Goodbye, Flat Convolutions.

MA5852 Master Class 2

Assessment 3: Project Plan

By Anthony Lighterness

18 June 2021

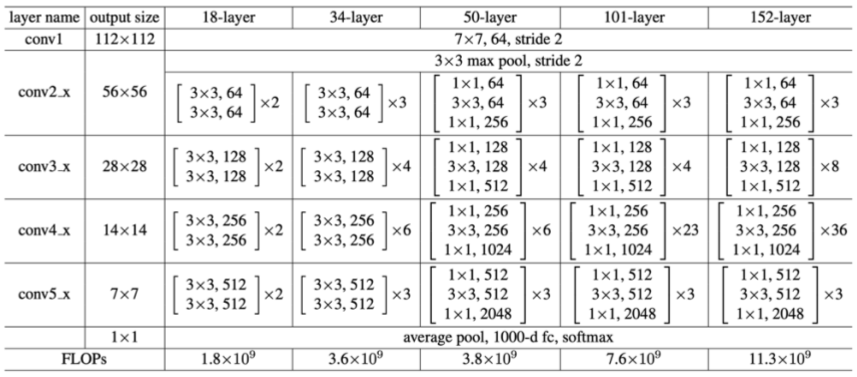
Table of Contents

Part 1: Revisiting ResNet50

With Big Data comes even bigger models. The global ImageNet competition challenges data scientists to classify 20,000 classes of objects from 14 million hand-labelled images (ImageNet, 2021). Since 2011, over 400 models have attempted to beat its predecessor (Image Classification on ImageNet, 2021). Convolutional neural networks (CNNs) are particularly prevalent in this competition as they can learn spatial features from imaging data. To learn more effective, data scientists found that CNNs need to go deeper. However, building deeper networks have revealed two counterintuitive problems: the degradation problem and vanishing gradient descent. As such, the Microsoft Research team tackled this by inventing residual networks, or ResNets.

The Architecture of ResNet50

The residual learning framework consists of a novel architecture of deep convolutional networks containing skip connections in combination with heavy batch normalisation. These skip connections are comparable to the features that make recurrent neural networks successful and allow networks to be designed with increasing number of layers whilst avoiding the risk of accuracy degradation. To demonstrate this, He et al. (2015) published five variants of the ResNet architecture, as seen in Figure 1 below.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 1: Architectures of ResNet variants published by He et al. (2015).

We chose the 50-layer ResNet architecture to study as its performance is significantly better than the 18- and 34-layer networks, but less complex than the 101- and 152-layer variants. Overall, ResNet50 takes an input shape of (224 x 224 x 3), which then outputs to the first convolutional layer. Then, four convolutional blocks follow before a fully connected output layer is adopted (He et al., 2015). The convolutional layers apply (3 x 3) filters with a stride of 2 (He et al., 2015). A batch normalisation layer is applied immediately after each convolutional layer, as well as immediately before the activation layers that follow (He et al., 2015). The network also adopts 'skip' connections to 'every few' convolutional layers (He et al., 2015). These are the key innovation of this work, so we describe them in more detail below.

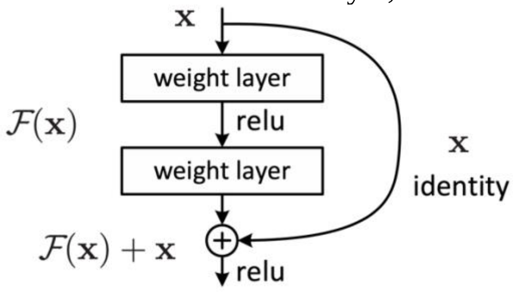
Since the authors set out to address the degradation problem, a key problem faced with networks containing bottleneck designs, the authors decided to include bottleneck shapes in the ResNet50 and higher variants. This means that the smaller ResNets contain only 2-layer convolutional blocks of (3 x 3) shapes, while ResNet50 and higher adopt three-layer convolutionals of (1 x 1), (3 x 3), (1 x 1), as seen in Figure 1 above. In sum, ResNet50 contains 50 convolutional layers, a final fully connected output layer, approximately 23 million trainable parameters, and a sum of 3.8 billion floating point operations (FLOPs) (He et al., 2015).

The parameters used to implement the ResNet50 model are now considered. Each convolutional layer adopts a stochastic gradient descent (SGD) optimizer with a mini-batch size of 256 and a rectified linear unit (ReLU) activation function (He et al., 2015). An initial learning rate of 0.1 was applied, which dropped by a division of 10 each time the error rate plateaud. The model trained over 60 x 10^4 epochs, with a weight decay of 1 x 10^{-4} and a momentum of 0.9 (He et al., 2015).

The Innovation of Residual Learning

The key innovation of the residual learning framework is the application of 'skip' connections. He et al. (2015) observed that when deep networks start to converge, accuracy degraded, and gradient descent diminished to an infinitely small value. Skip connections solved this.

Briefly, a plain, shallow neural network can be described to receive input x, and output the sum value, y. The output can as such be described as a function of x, or y = F(x). Stacking hidden layers in sequence causes the outputs of subsequent layers to accumulate the sum of the previous functions. For example, a second layer, G, receives the first output, F(x), making the second output to be y = G(F(x)). A third layer, M, then outputs y = M(G(F(x))), and so on. This means that the G and M layers are called identity functions (He et al., 2015). During optimisation, the gradient is back-propagated by multitudes of the number of previous layers. When many layers are stacked in a linear design, this causes the gradient to diminish rapidly, resulting in degradation of training accuracy. Therefore, skip connections, as shown in Figure 2 below, simply adds the input x to the output of two resulting subsequent layers, F(x), hence resulting in an output F(x) + x (He et al., 2015). As such, the two resulting layers have an uninterrupted gradient flow from the first to the last layer, thus avoiding a vanishing gradient.



x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2: A residual learning block with a skip connection, adapted from He et al. (2015).

Limitations of ResNet50 and Residual Learning Framework

Skip connections have solved a significant problem for very deep networks. However, ResNet architectures heavily rely on batch normalisation layers and bring with them new layers of complexities. In practical terms, adding many skip connections to hidden convolutional layers makes it more challenging to consider the changes in data dimensionality in deep layers. This is exacerbated as new networks can exploit this technique and reach significantly more deep and complex architectures. Finally, a general limitation of most contemporary pretrained models is that they confer weights useful only for 2D projects, i.e. image data with 3-colour channels. Medical image analysis, where z-stack MRI or CT data necessitate ease of computational demand, would require 3D pretrained models instead.

Key Results and Contributions

He et al. (2015) showed that skip connections allow CNNs of increasing numbers of layers, going from 18 to 152, achieved greater accuracy and error metrics whilst avoiding the vanishing gradient and degradation problem. This achievement allowed the authors to take first place in the 2015 ImageNet Large Scale Visual Recognition Challenge. As such, models since this publication have included skip connections, achieving more competitive validation error on the ImageNet dataset. As a result, the most competitive and recent CNNs in the ImageNet competition were able to adopt hundreds of millions of trainable parameters across hundreds of layers deep.

Part 2: Computer Vision in 3D

Convolutional neural networks (CNNs) have shown promising potential in the field of computer vision. In the medical domain, we see the potential for deep learning networks to assist radiologists during the diagnostic process. Most notably, the coronavirus global pandemic has motivated many researchers to apply what the last decade of ImageNet computer vision research has found. However, a recent study published in Nature Machine Intelligence found that of 2,212 studies that attempted to classify COVID-19 from chest radiographs and computed tomography (CT) scans using deep learning models, none were likely candidates for clinical translation (Roberts et al., 2021).

One of the key issues in medical image analysis is the heavy reliance on X-ray images or single CT slices. This is problematic due to two critical reasons: firstly, radiographs, while cheap and fast, are unreliable, inaccurate, and therefore serve merely to a supplement a medical diagnosis. This could be one cause of deep learning models to learn incorrect features when attempting to classify chest radiographs, as shown by DeGrave, Janizek, and Lee (2021). Secondly, while CT and MRI scans have higher potential for detail and therefore serve a greater role for diagnostics, most current deep learning applications either extract a single slice from the full z-stack or split the entire stack and treat each slice as an independent sample. While the former method deliberately discards the depth information offered by CT and MRI stacks, the latter introduces a sampling bias as individual slices are no longer truly independent samples. Clearly there is a need to address these issues.

Research Proposal: From 2D to 3D Models

The primary objective of the current report is to demonstrate the rationale for moving towards using 3-dimensional (3D) CNN architectures, especially when analysing multi-slice stacks of CT or MRI images. Recently, this was demonstrated by Zunair et al. (2020), who evaluated 2D versus 3D CNN's for classifying tuberculosis (TB) from CT scans. To this end, we set out to conduct two experiments in parallel: one using 2D, and another using 3D CNN networks. Each experiment evaluated three CNN architectures of increasing complexity. However, to make a fair comparison between the 2D and 3D models, we maintained the same architectures. This is made seamless using Keras utilities, which include changing Conv2D to Conv3D, and MaxPooling2D to MaxPooling3D (Keras, 2021). As such, we attempted to answer the following questions when classifying TB from CT scans:

1) Do 3D CNNs outperform 2D CNNs?
2) Does batch normalisation improve the performances for any 2D or 3D model?
3) Do additional convolutional and fully connected hidden layers improve performance?
4) Can a pretrained ResNet50 model outperform the in-house designed 2D or 3D models?

Data and Pre-Processing Methodology

The current project used Amazon Web Services (AWS) to leverage more powerful computational resources for CNN model training on z-stack CT volume data. Briefly, we stored our data in Amazon's Simple Storage Service (S3) bucket, and then used an Amazon SageMaker notebook instance to prepare the image dataset for training. Then, we trained our models on an Amazon Elastic Computing (EC2) instance called *ml.p2.xlarge*, which offers 61GB of RAM. Models were then deployed on an *ml.m4.xlarge* instance for inference and further evaluation. This workflow is visually displayed in the Appendix, in Figure 11 (part 1/3), Figure 12 (part 2/3), and Figure 13 (part 3/3) below. Now we describe in detail the pre-processing methodology before model building and training.

We chose the *MosMedData* image dataset, which is one of the largest and only available CT datasets that preserve the full z-stack slices, i.e. depth. Since it was published only recently few work is published analysing it using CNN deep learning models (Morozov et al., 2020; Gordaliza et al., 2019; Zunair et al., 2020). This dataset contains over 1,000 labelled chest CT images of (512 x 512 x 64) shape, which is available freely from https://mosmed.ai/en/ and requires the submission of an email address. The current project unfortunately could not utilise the already small 1,000 sample size, consiting of >12GB in size. While the S3 bucket could store this data, it was in fact the SageMaker EC2 instance which was computationally overwhelmed during the pre-processing of this dataset – a critical step to ready the data before model training. We showcase an example of such error in the Appendix below in Figure 14, where we attempt to crop only 200 image stacks of shape (512 x 512 x 64) down to only (224 x 224 x 64). As such, we utilised a subset of 200 samples, consisting of an even 50% split of normal and abnormal patients, and resized these images into (128 x 128 x 64).

Before model building, the CT dataset was pre-processed following the protocols of previous 2D and 3D CT and MRI work. As previously mentioned, the current project conducted a 2D and 3D experiment in parallel with one another. To maintain a fair comparison between 2D and 3D models, we maintained the same sample selection. However, for the 2D experiment, we followed what is standard protocol in this domain, which is extracting a single slice from the mid-section of each stack, thus preserving maximum volume of the lung cavity (Lin et al., 2018; Wacker et al., 2020). This was achieved seamlessly using the *pfdo-med2image* pypi package (version 1.1.6) (Pypi, 2021).

Next, to mimic the 3-channel image representation, i.e. RGB, we applied a colour space conversion function and then resized the width and height dimensions to (224 x 224) as per prior work and the ImageNet protocol (Zunair et al., 2020; Rohila et al., 2021). As mentioned previously, the 3D images were reshaped into (128 x 128 x 64) due to due computational demand of (224 x 224 x 64) dimensions, which again is evidenced below in the Appendix, in Figure 14. A quick note on the difference between 2D vs 3D work is important. Some authors have suggested that convolutions on 3D images are comparable to 2D ones, because they simply extend a 3-colour channel to multiple layers. However, we point out, as seen in Figure 3 below, that why this is not the case. The key difference is that 2D convolutions aggregates the sum of three 2D matrices, resulting in a 2D output, whereas a 3D convolutions result in 3D a 3D cube.

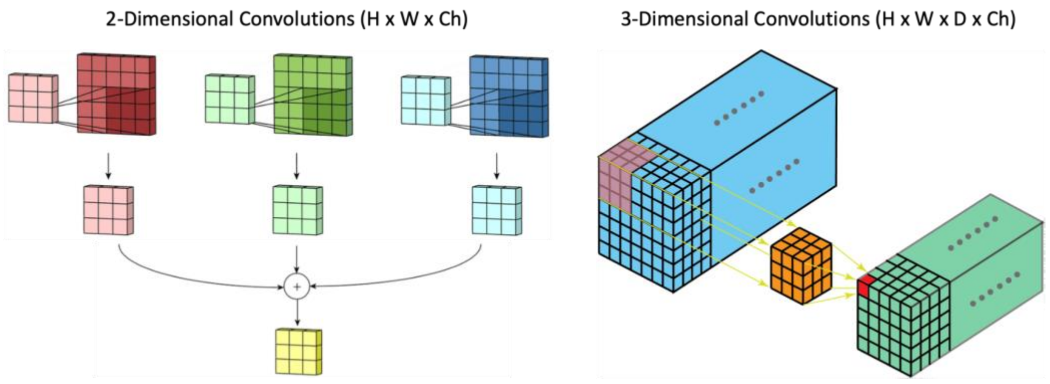

2-Dimensional Convolutions (H x W x Ch)
3-Dimensional Convolutions (H x W x D x Ch)

Figure 3: Convolutions in 2D versus 3D, adapted from Shafkat (2018), Bai (2019) and Ganesh (2019).

