // BS,
                         validation_data=(testX, testY), validation_steps=len(testX)
// BS, epochs=EPOCHS)



... ...
Compiling model...
Training the full stack model...
... ...
Use tf.cast instead.
Epoch 1/50
40/40 [==============================] - 1s 33ms/sample - loss: 1.1898 - acc: 0.3000
```

```
20/20 [==============================] - 16s 800ms/step - loss: 1.1971 - acc: 0.3812
- val_loss: 1.1898 - val_acc: 0.3000
Epoch 2/50
40/40 [==============================] - 0s 6ms/sample - loss: 1.1483 - acc: 0.3750
20/20 [==============================] - 3s 143ms/step - loss: 1.0693 - acc: 0.4688 -
 val_loss: 1.1483 - val_acc: 0.3750
Epoch 3/50
... ...
... ...
Epoch 49/50
40/40 [==============================] - 0s 5ms/sample - loss: 0.1020 - acc: 0.9500
20/20 [==============================] - 3s 148ms/step - loss: 0.0680 - acc: 0.9875 -
 val_loss: 0.1020 - val_acc: 0.9500
Epoch 50/50
40/40 [==============================] - 0s 6ms/sample - loss: 0.0892 - acc: 0.9750
20/20 [==============================] - 3s 148ms/step - loss: 0.0751 - acc: 0.9812 -
 val_loss: 0.0892 - val_acc: 0.9750
```

4. Plot confusion matrix for the validation results:

```
print("Evaluating the trained model ...")
predIdxs = model.predict(testX, batch_size=BS)

predIdxs = np.argmax(predIdxs, axis=1)

print(classification_report(testY.argmax(axis=1), predIdxs, target_names=label_encode
r.classes_))

# calculate a basic confusion matrix
cm = confusion_matrix(testY.argmax(axis=1), predIdxs)
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1] + cm[2, 2]) / total
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1] + cm[0, 2])
specificity = (cm[1, 1] + cm[1, 2] + cm[2, 1] + cm[2, 2]) / (cm[1, 0] + cm[1, 1] + cm
[1, 2] + cm[2, 0] + cm[2, 1] + cm[2, 2])

# show the confusion matrix, accuracy, sensitivity, and specificity
print(cm)
print("acc: {:.4f}".format(acc))
print("sensitivity: {:.4f}".format(sensitivity))
print("specificity: {:.4f}".format(specificity))

# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["acc"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_acc"], label="val_acc")
plt.title("Training Loss and Accuracy on COVID-19 Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig("./Covid19/s-class-plot.png")
```

```
Evaluating the trained model ...
              precision    recall  f1-score   support

       covid       1.00      1.00      1.00        12
      normal       1.00      0.93      0.96        14
pneumonia_bac       0.93      1.00      0.97        14

    accuracy                           0.97        40
   macro avg       0.98      0.98      0.98        40
weighted avg       0.98      0.97      0.97        40

[[12  0  0]
 [ 0 13  1]
 [ 0  0 14]]
acc: 0.9750
sensitivity: 1.0000
specificity: 1.0000
```



From the above figure, it can be seen that with the benefit of "transfer learning" even with a small dataset and a quick training of less than 5 minutes, the result is not that bad: all 12x Covid-19 Lungs are classified correctly, and only 1x Normal lung out of 40 in total is classified wrongly as "Bacterial Pneumonia" lung.

5. Plot confusion matrix for testing some real X-rays

Now why can't we leap one step further - send in some real X-Rays to test how effective this lightly trained classifier could be.

So I uploaded 27x X-Ray images into the model that were not yet used in the above training or validation set:

9x Covid-19 lungs vs. 9x normal lung vs. 9x Bacterial Lungs. (These images are attached in this post too.)

I only changed one line of the code in step 2, to make sure it loads test mages from a different path:

```
...
imagePathTest = "./Covid_M/all/test"
...
```

The we use the above trained model to predict:

```
predTest = model.predict(dataTest, batch_size=BS)
print(predTest)
predClasses = predTest.argmax(axis=-1)
print(predClasses)
...
```

Finally we can re-calculate the confusion matrix as in step 5:

```
testX = dataTest
testY = labelsTest
... ...
```

We got some real test results:

