Demonstration of Hardware Performance Accelerators on real time Large Language models using a Pneumonia Detection CNN model as an example with Intel One API

Abstract

In general, pneumonia affects children under 5 years and adults over 65 years of age which targets the lungs and fills the alveoli (air sacs) with liquid. In this paper, we employ convolutional neural networks (CNNs) of varying configurations on a machine learning based binary classification task with a given dataset of chest X-rays that depicts affected and unaffected cases of pneumonia. This report primarily focuses on putting forth the performances of different simple CNN architectures and selecting the best architecture based on optimum corresponding minimum loss and maximum accuracy which can serve as a viable tool for physicians and the medicine community to correctly identify and diagnose viral, bacterial, fungal-caused and community acquired pneumonia given only the chest X-ray of the patient.

the age of ever-evolving artificial intelligence, Large Language Models (LLMs) have emerged as powerhouses of natural language processing, capable of generating human-quality text, translating languages, and writing different kinds of creative content. However, training and running these complex models requires immense computational resources. This is where Hardware Performance Accelerators (HPAs) step in, acting as the nitrous oxide to the LLM engine, significantly boosting performance and efficiency.

HPAs, like GPUs and FPGAs, offload computationally intensive tasks from the central processing unit (CPU), enabling faster training and inference times for LLMs. This is particularly crucial for large models with billions of parameters, where traditional CPUs can become bottlenecks.

Intel oneAPI: This unified programming model from Intel simplifies the development and deployment of applications across various HPAs, including CPUs, GPUs, and FPGAs. With oneAPI, developers can write a single codebase that seamlessly targets different hardware architectures, maximizing performance and portability.

Large Language Models (LLMs) are revolutionizing AI with their natural language prowess, but training and running them requires immense computational power. This is where Hardware Performance Accelerators (HPAs) come in, acting like turbochargers, significantly speeding up processing. However, harnessing the power of diverse HPAs can be complex. Enter Intel oneAPI, a unified programming model that acts as a translator, simplifying development and deployment across CPUs, GPUs, and FPGAs. Developers write once, code runs anywhere, maximizing performance and portability.

Think faster training, real-time language translation, and energy-efficient AI applications. Intel oneAPI unlocks the true potential of HPAs, empowering developers to build a brighter, more fluent AI future.

Introduction

Pneumonia is a widely occurring severe form of acute respiratory disease that is caused due to infectious agents with high attack rates among individuals that belong to age groups of either of the extreme ends of the lifespan of an average human being. These infectious agents may be viral, fungal or bacterial and the lungs react to the infiltration of these foreign microorganisms by an inflammatory response which causes the bronchioles and alveoli to get filled with liquid. This in turn causes difficulty in respiration for the affected individual. In the case of children under 5 years of age, pneumonia is responsible for over 15% deaths recorded globally, with 920,000 deaths in 2015 alone. In the same year, the US witnessed over 500,000 cases of emergency admissions in hospitals due to pneumonia with 50,000 casualties reported which puts the disease among the top 10 causes of deaths in the US.

There are three main types of pneumonia: a) community acquired, b) viral, and c) bacterial.

Community acquired pneumonia is mainly distinguished from nosocomial acquired pneumonia and roughly 1.5 million people in the US are hospitalized owing to community acquired pneumonia every year. Globally, around 200 million people get affected by viral pneumonia annually, with a 50- 50 infection rate between adults and children. Generally, bacteria are classified as typical or atypical. Typical bacteria can be seen on the Gram strain and cultured through standard media, which, on the other hand, is not seen in atypical class of bacteria. Typical bacteria-caused pneumonia can be listed as staphylococcus aureus, streptococcus pneumoniae, haemophilus influenza, etc. whereas atypical bacteria-caused pneumonia is mostly caused by chlamydia pneumoniae, legionella, chlamydia psittacine, and mycoplasma pneumonia. Other types of pneumonia are fungal, aspiration, and hospital-acquired.