Both 2D and 3D z-stack image samples were normalised following the protocol used in previous work. This is based on the reasoning that CT signal intensities are absorbed differently based on differing body tissue densities (Rohila et al., 2021; Zunair et al., 2020; Adaloglou, 2021). These signal intensities are measured on the Hounsfield scale, which ranges from -1000 HU (air) to 400 HU (dense, cortical bone) (Adaloglou, 2021). Therefore, unlike RGB images, the grayscale CT images must follow this normalisation step in addition to the /255 normalisation.

Finally, since both the 2D and 3D image datasets were already split into normal and abnormal classes, we further split these two classes into training, validation, and hold-out test subsets. As such, for both experiments, we used 140 samples for training, 40 for validation, and 20 as a hold-out to test and evaluate the predictions made by our models. Each data subset contains an even 50% split of healthy or abnormal patient samples. The python script for this protocol is shown in the Appendix in Figure 15 below. Figure 4 below shows an example of a series of 40 z-stack CT images in sequence, while Figure 5 shows examples of single mid-section CT slices from healthy and abnormal patients.

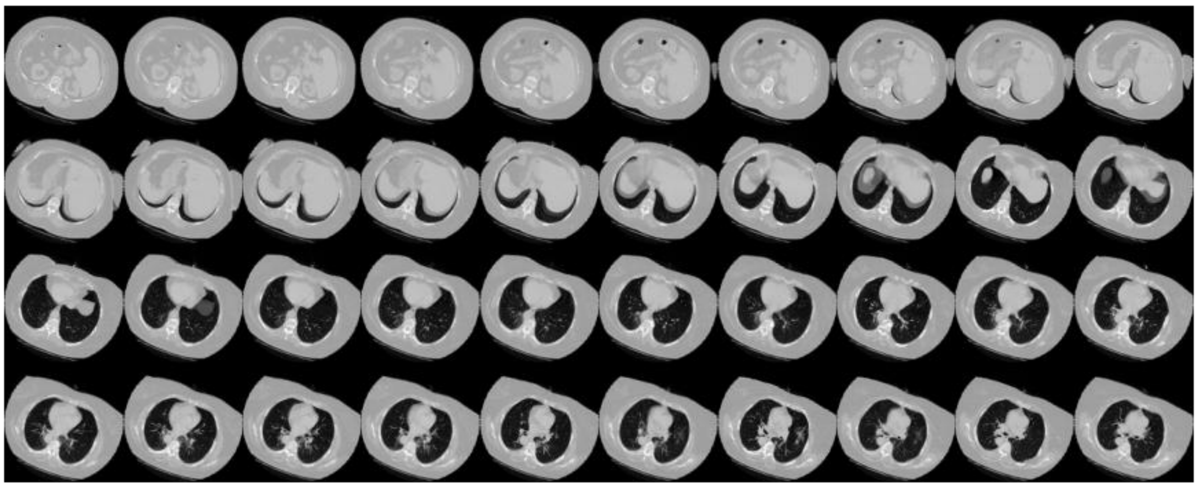


Figure 4: Example series of stacked CT scans for one patient.

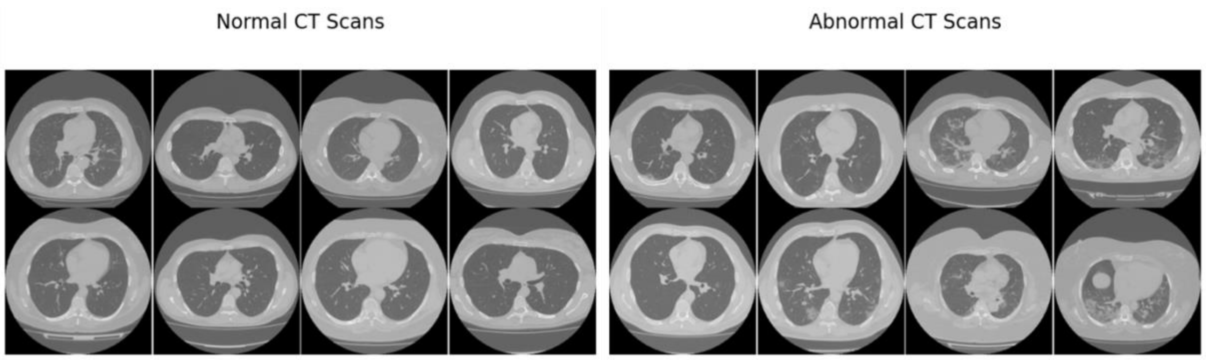


Figure 5: Example 2D CT samples from normal and abnormal patients.

Model Architectures, Parameters and Hyperparameters

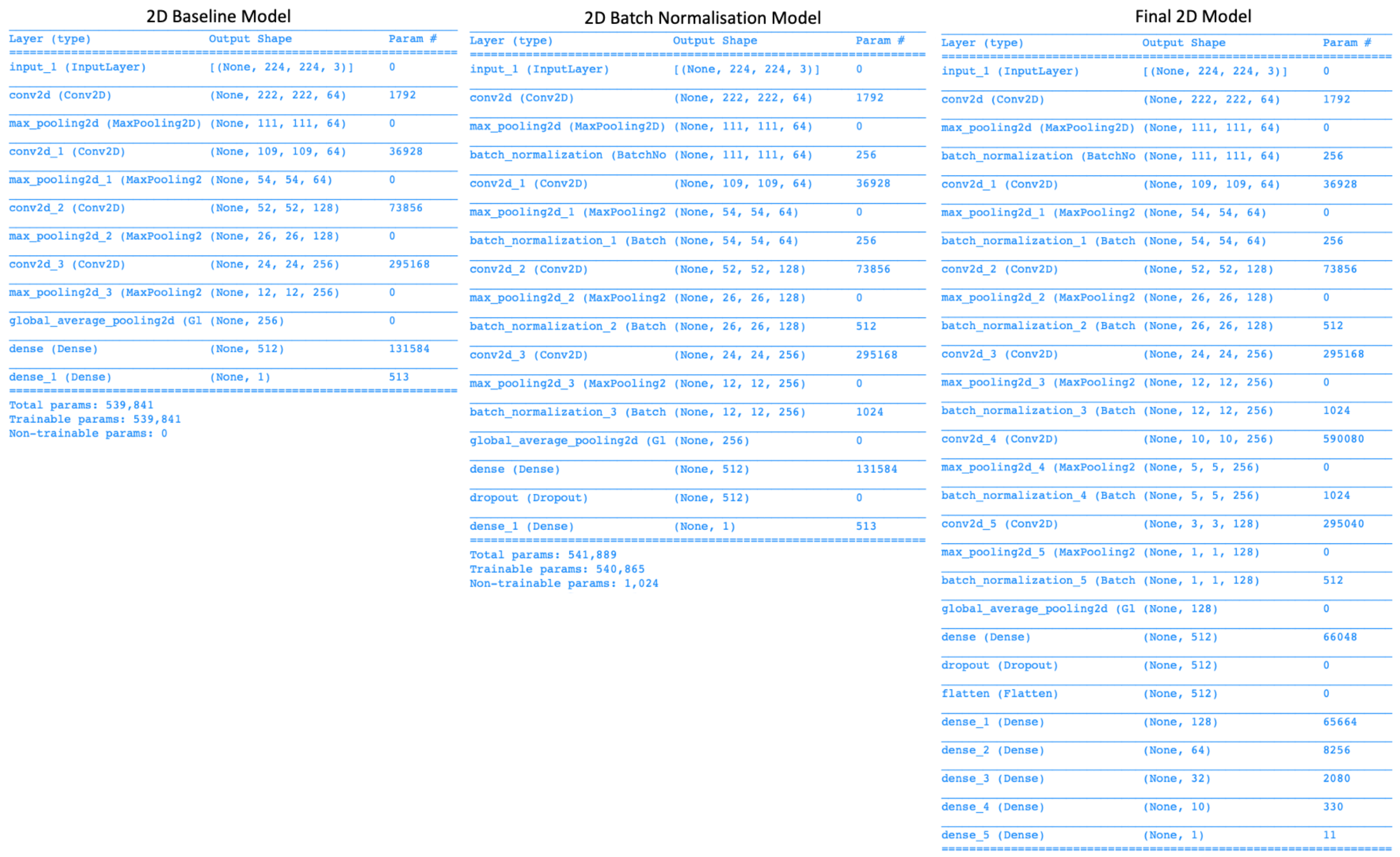
In order to study the impact moving from a 2D to 3D CNN design, we designed a 2D and 3D version of three models of increasing complexity. As such, we will describe the three models as follows, keeping in mind that a 2D as well as a 3D version of each was evaluated. The architectures of all six in-house networks evaluated are displayed Figure 6 and Figure 7 below. Firstly, a simple CNN model was designed – our baseline, which consisted of an input shape of (224 x 224 x 3) for the 2D version and (128 x 128 x 64 x 1) for the 3D version. Following this, both versions consisted of four convolutional layers immediately followed by max pooling layers. Then, we adopted a global average pooling layer immediately before a 512-node dense layer, finally outputting to a single node output layer. This architecture is commonly found in simple CNN models, where convolutional layers are immediately followed by either max or average pooling, batch normalisation, and/or global average pooling layers (Ganesh, 2019; Gordaliza et al., 2019; Lin et al., 2018; Raghu et al., 2019; Rohila et al., 2021).

Next, we designed our second model, which adopted a batch normalisation layer after each convolutional and max pooling layer. In other words, each of four convolutional blocks contained three layers: convolutional → max pooling → batch normalisation. In addition, the second model also adopted a dropout layer, set at 0.2, immediately following the 512-node dense layer. Our final model for evaluation adopts the same architecture as the second model, but adds two additional convolutional blocks, as well as a flatten layer, and four additional fully connected dense layers, which connects the 512-node layer found in model 2 with the final single node output layer. The dense layers in sequence contained 256, 128, 64, and 32 nodes, respectively. All model architectures are displayed for visual comprehension below in Figure 6 and Figure 7.

All parameters were preserved between the 2D and 3D equivalents. For example, all convolutional layers in all six models adopted a kernel size of 3, but each subsequent convolution varied in filter size, going from 64 to 128 and then 256. It should be noted that when applying a single integer for the kernel size, the Keras utilities assume for 2D convolutional layers a kernel dimension of (3 x 3) and for 3D layers (3 x 3 x 3) (Keras, 2021). All convolutional layers adopted a ReLU activation function, apart from the single node output layer which used sigmoid. Likewise, all max pooling layers adopted a pool size of 2. As such, while the output layer for all networks adopted a binary cross entropy loss and sigmoid activation function, the convolutional and hidden layers adopted a ReLU activation function and an ADAM optimiser. The ReLU activation function and ADAM optimiser are currently among the most popular choices for CNN architectures. However, we included a stochastic gradient descent (SGD) optimiser in a hyperparameter tuning job to measure a possible improvement in training convergence.

The hyperparameters chosen were informed by prior literature as well as a hyperparameter tuning job using Bayesian optimisation (Turner et al., 2021). As seen in the Appendix in Figure 1 and Figure 17, the hyperparameter tuning job explored two activation functions, including ReLU as well as SGD, a range of epochs from 1 to 50, a continuous range of epochs from 0.00001 to 0.1, batch sizes of 1, 2, 4, or 8, a continuous range of dense layers from 16 to 530, L1 regularisation ranging from 0.005 to 1, and L2 (weight decay) ranging also from 0.005 to 1. The tuning job was adopted in attempt to optimise our baseline model – a simple CNN described above.

While the ReLU activation function yielded a slight improvement compared to SGD, the overall hyperparameter tuning job revealed poor validation accuracies for all combinations, ranging from 40% to 50%, as shown in the Appendix in Figure 16 and Figure 17. As such, it was decided to train all models over 50 epochs, at a batch size of 1, an initial learning rate of 0.0001, decay rate of 0.96 and 10,000 decay steps. Zunair et al. (2020) yielded a validation AUC ranging from 53% to 67% using these parameters for 3D CNNs. We also consulted Smith's (2018) technical review of hyperparameter tuning to validate the final choice of values. Logically, we adopted a small learning rate as our dataset is very small and may allow models to overfit, therefore potentiating an improvement with the batch normalisation network.