```
Evaluating real test samples ...
              precision    recall  f1-score   support

       covid       1.00      1.00      1.00         9
      normal       1.00      0.89      0.94         9
pneumonia_bac       0.90      1.00      0.95         9

    accuracy                           0.96        27
   macro avg       0.97      0.96      0.96        27
weighted avg       0.97      0.96      0.96        27

[[9 0 0]
 [0 8 1]
 [0 0 9]]
acc: 0.9630
sensitivity: 1.0000
specificity: 1.0000
```

Again, the trained model seems to be able to classify all Covid-19 lungs correctly. It's not that bad for such a small data set.

6. Some further observations:

I played with various dataset over the Easter weekend, and noticed that Covid-19 lungs seemed to have some distinct features somehow, from the AI classifier point of view - it's relatively easy to tell them apart from other normal Bacterial or Viral (Flu) Lungs.

Also I noticed with some quick tests that it's really difficult to distinguish between such as Bacterial and Viral (normal flu) lungs. I would have to go for ensemble cluster to chase down their difference if I have the time, just like other Kaggle competitors would possibly do in those circumstances.

Is the above really true from the clinical point of view? Does Covid-19 lungs really have some distinct features on X-Ray? I am not so sure. I would have to ask real chest radiologist for opinions. For now, I'd rather assume the current dataset is just too small to draw a conclusion yet.

Next: I would love to collect more real site X-Rays to give them a serious look with some xgsboot, AutoML or our new IRIS IntegratedML workbenches. And even more, we should be able to further classify the Covid-19 lungs into such as Level 1, Level 2 & Level 3 per its seriousness for clinicians and A&E triage doctors, I hope?

Anyway, I attached the dataset and the above Jupyter Notebook.

Deployment

We touched a bit simple starting point for some quick setups above in this "Medical Imaging" field. This Covid-19 front is actually the 3rd I tried to look into for the past year over some weekends and long holidays. Others include

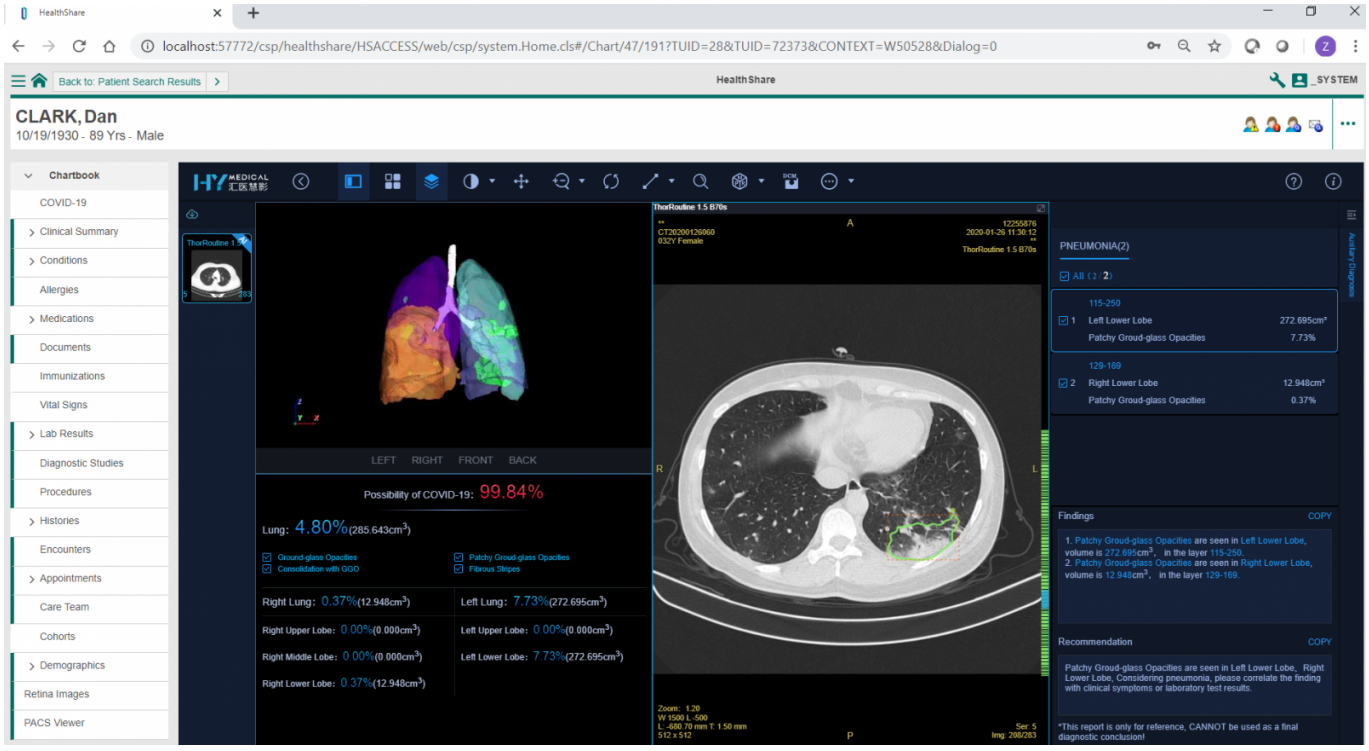
an "AI Assisted Bone Fracture detection" system, and an "AI assisted eye(ophthalmology) retina diagnosis" system.

The above model might be too simple to bother for now, but sooner than we might think when we can't avoid the common question: how are we going to deploy it into a kind of "AI service""?

It is about technology stacks and service life-cycle, but also about the actual "Use Case" - what problems are we trying to solve, and what real values can it provide? The answers sometimes are not as clear as technology itself.