2D Baseline Model

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0

conv2d (Conv2D)              (None, 222, 222, 64)      1792

max_pooling2d (MaxPooling2D) (None, 111, 111, 64)      0

conv2d_1 (Conv2D)            (None, 109, 109, 64)      36928

max_pooling2d_1 (MaxPooling2 (None, 54, 54, 64)        0

conv2d_2 (Conv2D)            (None, 52, 52, 128)       73856

max_pooling2d_2 (MaxPooling2 (None, 26, 26, 128)       0

conv2d_3 (Conv2D)            (None, 24, 24, 256)       295168

max_pooling2d_3 (MaxPooling2 (None, 12, 12, 256)       0

global_average_pooling2d (Gl (None, 256)               0

dense (Dense)                (None, 512)               131584

dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 539,841
Trainable params: 539,841
Non-trainable params: 0
```

2D Batch Normalisation Model

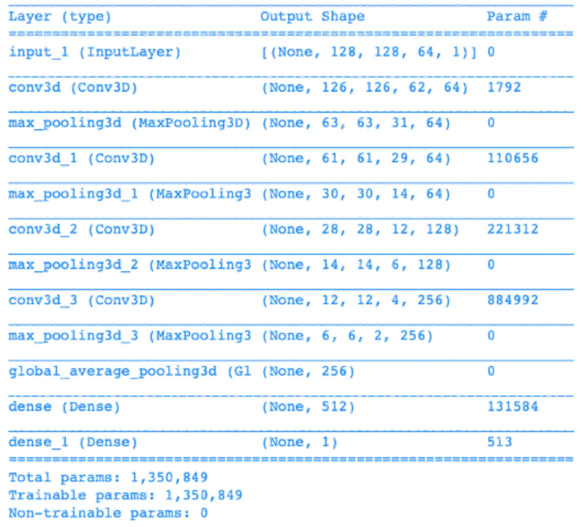
```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0

conv2d (Conv2D)              (None, 222, 222, 64)      1792

max_pooling2d (MaxPooling2D) (None, 111, 111, 64)      0

batch_normalization (BatchNo (None, 111, 111, 64)      256

conv2d_1 (Conv2D)            (None, 109, 109, 64)      36928

max_pooling2d_1 (MaxPooling2 (None, 54, 54, 64)        0

batch_normalization_1 (Batch (None, 54, 54, 64)        256

conv2d_2 (Conv2D)            (None, 52, 52, 128)       73856

max_pooling2d_2 (MaxPooling2 (None, 26, 26, 128)       0

batch_normalization_2 (Batch (None, 26, 26, 128)       512

conv2d_3 (Conv2D)            (None, 24, 24, 256)       295168

max_pooling2d_3 (MaxPooling2 (None, 12, 12, 256)       0

batch_normalization_3 (Batch (None, 12, 12, 256)       1024

global_average_pooling2d (Gl (None, 256)               0

dense (Dense)                (None, 512)               131584

dropout (Dropout)            (None, 512)               0

dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 541,889
Trainable params: 540,865
Non-trainable params: 1,024
```

Final 2D Model

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0

conv2d (Conv2D)              (None, 222, 222, 64)      1792

max_pooling2d (MaxPooling2D) (None, 111, 111, 64)      0

batch_normalization (BatchNo (None, 111, 111, 64)      256

conv2d_1 (Conv2D)            (None, 109, 109, 64)      36928

max_pooling2d_1 (MaxPooling2 (None, 54, 54, 64)        0

batch_normalization_1 (Batch (None, 54, 54, 64)        256

conv2d_2 (Conv2D)            (None, 52, 52, 128)       73856

max_pooling2d_2 (MaxPooling2 (None, 26, 26, 128)       0

batch_normalization_2 (Batch (None, 26, 26, 128)       512

conv2d_3 (Conv2D)            (None, 24, 24, 256)       295168

max_pooling2d_3 (MaxPooling2 (None, 12, 12, 256)       0

batch_normalization_3 (Batch (None, 12, 12, 256)       1024

conv2d_4 (Conv2D)            (None, 10, 10, 256)       590080

max_pooling2d_4 (MaxPooling2 (None, 5, 5, 256)         0

batch_normalization_4 (Batch (None, 5, 5, 256)         1024

conv2d_5 (Conv2D)            (None, 3, 3, 128)         295040

max_pooling2d_5 (MaxPooling2 (None, 1, 1, 128)         0

batch_normalization_5 (Batch (None, 1, 1, 128)         512

global_average_pooling2d (Gl (None, 128)               0

dense (Dense)                (None, 512)               66048

dropout (Dropout)            (None, 512)               0

flatten (Flatten)            (None, 512)               0

dense_1 (Dense)              (None, 128)               65664

dense_2 (Dense)              (None, 64)                8256

dense_3 (Dense)              (None, 32)                2080

dense_4 (Dense)              (None, 10)                330

dense_5 (Dense)              (None, 1)                 11
=================================================================
```

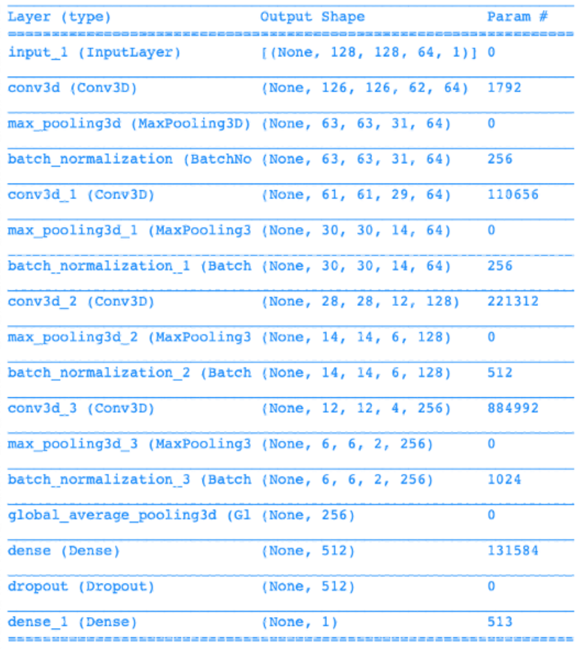
Figure 6: Three architectural designs for three 2D CNN's.

3D Baseline Model



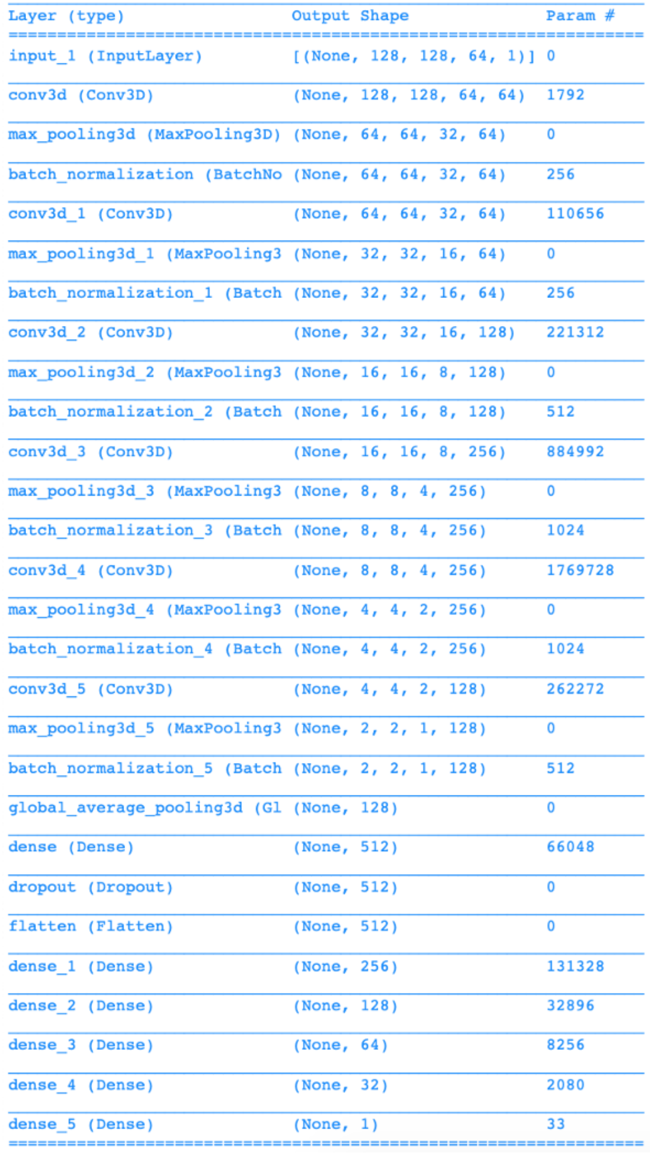
```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 128, 128, 64, 1)] 0
_________________________________________________________________
conv3d (Conv3D)              (None, 126, 126, 62, 64)  1792
_________________________________________________________________
max_pooling3d (MaxPooling3D) (None, 63, 63, 31, 64)    0
_________________________________________________________________
conv3d_1 (Conv3D)            (None, 61, 61, 29, 64)    110656
_________________________________________________________________
max_pooling3d_1 (MaxPooling3 (None, 30, 30, 14, 64)    0
_________________________________________________________________
conv3d_2 (Conv3D)            (None, 28, 28, 12, 128)   221312
_________________________________________________________________
max_pooling3d_2 (MaxPooling3 (None, 14, 14, 6, 128)    0
_________________________________________________________________
conv3d_3 (Conv3D)            (None, 12, 12, 4, 256)    884992
_________________________________________________________________
max_pooling3d_3 (MaxPooling3 (None, 6, 6, 2, 256)      0
_________________________________________________________________
global_average_pooling3d (Gl (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 512)               131584
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 1,350,849
Trainable params: 1,350,849
Non-trainable params: 0
```

3D Batch Normalisation Model



```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 128, 128, 64, 1)] 0
_________________________________________________________________
conv3d (Conv3D)              (None, 126, 126, 62, 64)  1792
_________________________________________________________________
max_pooling3d (MaxPooling3D) (None, 63, 63, 31, 64)    0
_________________________________________________________________
batch_normalization (BatchNo (None, 63, 63, 31, 64)    256
_________________________________________________________________
conv3d_1 (Conv3D)            (None, 61, 61, 29, 64)    110656
_________________________________________________________________
max_pooling3d_1 (MaxPooling3 (None, 30, 30, 14, 64)    0
_________________________________________________________________
batch_normalization_1 (Batch (None, 30, 30, 14, 64)    256
_________________________________________________________________
conv3d_2 (Conv3D)            (None, 28, 28, 12, 128)   221312
_________________________________________________________________
max_pooling3d_2 (MaxPooling3 (None, 14, 14, 6, 128)    0
_________________________________________________________________
batch_normalization_2 (Batch (None, 14, 14, 6, 128)    512
_________________________________________________________________
conv3d_3 (Conv3D)            (None, 12, 12, 4, 256)    884992
_________________________________________________________________
max_pooling3d_3 (MaxPooling3 (None, 6, 6, 2, 256)      0
_________________________________________________________________
batch_normalization_3 (Batch (None, 6, 6, 2, 256)      1024
_________________________________________________________________
global_average_pooling3d (Gl (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 512)               131584
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
```

Final 3D Model



```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 128, 128, 64, 1)] 0
_________________________________________________________________
conv3d (Conv3D)              (None, 128, 128, 64, 64)  1792
_________________________________________________________________
max_pooling3d (MaxPooling3D) (None, 64, 64, 32, 64)    0
_________________________________________________________________
batch_normalization (BatchNo (None, 64, 64, 32, 64)    256
_________________________________________________________________
conv3d_1 (Conv3D)            (None, 64, 64, 32, 64)    110656
_________________________________________________________________
max_pooling3d_1 (MaxPooling3 (None, 32, 32, 16, 64)    0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 16, 64)    256
_________________________________________________________________
conv3d_2 (Conv3D)            (None, 32, 32, 16, 128)   221312
_________________________________________________________________
max_pooling3d_2 (MaxPooling3 (None, 16, 16, 8, 128)    0
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 8, 128)    512
_________________________________________________________________
conv3d_3 (Conv3D)            (None, 16, 16, 8, 256)    884992
_________________________________________________________________
max_pooling3d_3 (MaxPooling3 (None, 8, 8, 4, 256)      0
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 4, 256)      1024
_________________________________________________________________
conv3d_4 (Conv3D)            (None, 8, 8, 4, 256)      1769728
_________________________________________________________________
max_pooling3d_4 (MaxPooling3 (None, 4, 4, 2, 256)      0
_________________________________________________________________
batch_normalization_4 (Batch (None, 4, 4, 2, 256)      1024
_________________________________________________________________
conv3d_5 (Conv3D)            (None, 4, 4, 2, 128)      262272
_________________________________________________________________
max_pooling3d_5 (MaxPooling3 (None, 2, 2, 1, 128)      0
_________________________________________________________________
batch_normalization_5 (Batch (None, 2, 2, 1, 128)      512
_________________________________________________________________
global_average_pooling3d (Gl (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 512)               66048
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
```

Figure 7: 3D equivalent architectures of the proposed 2D CNN designs.

Model Evaluation and Performance Results

The current study sought to investigate the performance of 2D versus 3D CNN models when classifying TB from chest CT images. Furthermore, we attempted to understand if classification performance is impacted by the addition of batch normalisation layers and additional convolutional layers feeding further into four hidden dense layers, i.e. 'going deeper'. These models were trained and deployed to endpoints on AWS for inference, and evidence of this is provided in the Appendix in Figure 19 and Figure 20.

The metrics chosen to evaluate model performances include loss, accuracy, F1 score, precision, recall, and AUC, which are summarised for all models considered, as well as a pretrained 2D ResNet50 model, as seen in Table 1 below. We further tested all models' performances when predicting the 20-sample hold-out test dataset. These predictive performances were evaluated using confusion matrices, which are presented in Figure 8 below.

The best score for each respective metric is bolded in Table 1 below for better visibility. Overall, we observed optimal performance metrics in our baseline 3D model, which achieved the best scores for all metrics except in four: training recall, and validation loss, recall and AUC. Yet, its validation AUC was a mere 0.88% behind the top position. These results transfer congruently to the models' predictive capabilities, which show again that our 3D baseline model correctly predicted 9 out of 10 healthy samples and 5 out of 10 abnormal samples. In contrast, all other models developed a tendency to predict either healthy or abnormal labels, suggesting inadequate learning. The validation accuracies achieved for all models ranged from 40% to 65%, which is a range found in previous work that also studied z-stack medical images (Bolhassani, 2021; Lin et al., 2018; Zhou et al., 2019; Yang et al., 2019; Zunair et al., 2020; Raghu et al., 2019).

Performance Metrics		2D				3D		
		Baseline	Batch Norm	Final	ResNet50	Baseline	Batch Norm	Final
Training	Loss	0.6802	0.6935	0.6932	1.5007	**0.0014**	0.6932	0.6935
	Accuracy	0.5429	0.4643	0.4714	0.6	**1**	0.4929	0.5
	F1 Score	0.5037	0.495	0.4727	0.5101	**0.7306**	0.5465	0.5962
	Precision	0.512	0.5096	0.5274	0.5322	**0.8016**	0.4742	0.4893
	Recall	0.5205	0.5251	0.5965	0.4898	0.6712	0.6448	**0.7628**
	AUC	0.6092	0.4857	0.5	0.6417	**1**	0.4932	0.4681
Validation	Loss	0.6952	0.6862	**0.6931**	2.6571	1.8937	0.7218	0.6944
	Accuracy	0.4	0.5	0.5	0.525	**0.65**	0.4	0.525
	F1 Score	0.5035	0.4946	0.473	0.5099	**0.7327**	0.5459	0.4892
	Precision	0.512	0.5078	0.528	0.5328	**0.8034**	0.4741	0.594
	Recall	0.5209	0.5218	0.5973	0.4888	0.6735	0.6434	**0.7558**
	AUC	0.4075	**0.6488**	0.5	0.4637	0.64	0.355	0.4688
Run Time (sec)		236	291	279	**28**	1727	1812	1908
Total # Params		539,841	541,889	1,438,837	23,688,065	1,350,849	1,352,897	3,494,977
# Trainable Params		539,841	540,865	1,437,045	100,353	1,350,849	1,351,873	3,493,185
# Nontrain Params		0	1,024	1,792	23,587,712	0	1,024	1,792
# Conv Layers		4	4	6	50	4	4	6

Table 1: Performance metrics for all models investigated, where best results are in bold for each respective metric.

The results of the models' predictive capabilities concur with those observed in the training and validation metrics. Interestingly, our 2D models' predictive performances deteriorated with the addition of batch normalisation and additional convolutional layers. As seen below, in Figure 8, adding batch normalisation and 'going deeper' caused a higher rate of misclassification of abnormal predictions in the 2D models. As such, all 2D models were inadequate predictors of abnormal CT slices, while the 3D baseline and batch normalisation models correctly predicted 5 and 6 out of 10 TB samples, respectively. This concurs with our 3D baseline model's validation F1-score and AUC of 73% 64%, respectively.

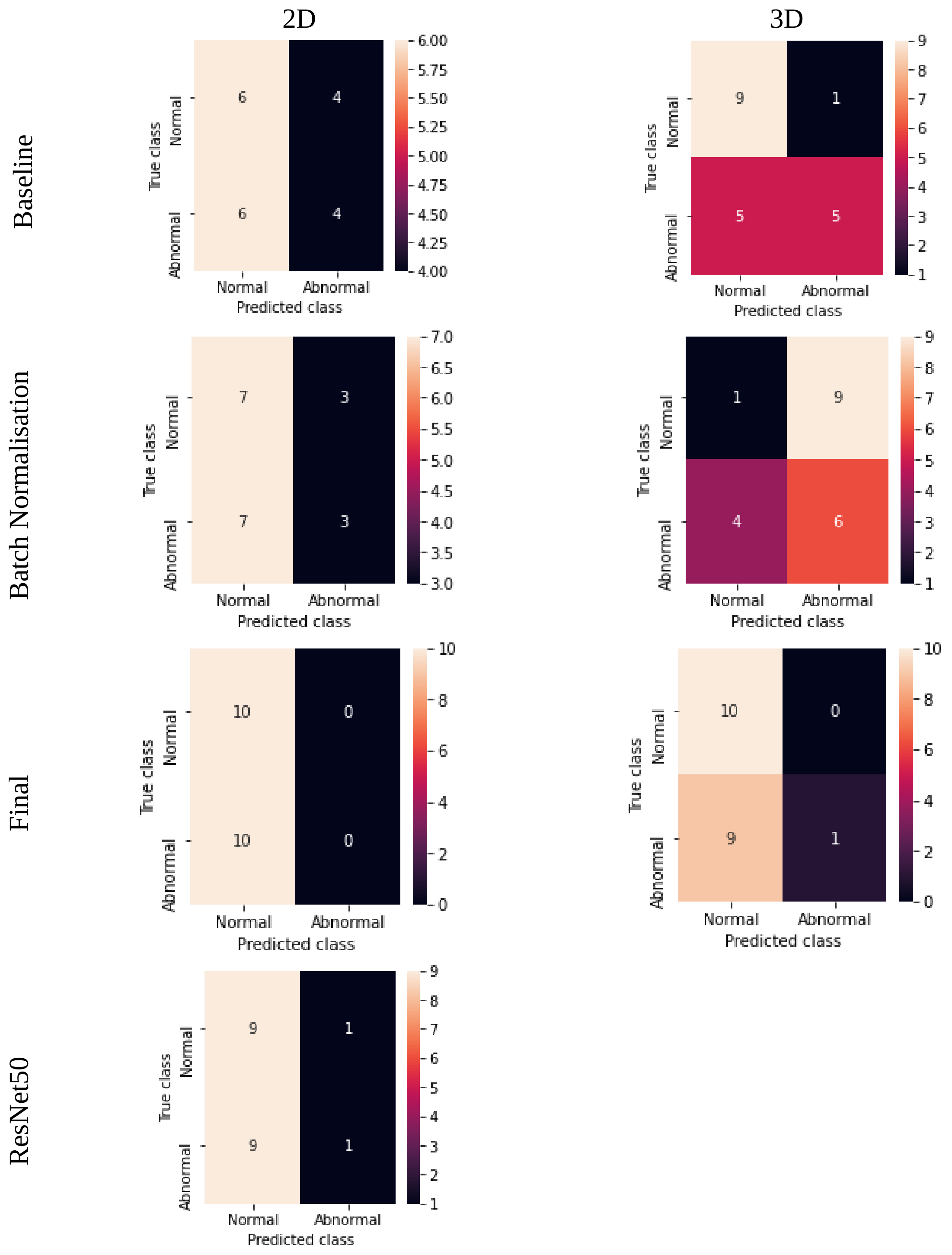


Figure 8: Confusion matrices for all models investigated.

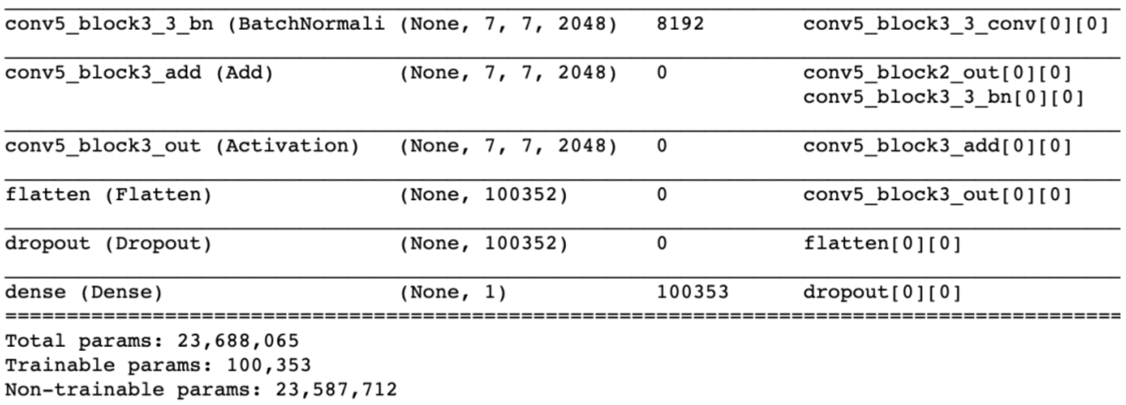
The addition of convolutional and dense layers to both 2D and 3D models, i.e. as seen in our 'final' 2D and 3D models, caused both versions to misclassify nearly all abnormal CT samples. This is likely indicative of overfitting behaviour. A similar result is observed with the ResNet50 model, which given the small sample size is unsurprising. Interestingly, the addition of batch normalisation layers to the 3D baseline model caused the model to classify 9 normal CT scans as abnormal. This is the only model to achieve the highest correct classification score for abnormal samples, being 6 out of 10. Yet its propensity to misclassify 9 normal samples indicates overfitting to abnormal features. We show more detailed evaluation metrics for all models' predictive capabilities below in the Appendix in Figure 18.

Comparison with ResNet50 for Transfer Learning

The current project, in addition to evaluating 2D to 3D deep learning, evaluated the training and predictive performance of a ResNet50 model pretrained using the ImageNet dataset. This describes the process of transfer learning, which primarily serves to reduce the training computation required to leverage a powerful and optimised model. Since ResNet50 was designed to classify 20,000 classes, we modified its output design by appending a flatten, dropout and single-node output layer. This can be seen in Figure 9 below.

Ballew (2017) describes four scenarios in which transfer learning may occur, with respective approaches to handle each. Briefly, the first is where the input dataset is large and highly comparable to the data used for the pretrained model. The second, slightly more challenging scenario, is with a small dataset, but still comparable new input images or subjects. Thirdly, is where the input data is of large volume, but from a different domain, e.g. medical images. Lastly and most challenging, is having a small input dataset from a different domain, which describes the current scenario. As such, it was not surprising that the renowned ResNet50 model achieved nearly the poorest performance metrics among all models considered. Interestingly, we observed highly volatile, fluctuating scores from the training history, as seen in Figure 10 below. The training history shows highly volatile, unstable, and fluctuating performance metrics over 50 training epochs, which is in fact similar to the findings reported by Bolhassani (2021) when attempting to apply a 2D pretrained model for medical imaging analysis.

Collectively, these results serve as a strong rationale for the need to continue this work and to develop 3D models that could be used for transfer learning. Previous work did attempt to convert 2D pretrained models to 3D structures, which yielded the production of a package called ACSConv (Yang et al., 2019). However, we regrettably report an unsuccessful attempt at implementing this technique due to technical, unsolvable conflicts in the different versions of the package dependencies.



```
________________________________________________________________________________________
conv5_block3_3_bn (BatchNormali (None, 7, 7, 2048)    8192        conv5_block3_3_conv[0][0]
________________________________________________________________________________________
conv5_block3_add (Add)          (None, 7, 7, 2048)    0           conv5_block2_out[0][0]
                                                                  conv5_block3_3_bn[0][0]
________________________________________________________________________________________
conv5_block3_out (Activation)   (None, 7, 7, 2048)    0           conv5_block3_add[0][0]
________________________________________________________________________________________
flatten (Flatten)               (None, 100352)        0           conv5_block3_out[0][0]
________________________________________________________________________________________
dropout (Dropout)               (None, 100352)        0           flatten[0][0]
________________________________________________________________________________________
dense (Dense)                   (None, 1)             100353      dropout[0][0]
========================================================================================
Total params: 23,688,065
Trainable params: 100,353
Non-trainable params: 23,587,712
```

Figure 9: Final layers of ResNet50, modified to fit the current application.

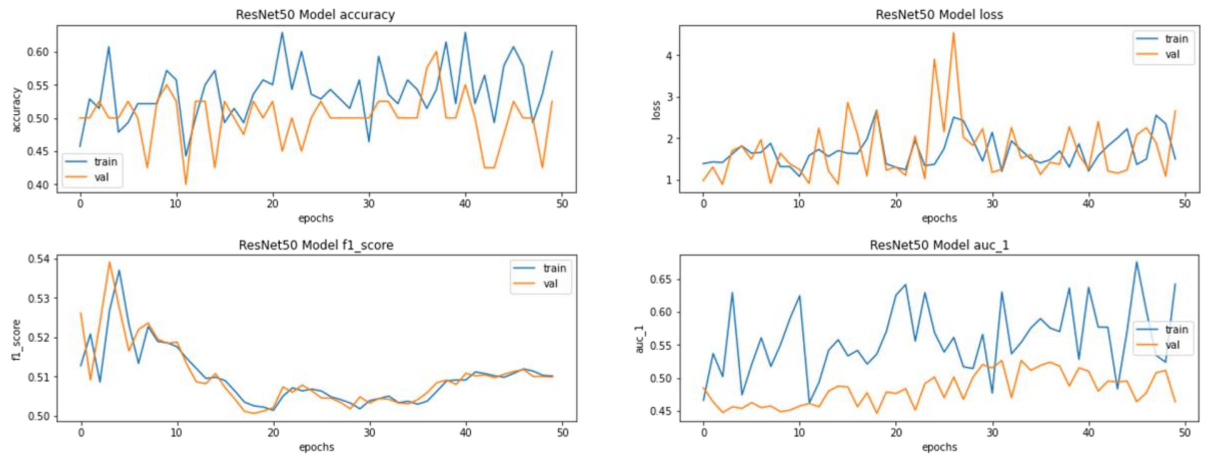


Figure 10: Performance history for a pre-trained ResNet50 model training only the final fully connecting layers.

Limitations

The current study sought to evaluate the impact that moving from 2D to 3D convolutional architectures, batch normalisation, and deeper networks may have on predicting TB from CT images. While these models were developed and evaluated using scientific methodologies and prior work, the current author wishes to propose recommendations for future work. Firstly, the most impactful limitation of the current study is the small sample size of 200 images. This is particularly small in the field of computer vision, where competitive models train on millions of image samples over tens of thousands of epochs. This is difficult to address due to the limited availability of reliably labelled CT and/or MRI image datasets that preserve the full z-stack slices. Moreover, the computational resources required for image processing exceed the standard, free EC2 instance on which SageMaker is run.

Secondly, the current project did not explore the impact of different image augmentation techniques, which is another widely impactful technique studied in the field of computer vision. To address this, future work should attempt to study the impacts of adding noise, different normalisation techniques, and cropping or resizing of images on model performance. Thirdly, the current authors regret an unsuccessful implementation a 3D ResNet50 model, which may have yielded optimal performance metrics overall. Lastly, the hyperparameter tuning job was limited by a range of 1 to 50 epochs. We found optimal performance at 50 epochs, even compared to 100. Yet, other work train models over multiple thousands of epochs, suggesting the need to further explore this.