UK RCR(Royal College of Radiologist) draft proposed 2 simple use cases: "Radiologist's AI assistant", and "AI Triage in A&E or primary care settings". To be honest, personally I agree and think the 2nd one "AI Triage" would provide more value for now. And fortunately today's developer is so much more powered than ever before by the Cloud, Docker, AI, and certainly our HealthShare to address this sort of cases.

For example, the screen capture below actually shows an enterprise-grade "AI assisted CT detection of Covid-19 lungs" service hosted in AWS, and how it could be embedded directly into a HealthShare Clinical Viewer for demo purpose. Similar to X-Rays, CT set in DICOM can be directly uploaded or sent into this open PACS Viewer, then with one click on "AI diagnosis", it can give a quantified dictation of Covid-19 probability in less than 10 seconds based on trained models, working 24x7 for the quick "AI triage" use case. The X-Ray classification etc models can be deployed and invoked in the same approach on top of existing PACS viewer within the same patient's context to help the front-line clinicians.



Acknowledgement

Again, test images are from Covid-19 Lung X-Ray set by Joseph Paul Cohen, and some clean lungs from the open Kaggle Chest X-Ray sets, collected by Adrian Yu at the GradientCrescent repository. I also re-use the structure of Adrian at PyImageSearch with my own improved training as listed under "Test" section. And thanks to HYM providing the AWS cloud based Open PACS Viewer with AI modules for X-Ray and CT images to look into test datasets.

What's Next

Today AI has "eroded" almost every aspect of human's health and daily life. In my oversimplified view, AI applications in healthcare might largely have the following few directions:

- **Medical Imaging**: including chest, heart, eye and brain etc X-Ray, CT, or MRI etc images.
- **NLP comprehension**: mining, understanding and learning of vast text asset and knowledge base.
- **Population Health**: including Epidemiology etc trend predictions, analysis and modeling.
- **Personalised AI**: a set of AI/ML/DL model(s) specially trained for and dedicated to an individual, growing up and growing old with him/her, as a personal health assistant?
- **Other AI(s):** such as AlphaGo or even AlphaFold for 3D protein structure predictions etc - fighting against Covid-19 - really impressed with those cutting-edge breakthroughs.

We will see what we can pick up along the journey. It might be just a wish list anyway, unless we stay home for far too long.

Appendix - File Uploads here. In includes image as we used above and a Jupyter Notebook file as above. It might take a couple of hours to set up and run from scratch during a weekend.

31 1 3 725 3

Related posts

- Run A Deep Learning Demo with Python3 Binding to HealthShare (Part II)
- Run some Covid-19 lung X-Ray classification and CT detection demos
- Explainability and Visibility into Covid-19 X-Ray Classifiers by Deep Learning

Show all

Source URL: https://community.intersystems.com/post/run-some-covid-19-lung-x-ray-classification-and-ct-detection-demos