Conclusion

In conclusion, the current investigation applied AWS to build and deploy CNNs of varying architectures for the binary classification of TB found in chest CT images. Currently, most computer vision tasks focus on the use of 2D CNN architectures that analyse single 2D images. However, we highlight the limitations of these protocols, particularly in the medical domain. Furthermore, we report that a 3D models outperform their 2D counterparts, as well as a ResNet50 model pretrained on the large ImageNet database. While we were strictly limited by a small sample size, our 3D baseline model still correctly classified 9 out of 10 healthy samples and 5 out of 10 TB cases, whereas the addition of batch normalisation improved this to 6 out of 10 TB samples. Meanwhile, all 2D models were inadequate predictors of TB samples. As such, this offers a strong rational to abandon old 2D protocols, which extract single 2D slices from z-stack CT and MRI images, thereby deliberately discarding important depth information.

References

Adaloglou, N. (2021). Introduction to Medical Image Processing with Python: CT Lung and Vessel Segmentation without Labels. Retrieved from the AI Summer website from https://theaisummer.com/medical-image-python/

Bai, K. (2019). A Comprehensive Introduction to Different Types of Convolutions in Deep Learning. Retrieved from *Towards Data Science* from https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Ballew, G. (2017). Using Transfer Learning and Bottlenecking to Capitalize on State of the Art DNNs. Retrieved from the Medium website from https://medium.com/@galen.ballew/transferlearning-b65772083b47

DeGrave, A.J., Janizek, J.D., and Lee, S.I. (2021). AI for Radiographic COVID-19 Detection Selects Shortcuts Over Signal. *Nature Machine Intelligence*. Doi: 10.1038/s42256-021-00338-7.

Ganesh, P. (2019). Types of Convolution Kernels: Simplified. Retrieved from *Towards Data Science* from https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37

Gordaliza, P.M., Vaquero, J.J., Sharpe, S., Gleeson, F., Munoz-Barrutia, A. (2019). A Multi-Task Self-Normalizing 3D-CNN to Infer Tuberculosis Radiological Manifestations. *arXiv*: 1908.12331.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv*: 1512.03385v1.

Image Classification on ImageNet. (2021). Retrieved from the Papers with Code website from https://paperswithcode.com/sota/image-classification-on-imagenet
ImageNet. (2021). ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Retrieved from https://www.image-net.org/challenges/LSVRC/

Keras. (2021). Conv3D layer. Retrieved from https://keras.io/api/layers/convolution_layers/convolution3d/

Lin, W., Tong, T., Gao, Q., Guo, D., Du, X., Yang, Y., Guo, G., Xiao, M., Du, M., and Qu, X. (2018). Convolutional Neural Networks-Based MRI Image Analysis for the Alzheimer's Disease Prediction from Mild Cognitive Impairment. *Frontiers in Neuroscience*, **12** (777), DOI: 10.3389/fnins.2018.0077.

Morozov, S.P., Andreychenko, A.E., Pavlov, N.A., Vladzymyrskyy, A.V., Ledikhova, N.V., Gombolevskiy, V.A., Blokhin, I.A., Gelezhe, P.B., Gonchar, A.V., and Chernina, V.Yu. (2020). *medRxiv*. Doi: 10.1101/2020/05/20/20100362. Available at www.mosmed.ai.

Pypi. (2021). Pfdo-Med2Image 1.1.6 package. Retrieved from https://pypi.org/project/pfdo-med2image/

Raghu, M., Zhang, C., Kleinberg, J., and Bengio, S. (2019). Transfusion: Understanding Transfer Learning for Medical Imaging. *arXiv*: 1902.07208v3.

Roberts, M., Driggs, D., Thorpe, M., Gilbey, J., Yeung, M., Ursprung, S., Aviles-Rivero, A.I., Etmann, C., McCague, C., Beer, L., Weir-McCall, J.R., Teng, Z., Gkrania-Klotsas, E., AIX-COVNET, Rudd, J.H.F., Sala, E., and Schnonlieb, C.B. (2021). Common Pitfalls and Recommendations for Using Machine Learning to Detect and Prognosticate for COVID-19 Using Chest Radiographs and CT Scans. *Nature Machine Intelligence*. Doi: 10.1038/s42256-021-00307-0.

Rohila, V.S., Gupta, N., Kaul, A., and Sharma, D.K. (2021). Deep Learning Assisted COVID-19 Detection Using Full CT-Scans. *Internet of Things*, **14**. Doi: 10.1016/j.iot.2021.100377.

Shafkat, I. (2018). Intuitively Understanding Convolutions for Deep Learning. Retrieved form *Towards Data Science* from https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Smith, L.N. (2018). A Disciplined Approach to Neural Network Hyper-Parameters: Part 1 – Learning Rate, Batch Size, Momentum, and Weight Decay. *arXiv*: 1803.09820.

Turner, R., Eriksson, D., McCourt, M., Kiili, J., Laaksonen, E., Xu, Z., and Guyon, I. (2021). Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020. *arXiv*: 2104.10201v1.
Wacker, J., Ladeira, M., and Nascimento, J.E.V. (2020). Transfer Learning for Brain Tumor Segmentation. *arXiv*: 1912.12452v2.

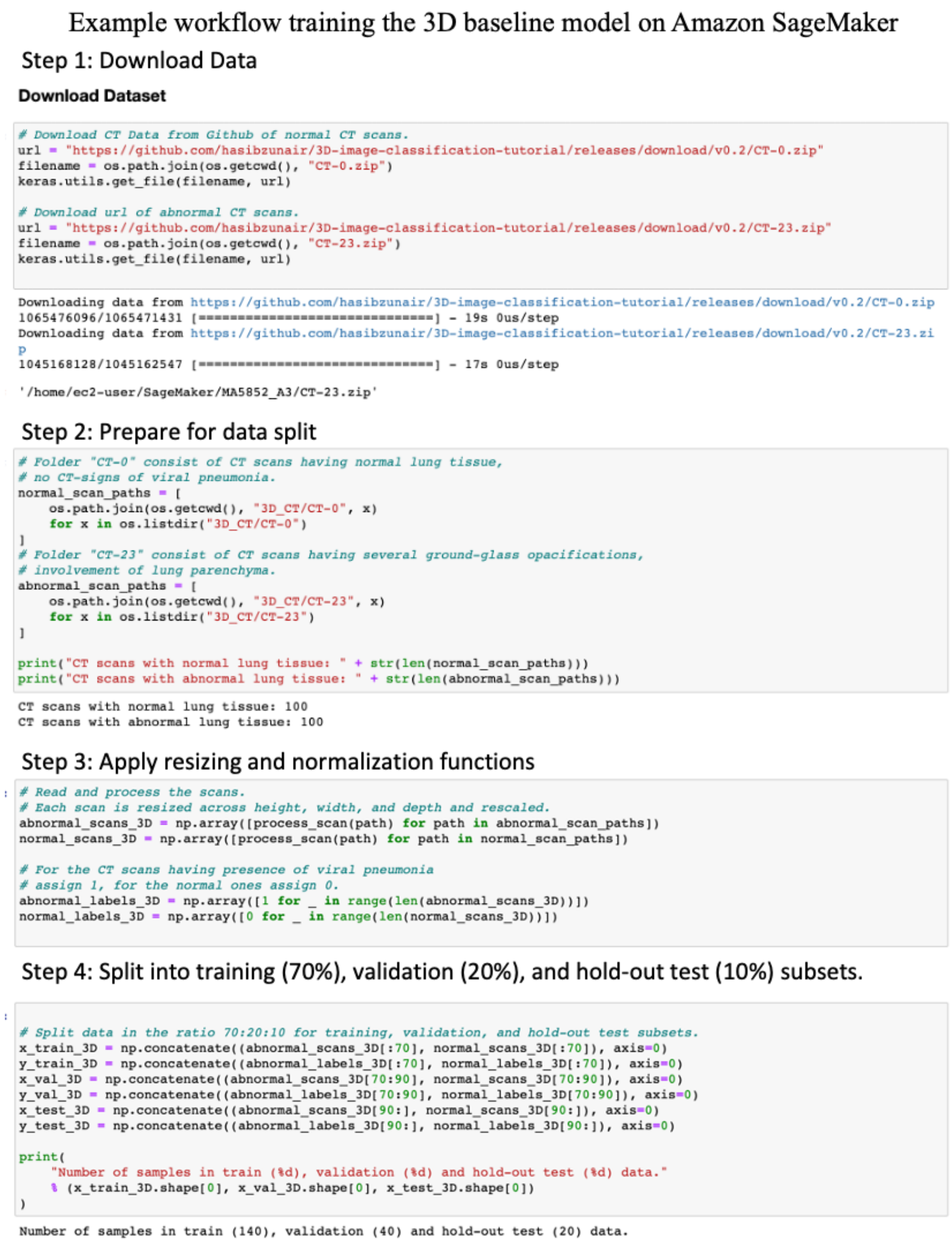
Wu, W., Li, X., Du, P., Lang, G., Xu, M., Xu, K., Li, L. (2019). A Deep Learning System that Generates Quantitative CT Reports for Diagnosing Pulmonary Tuberculosis. *arXiv*: 1910.02285.

Yang, J., Huang, X., Ni, B., Xu, J., Yang, C., and Xu, G. (2019). Reinventing 2D Convolutions for 3D Medical Images. *arXi*: 1991.10477v1.

Zhou, Z., Sodha, V., Siddiquee, M.R., Feng, R., Tajbaksh, N., Gotway, M.B., and Liang, J. (2020). Model Genesis. *Medical Image Analysis*. *arXiv*: 2004.07882v4.

Zunair, H., Rahman, A., Mohammed, N., and Cohen, J.P. (2020). Uniformizing Techniques to Process CT scans with 3D CNNs for Tuberculosis Prediction. *arXiv*: 2007.13224v1.

Appendix



Example workflow training the 3D baseline model on Amazon SageMaker

Step 1: Download Data

Download Dataset

```
# Download CT Data from Github of normal CT scans.
url = "https://github.com/hasibzunair/3D-image-classification-tutorial/releases/download/v0.2/CT-0.zip"
filename = os.path.join(os.getcwd(), "CT-0.zip")
keras.utils.get_file(filename, url)

# Download url of abnormal CT scans.
url = "https://github.com/hasibzunair/3D-image-classification-tutorial/releases/download/v0.2/CT-23.zip"
filename = os.path.join(os.getcwd(), "CT-23.zip")
keras.utils.get_file(filename, url)
```

```
Downloading data from https://github.com/hasibzunair/3D-image-classification-tutorial/releases/download/v0.2/CT-0.zip
1065476096/1065471431 [==============================] - 19s 0us/step
Downloading data from https://github.com/hasibzunair/3D-image-classification-tutorial/releases/download/v0.2/CT-23.zi
p
1045168128/1045162547 [==============================] - 17s 0us/step

'/home/ec2-user/SageMaker/MA5852_A3/CT-23.zip'
```

Step 2: Prepare for data split

```
# Folder "CT-0" consist of CT scans having normal lung tissue,
# no CT-signs of viral pneumonia.
normal_scan_paths = [
    os.path.join(os.getcwd(), "3D_CT/CT-0", x)
    for x in os.listdir("3D_CT/CT-0")
]
# Folder "CT-23" consist of CT scans having several ground-glass opacifications,
# involvement of lung parenchyma.
abnormal_scan_paths = [
    os.path.join(os.getcwd(), "3D_CT/CT-23", x)
    for x in os.listdir("3D_CT/CT-23")
]

print("CT scans with normal lung tissue: " + str(len(normal_scan_paths)))
print("CT scans with abnormal lung tissue: " + str(len(abnormal_scan_paths)))
```

```
CT scans with normal lung tissue: 100
CT scans with abnormal lung tissue: 100
```

Step 3: Apply resizing and normalization functions

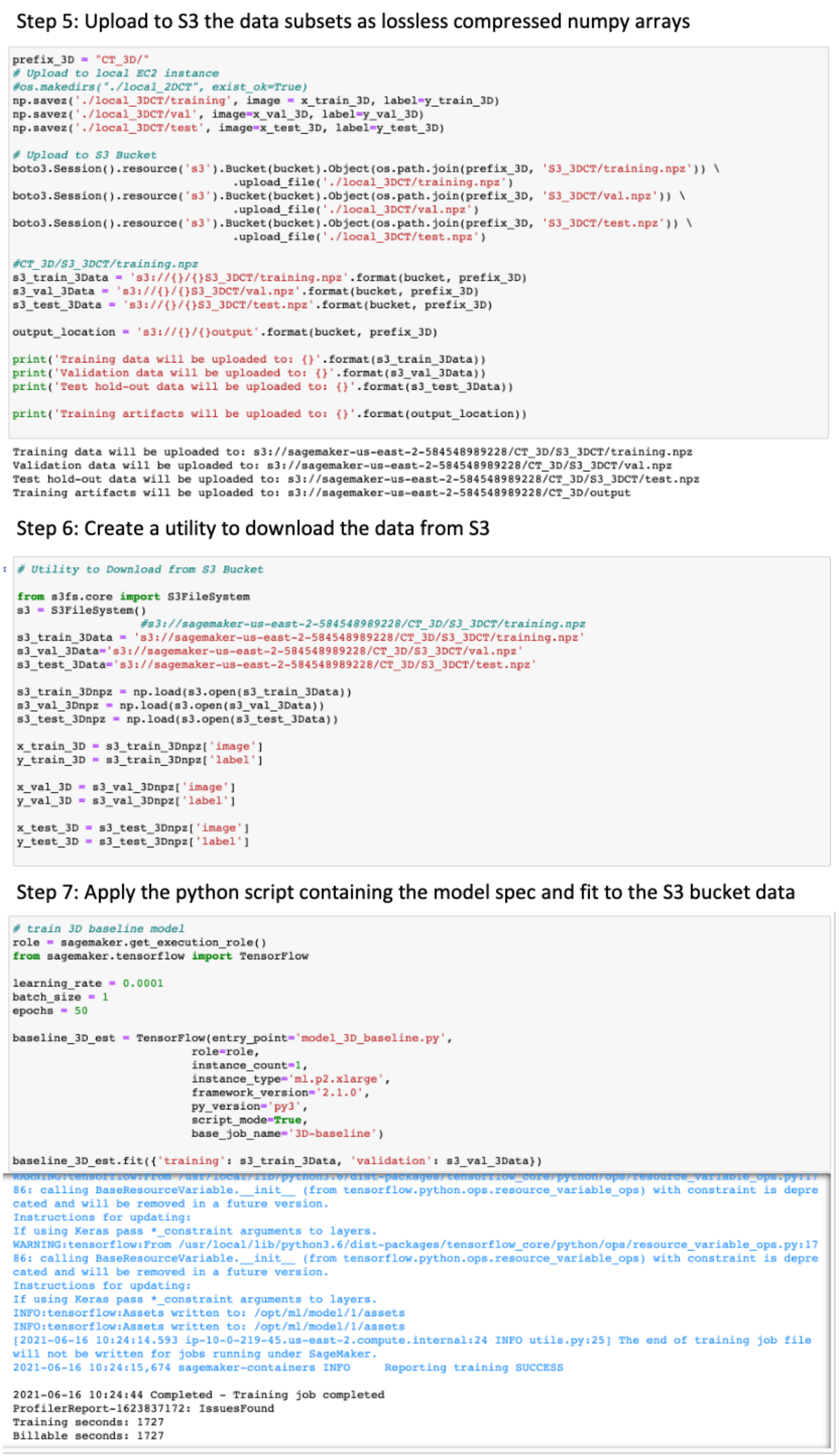
```
# Read and process the scans.
# Each scan is resized across height, width, and depth and rescaled.
abnormal_scans_3D = np.array([process_scan(path) for path in abnormal_scan_paths])
normal_scans_3D = np.array([process_scan(path) for path in normal_scan_paths])

# For the CT scans having presence of viral pneumonia
# assign 1, for the normal ones assign 0.
abnormal_labels_3D = np.array([1 for _ in range(len(abnormal_scans_3D))])
normal_labels_3D = np.array([0 for _ in range(len(normal_scans_3D))])
```

Step 4: Split into training (70%), validation (20%), and hold-out test (10%) subsets.

```
# Split data in the ratio 70:20:10 for training, validation, and hold-out test subsets.
x_train_3D = np.concatenate((abnormal_scans_3D[:70], normal_scans_3D[:70]), axis=0)
y_train_3D = np.concatenate((abnormal_labels_3D[:70], normal_labels_3D[:70]), axis=0)
x_val_3D = np.concatenate((abnormal_scans_3D[70:90], normal_scans_3D[70:90]), axis=0)
y_val_3D = np.concatenate((abnormal_labels_3D[70:90], normal_labels_3D[70:90]), axis=0)
x_test_3D = np.concatenate((abnormal_scans_3D[90:], normal_scans_3D[90:]), axis=0)
y_test_3D = np.concatenate((abnormal_labels_3D[90:], normal_labels_3D[90:]), axis=0)

print(
    "Number of samples in train (%d), validation (%d) and hold-out test (%d) data."
    % (x_train_3D.shape[0], x_val_3D.shape[0], x_test_3D.shape[0])
)
```

```
Number of samples in train (140), validation (40) and hold-out test (20) data.
```

Figure 11: Example workflow of AWS model deployment (part 1 of 3).



Step 5: Upload to S3 the data subsets as lossless compressed numpy arrays

```
prefix_3D = "CT_3D/"
# Upload to local EC2 instance
#os.makedirs("./local_2DCT", exist_ok=True)
np.savez('./local_3DCT/training', image = x_train_3D, label=y_train_3D)
np.savez('./local_3DCT/val', image=x_val_3D, label=y_val_3D)
np.savez('./local_3DCT/test', image=x_test_3D, label=y_test_3D)

# Upload to S3 Bucket
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix_3D, 'S3_3DCT/training.npz')) \
                                .upload_file('./local_3DCT/training.npz')
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix_3D, 'S3_3DCT/val.npz')) \
                                .upload_file('./local_3DCT/val.npz')
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix_3D, 'S3_3DCT/test.npz')) \
                                .upload_file('./local_3DCT/test.npz')

#CT_3D/S3_3DCT/training.npz
s3_train_3Data = 's3://{}/{}S3_3DCT/training.npz'.format(bucket, prefix_3D)
s3_val_3Data = 's3://{}/{}S3_3DCT/val.npz'.format(bucket, prefix_3D)
s3_test_3Data = 's3://{}/{}S3_3DCT/test.npz'.format(bucket, prefix_3D)

output_location = 's3://{}/{}output'.format(bucket, prefix_3D)

print('Training data will be uploaded to: {}'.format(s3_train_3Data))
print('Validation data will be uploaded to: {}'.format(s3_val_3Data))
print('Test hold-out data will be uploaded to: {}'.format(s3_test_3Data))

print('Training artifacts will be uploaded to: {}'.format(output_location))
```

```
Training data will be uploaded to: s3://sagemaker-us-east-2-584548989228/CT_3D/S3_3DCT/training.npz
Validation data will be uploaded to: s3://sagemaker-us-east-2-584548989228/CT_3D/S3_3DCT/val.npz
Test hold-out data will be uploaded to: s3://sagemaker-us-east-2-584548989228/CT_3D/S3_3DCT/test.npz
Training artifacts will be uploaded to: s3://sagemaker-us-east-2-584548989228/CT_3D/output
```

Step 6: Create a utility to download the data from S3

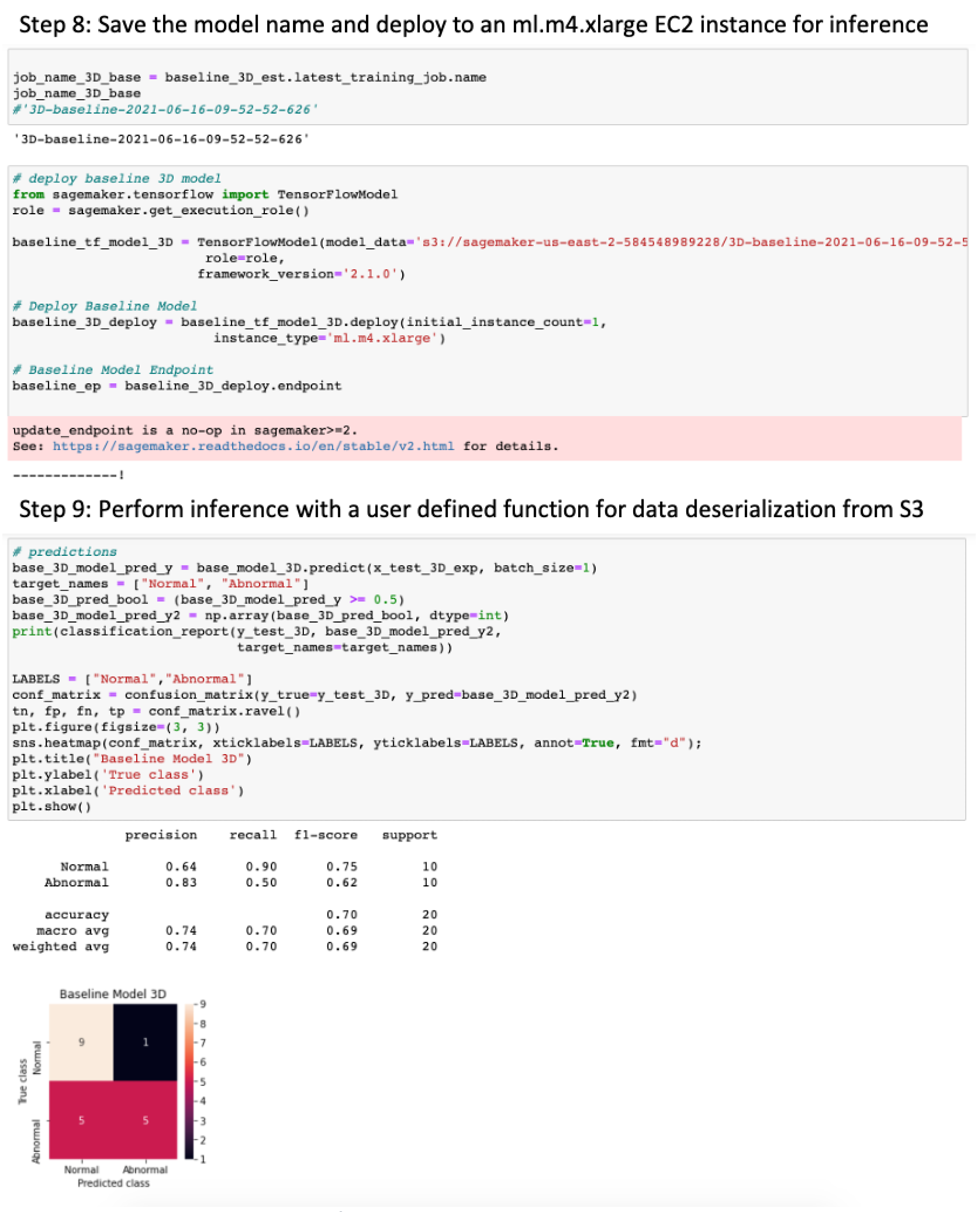
```
# Utility to Download from S3 Bucket

from s3fs.core import S3FileSystem
s3 = S3FileSystem()
                    #s3://sagemaker-us-east-2-584548989228/CT_3D/S3_3DCT/training.npz
s3_train_3Data = 's3://sagemaker-us-east-2-584548989228/CT_3D/S3_3DCT/training.npz'
s3_val_3Data='s3://sagemaker-us-east-2-584548989228/CT_3D/S3_3DCT/val.npz'
s3_test_3Data='s3://sagemaker-us-east-2-584548989228/CT_3D/S3_3DCT/test.npz'

s3_train_3Dnpz = np.load(s3.open(s3_train_3Data))
s3_val_3Dnpz = np.load(s3.open(s3_val_3Data))
s3_test_3Dnpz = np.load(s3.open(s3_test_3Data))

x_train_3D = s3_train_3Dnpz['image']
y_train_3D = s3_train_3Dnpz['label']

x_val_3D = s3_val_3Dnpz['image']
y_val_3D = s3_val_3Dnpz['label']

x_test_3D = s3_test_3Dnpz['image']
y_test_3D = s3_test_3Dnpz['label']
```

Step 7: Apply the python script containing the model spec and fit to the S3 bucket data

```
# train 3D baseline model
role = sagemaker.get_execution_role()
from sagemaker.tensorflow import TensorFlow

learning_rate = 0.0001
batch_size = 1
epochs = 50

baseline_3D_est = TensorFlow(entry_point='model_3D_baseline.py',
                          role=role,
                          instance_count=1,
                          instance_type='ml.p2.xlarge',
                          framework_version='2.1.0',
                          py_version='py3',
                          script_mode=True,
                          base_job_name='3D-baseline')

baseline_3D_est.fit({'training': s3_train_3Data, 'validation': s3_val_3Data})
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:17
86: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is depre
cated and will be removed in a future version.
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/resource_variable_ops.py:17
86: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is depre
cated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: /opt/ml/model/1/assets
INFO:tensorflow:Assets written to: /opt/ml/model/1/assets
[2021-06-16 10:24:14.593 ip-10-0-219-45.us-east-2.compute.internal:24 INFO utils.py:25] The end of training job file
will not be written for jobs running under SageMaker.
2021-06-16 10:24:15,674 sagemaker-containers INFO     Reporting training SUCCESS

2021-06-16 10:24:44 Completed - Training job completed
ProfilerReport-1623837172: IssuesFound
Training seconds: 1727
Billable seconds: 1727
```

Figure 12: AWS model deployment (part 2 of 3).



Step 8: Save the model name and deploy to an ml.m4.xlarge EC2 instance for inference

```
job_name_3D_base = baseline_3D_est.latest_training_job.name
job_name_3D_base
#'3D-baseline-2021-06-16-09-52-52-626'
```

```
'3D-baseline-2021-06-16-09-52-52-626'
```

```
# deploy baseline 3D model
from sagemaker.tensorflow import TensorFlowModel
role = sagemaker.get_execution_role()

baseline_tf_model_3D = TensorFlowModel(model_data='s3://sagemaker-us-east-2-584548989228/3D-baseline-2021-06-16-09-52-5
                        role=role,
                       framework_version='2.1.0')

# Deploy Baseline Model
baseline_3D_deploy = baseline_tf_model_3D.deploy(initial_instance_count=1,
                         instance_type='ml.m4.xlarge')

# Baseline Model Endpoint
baseline_ep = baseline_3D_deploy.endpoint
```

```
update_endpoint is a no-op in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
```

```
-------------!
```

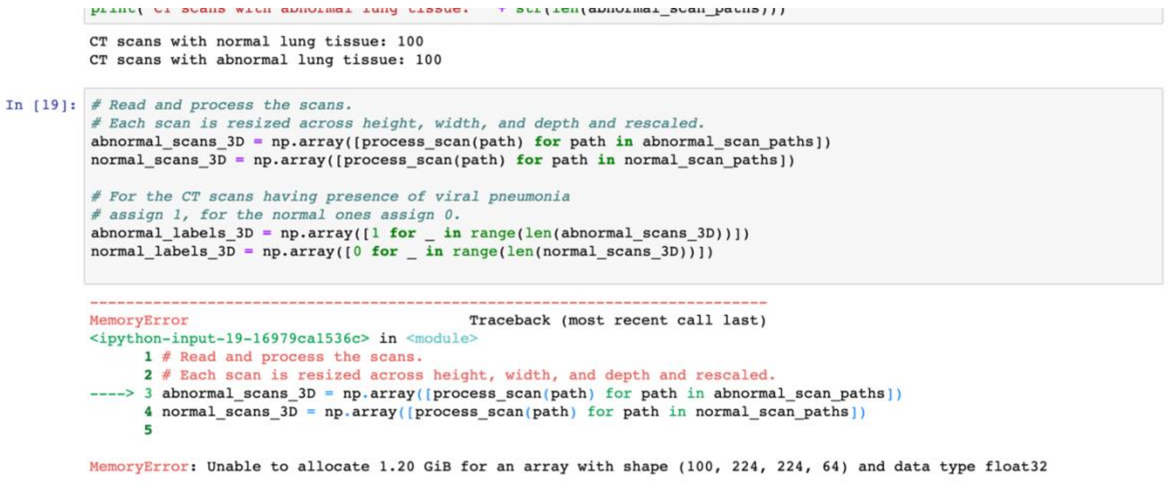
Step 9: Perform inference with a user defined function for data deserialization from S3

```
# predictions
base_3D_model_pred_y = base_model_3D.predict(x_test_3D_exp, batch_size=1)
target_names = ["Normal", "Abnormal"]
base_3D_pred_bool = (base_3D_model_pred_y >= 0.5)
base_3D_model_pred_y2 = np.array(base_3D_pred_bool, dtype=int)
print(classification_report(y_test_3D, base_3D_model_pred_y2,
                            target_names=target_names))

LABELS = ["Normal","Abnormal"]
conf_matrix = confusion_matrix(y_true=y_test_3D, y_pred=base_3D_model_pred_y2)
tn, fp, fn, tp = conf_matrix.ravel()
plt.figure(figsize=(3, 3))
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d");
plt.title("Baseline Model 3D")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

```
              precision    recall  f1-score   support

      Normal       0.64      0.90      0.75        10
    Abnormal       0.83      0.50      0.62        10

    accuracy                           0.70        20
   macro avg       0.74      0.70      0.69        20
weighted avg       0.74      0.70      0.69        20
```

Figure 13: AWS model deployment (part 3 of 3).

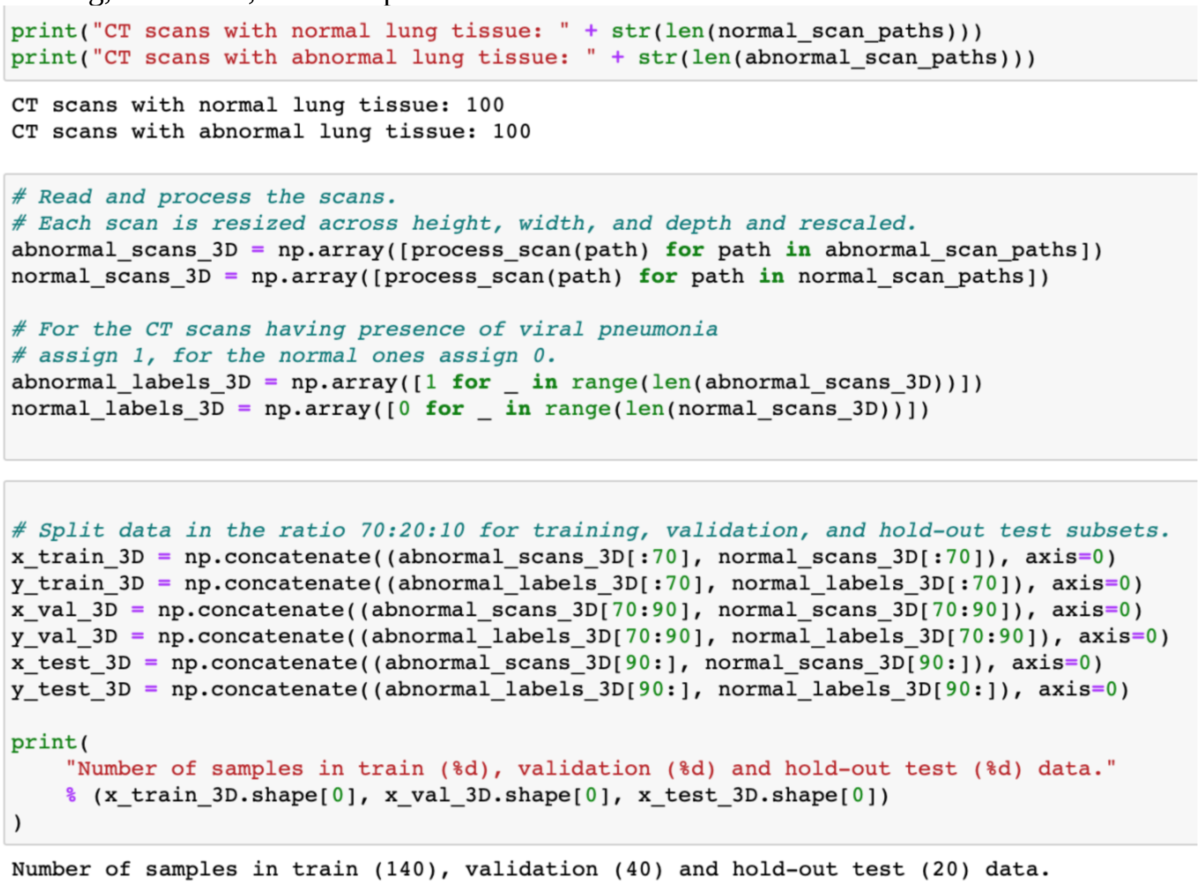
Computational/RAM restrictions on Amazon Samemaker EC2 instance: CT stacks (3D volumes) could not be reshaped to (100,224,224,64), i.e. had to be downsized to (100,128,128,64).



```
print("CT scans with abnormal lung tissue: " + str(len(abnormal_scan_paths)))

CT scans with normal lung tissue: 100
CT scans with abnormal lung tissue: 100

In [19]: # Read and process the scans.
         # Each scan is resized across height, width, and depth and rescaled.
         abnormal_scans_3D = np.array([process_scan(path) for path in abnormal_scan_paths])
         normal_scans_3D = np.array([process_scan(path) for path in normal_scan_paths])

         # For the CT scans having presence of viral pneumonia
         # assign 1, for the normal ones assign 0.
         abnormal_labels_3D = np.array([1 for _ in range(len(abnormal_scans_3D))])
         normal_labels_3D = np.array([0 for _ in range(len(normal_scans_3D))])

---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-19-16979ca1536c> in <module>
      1 # Read and process the scans.
      2 # Each scan is resized across height, width, and depth and rescaled.
----> 3 abnormal_scans_3D = np.array([process_scan(path) for path in abnormal_scan_paths])
      4 normal_scans_3D = np.array([process_scan(path) for path in normal_scan_paths])
      5

MemoryError: Unable to allocate 1.20 GiB for an array with shape (100, 224, 224, 64) and data type float32
```

Figure 14: Computational error when attempting to resize the 512x512x64 CT images into 224x224x64 on a SageMaker EC2 instance.

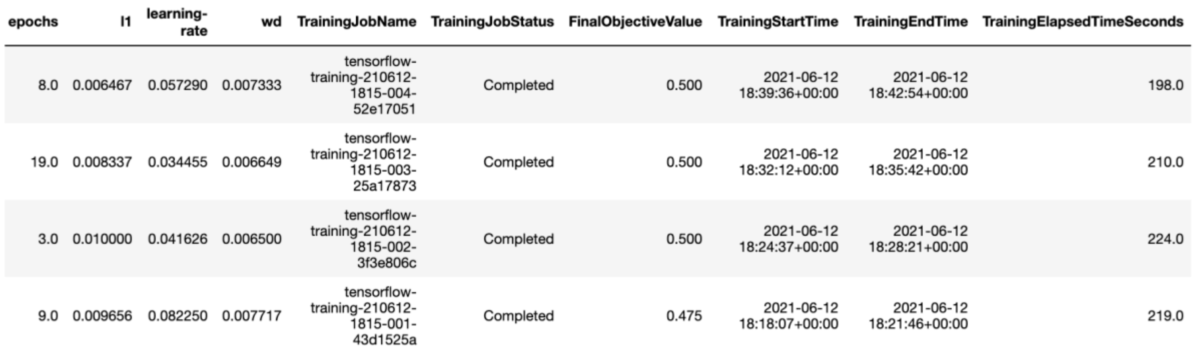
Training, validation, and test splits of the data.



```
print("CT scans with normal lung tissue: " + str(len(normal_scan_paths)))
print("CT scans with abnormal lung tissue: " + str(len(abnormal_scan_paths)))

CT scans with normal lung tissue: 100
CT scans with abnormal lung tissue: 100

# Read and process the scans.
# Each scan is resized across height, width, and depth and rescaled.
abnormal_scans_3D = np.array([process_scan(path) for path in abnormal_scan_paths])
normal_scans_3D = np.array([process_scan(path) for path in normal_scan_paths])

# For the CT scans having presence of viral pneumonia
# assign 1, for the normal ones assign 0.
abnormal_labels_3D = np.array([1 for _ in range(len(abnormal_scans_3D))])
normal_labels_3D = np.array([0 for _ in range(len(normal_scans_3D))])


# Split data in the ratio 70:20:10 for training, validation, and hold-out test subsets.
x_train_3D = np.concatenate((abnormal_scans_3D[:70], normal_scans_3D[:70]), axis=0)
y_train_3D = np.concatenate((abnormal_labels_3D[:70], normal_labels_3D[:70]), axis=0)
x_val_3D = np.concatenate((abnormal_scans_3D[70:90], normal_scans_3D[70:90]), axis=0)
y_val_3D = np.concatenate((abnormal_labels_3D[70:90], normal_labels_3D[70:90]), axis=0)
x_test_3D = np.concatenate((abnormal_scans_3D[90:], normal_scans_3D[90:]), axis=0)
y_test_3D = np.concatenate((abnormal_labels_3D[90:], normal_labels_3D[90:]), axis=0)

print(
    "Number of samples in train (%d), validation (%d) and hold-out test (%d) data."
    % (x_train_3D.shape[0], x_val_3D.shape[0], x_test_3D.shape[0])
)

Number of samples in train (140), validation (40) and hold-out test (20) data.
```

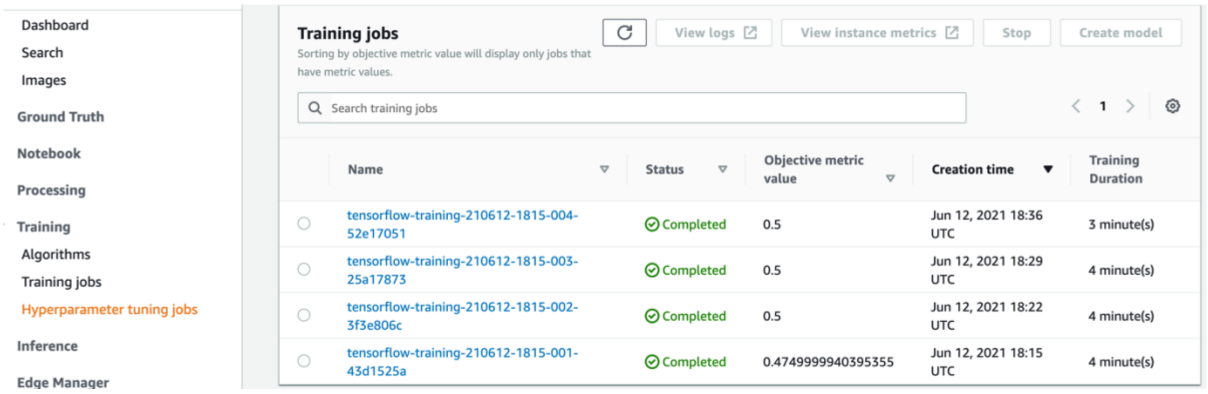
Figure 15: Protocol to split the data into training (70%), validation (20%), and hold-out test (10%) sets.

Hyperparameter tuning results in AWS consol.



epochs	l1	learning-rate	wd	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	TrainingElapsedTimeSeconds
8.0	0.006467	0.057290	0.007333	tensorflow-training-210612-1815-004-52e17051	Completed	0.500	2021-06-12 18:39:36+00:00	2021-06-12 18:42:54+00:00	198.0
19.0	0.008337	0.034455	0.006649	tensorflow-training-210612-1815-003-25a17873	Completed	0.500	2021-06-12 18:32:12+00:00	2021-06-12 18:35:42+00:00	210.0
3.0	0.010000	0.041626	0.006500	tensorflow-training-210612-1815-002-3f3e806c	Completed	0.500	2021-06-12 18:24:37+00:00	2021-06-12 18:28:21+00:00	224.0
9.0	0.009656	0.082250	0.007717	tensorflow-training-210612-1815-001-43d1525a	Completed	0.475	2021-06-12 18:18:07+00:00	2021-06-12 18:21:46+00:00	219.0

Figure 16: Hyperparameter tuning job results.



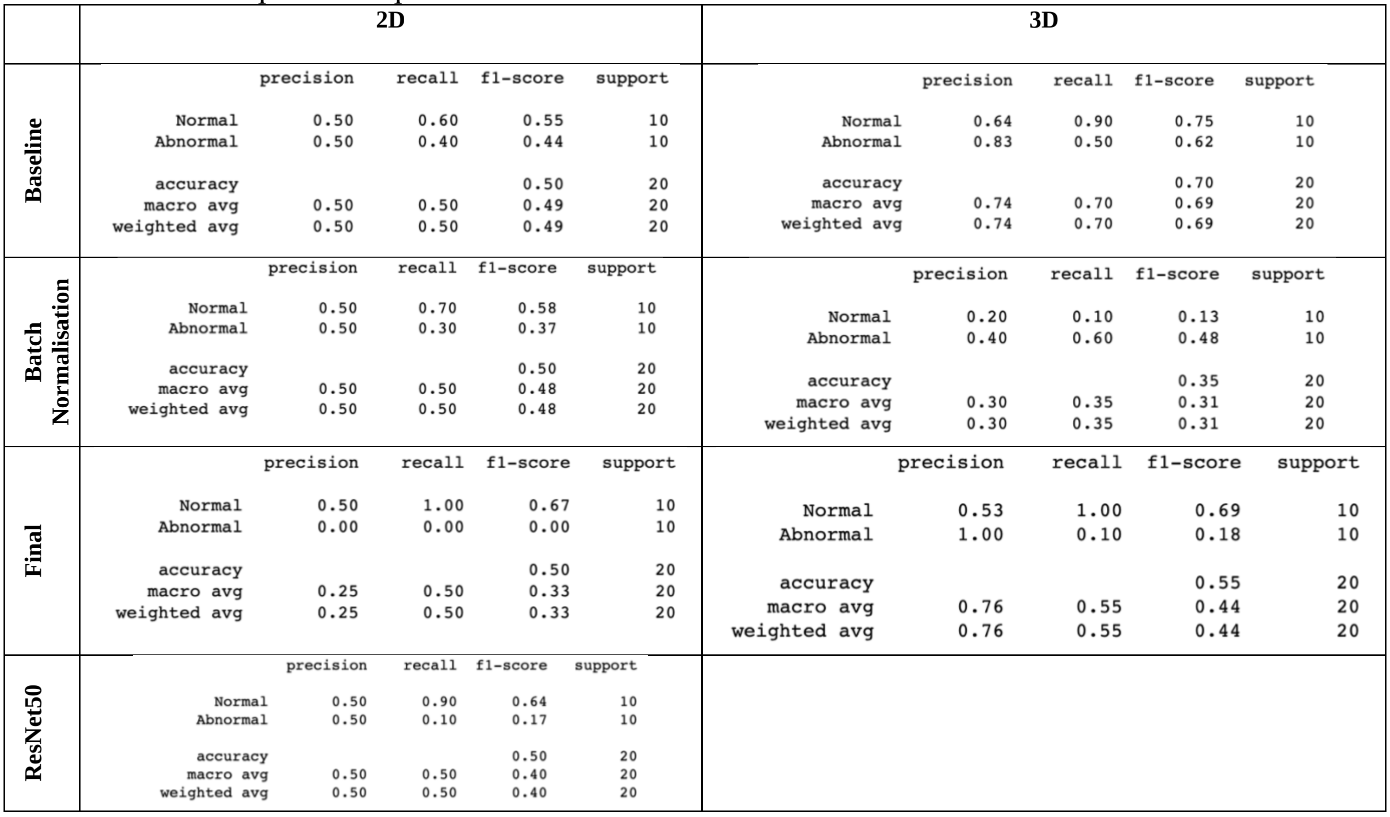
Training jobs

Sorting by objective metric value will display only jobs that have metric values.

Name	Status	Objective metric value	Creation time	Training Duration
tensorflow-training-210612-1815-004-52e17051	Completed	0.5	Jun 12, 2021 18:36 UTC	3 minute(s)
tensorflow-training-210612-1815-003-25a17873	Completed	0.5	Jun 12, 2021 18:29 UTC	4 minute(s)
tensorflow-training-210612-1815-002-3f3e806c	Completed	0.5	Jun 12, 2021 18:22 UTC	4 minute(s)
tensorflow-training-210612-1815-001-43d1525a	Completed	0.4749999940395355	Jun 12, 2021 18:15 UTC	4 minute(s)

Figure 17: Hyperparameter tuning job on AWS.

All models' predictive performance metrics for each class



	2D	3D
Baseline	see below	see below
Batch Normalisation	see below	see below
Final	see below	see below
ResNet50	see below	

Baseline — 2D

```
              precision    recall  f1-score   support

      Normal       0.50      0.60      0.55        10
    Abnormal       0.50      0.40      0.44        10

    accuracy                           0.50        20
   macro avg       0.50      0.50      0.49        20
weighted avg       0.50      0.50      0.49        20
```

Baseline — 3D

```
              precision    recall  f1-score   support

      Normal       0.64      0.90      0.75        10
    Abnormal       0.83      0.50      0.62        10

    accuracy                           0.70        20
   macro avg       0.74      0.70      0.69        20
weighted avg       0.74      0.70      0.69        20
```

Batch Normalisation — 2D

```
              precision    recall  f1-score   support

      Normal       0.50      0.70      0.58        10
    Abnormal       0.50      0.30      0.37        10

    accuracy                           0.50        20
   macro avg       0.50      0.50      0.48        20
weighted avg       0.50      0.50      0.48        20
```

Batch Normalisation — 3D

```
              precision    recall  f1-score   support

      Normal       0.20      0.10      0.13        10
    Abnormal       0.40      0.60      0.48        10

    accuracy                           0.35        20
   macro avg       0.30      0.35      0.31        20
weighted avg       0.30      0.35      0.31        20
```

Final — 2D

```
              precision    recall  f1-score   support

      Normal       0.50      1.00      0.67        10
    Abnormal       0.00      0.00      0.00        10

    accuracy                           0.50        20
   macro avg       0.25      0.50      0.33        20
weighted avg       0.25      0.50      0.33        20
```

Final — 3D

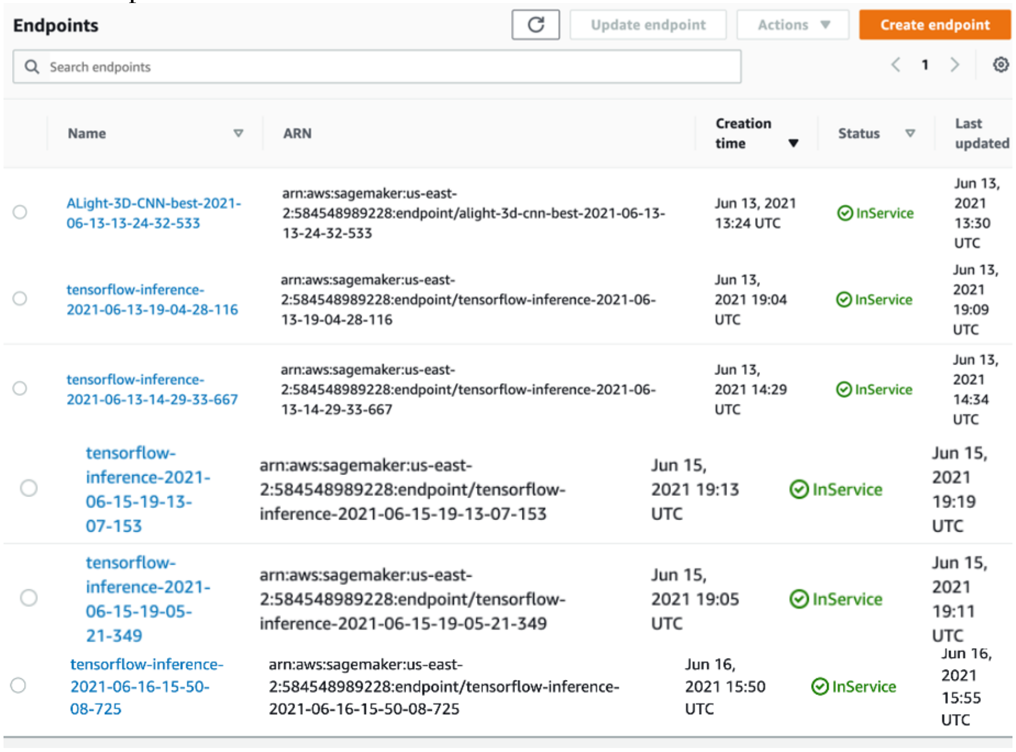
```
              precision    recall  f1-score   support

      Normal       0.53      1.00      0.69        10
    Abnormal       1.00      0.10      0.18        10

    accuracy                           0.55        20
   macro avg       0.76      0.55      0.44        20
weighted avg       0.76      0.55      0.44        20
```

ResNet50 — 2D

```
              precision    recall  f1-score   support

      Normal       0.50      0.90      0.64        10
    Abnormal       0.50      0.10      0.17        10

    accuracy                           0.50        20
   macro avg       0.50      0.50      0.40        20
weighted avg       0.50      0.50      0.40        20
```

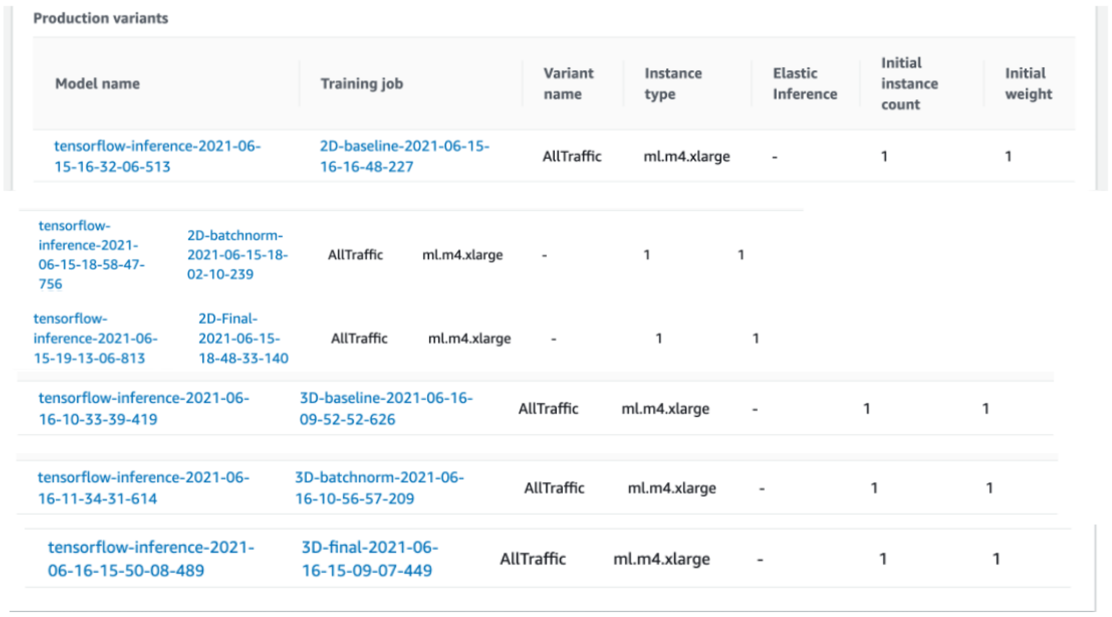
Figure 18: Prediction evaluation for all models considered for both class objects.

AWS Model Endpoints for Inference



Name	ARN	Creation time	Status	Last updated
ALight-3D-CNN-best-2021-06-13-13-24-32-533	arn:aws:sagemaker:us-east-2:584548989228:endpoint/alight-3d-cnn-best-2021-06-13-13-24-32-533	Jun 13, 2021 13:24 UTC	InService	Jun 13, 2021 13:30 UTC
tensorflow-inference-2021-06-13-19-04-28-116	arn:aws:sagemaker:us-east-2:584548989228:endpoint/tensorflow-inference-2021-06-13-19-04-28-116	Jun 13, 2021 19:04 UTC	InService	Jun 13, 2021 19:09 UTC
tensorflow-inference-2021-06-13-14-29-33-667	arn:aws:sagemaker:us-east-2:584548989228:endpoint/tensorflow-inference-2021-06-13-14-29-33-667	Jun 13, 2021 14:29 UTC	InService	Jun 13, 2021 14:34 UTC
tensorflow-inference-2021-06-15-19-13-07-153	arn:aws:sagemaker:us-east-2:584548989228:endpoint/tensorflow-inference-2021-06-15-19-13-07-153	Jun 15, 2021 19:13 UTC	InService	Jun 15, 2021 19:19 UTC
tensorflow-inference-2021-06-15-19-05-21-349	arn:aws:sagemaker:us-east-2:584548989228:endpoint/tensorflow-inference-2021-06-15-19-05-21-349	Jun 15, 2021 19:05 UTC	InService	Jun 15, 2021 19:11 UTC
tensorflow-inference-2021-06-16-15-50-08-725	arn:aws:sagemaker:us-east-2:584548989228:endpoint/tensorflow-inference-2021-06-16-15-50-08-725	Jun 16, 2021 15:50 UTC	InService	Jun 16, 2021 15:55 UTC

Figure 19: Model endpoints in service on AWS.


Production variants

Model name	Training job	Variant name	Instance type	Elastic Inference	Initial instance count	Initial weight
tensorflow-inference-2021-06-15-16-32-06-513	2D-baseline-2021-06-15-16-16-48-227	AllTraffic	ml.m4.xlarge	-	1	1
tensorflow-inference-2021-06-15-18-58-47-756	2D-batchnorm-2021-06-15-18-02-10-239	AllTraffic	ml.m4.xlarge	-	1	1
tensorflow-inference-2021-06-15-19-13-06-813	2D-Final-2021-06-15-18-48-33-140	AllTraffic	ml.m4.xlarge	-	1	1
tensorflow-inference-2021-06-16-10-33-39-419	3D-baseline-2021-06-16-09-52-52-626	AllTraffic	ml.m4.xlarge	-	1	1
tensorflow-inference-2021-06-16-11-34-31-614	3D-batchnorm-2021-06-16-10-56-57-209	AllTraffic	ml.m4.xlarge	-	1	1
tensorflow-inference-2021-06-16-15-50-08-489	3D-final-2021-06-16-15-09-07-449	AllTraffic	ml.m4.xlarge	-	1	1

Figure 20: Deployed model endpoints with associated job names.