The most commonly sought diagnostic technique for all forms of pneumonia involves studying the increased opacity in regions of the lungs as shown by the chest radiograph, or chest X-ray (CXR). The increased opacity is caused due to the inflammation of the lungs with the high amounts of liquid in the affected areas. There can be complications to the diagnosis of pneumonia through CXR because of the possibility of existence of pulmonary enema which is mostly caused by cardiac problems, or internal lung bleeding, lung cancer, or in some patients, atelectasis which results in unilateral collapse or shutdown of a part of a lung or the whole lung itself. In this condition, alveoli are deflated to very low volumes, visible from the increased opacity of the affected part seen in the CXR. Due to these complications, it becomes vital for having trained physicians and specialists, equipped with the patients' clinical record, to study the CXRs at different time frames for comparison and proper diagnosis.

The medical field has witnessed lots of breakthroughs in better diagnosis of diseases through machine learning. In this paper, we use the principles of deep learning, which is a branch of machine learning by employing convolutional neural networks (CNN) trained on normal and pneumonia positive CXR images to correctly identify whether or not a new CXR fed into the network is pneumonia positive when in the field for diagnostic purposes.

Problem Statement:

The concept that we wish to highlight through this project is the real time demonstration of Hardware Performance Accelerators on real time Large Language models. Showing this example model of Pneumonia detection with an image dataset of about 5500 belonging to 3 categories, we can distinctly see the difference in efficiency and performance of the code. Imaging the same being implemented in real time projects and large language model training on in-demand domains like MLOPS, Big Data, Bioinformatics or Generative AI where we can have efficient models that can thus have a lot of practical applications in real time getting us to a new era in these fields.

Methods and Data

In this section, we describe the methods used behind the classification process of CXRs determined to be pneumonia positive or negative. We also specify the dataset used for the task and the technology with which the experiments and computations were carried out.

Table 2. Dataset layout and proportions of images taken for training and testing.

	Normal	Pneumonia
Training	1341	3875
Testing	242	389
Total	1583	4264

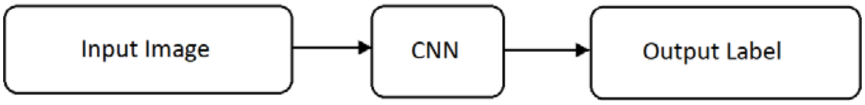
Image augmentation

Deep learning models usually require enormous amounts of data to train on for them to correctly function. In the case of CNNs, thousands upon thousands of images are required which, pragmatically speaking, is a difficult prospect. To solve this, a technique named image augmentation is used. Image augmentation virtually increases the size of an existing dataset to a large extent with techniques like standardization of features (pixel values), whitening transforms, random rotations and shifts, flipping, rescaling, shearing, zooming, etc. In our approach, we rescale, shear, zoom and flip horizontally the training image for a wider variety and augmentation of training data.

Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNN) are one of the most popular deep learning framework models proposed by LeCun et al. which revolutionized deep learning and is also the model we use in our approach. CNNs are mainly used to operate on image data to help classify objects. CNNs have been used to detect face, scene labelling, action recognition, etc.

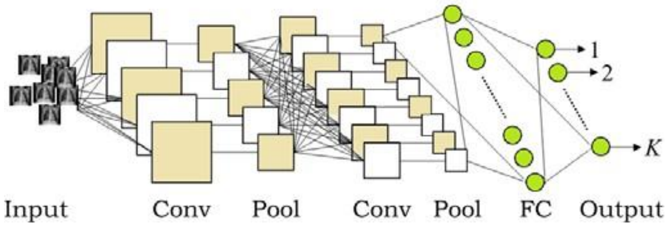
CNNs are composed of two main types of layers - convolutional and dense (here dense refers to the ANN layers), Fig. 3 shows the procedure of usage of CNNs. The main objective of convolutional layers is to find features in an image using feature detectors which are 2D matrices called kernels or filters. This obtains a feature map which stores the spatial relationships between pixels of the input image.



These spatial relationships are vital to finding differences between two objects in two images. A convolution operation of two functions $ff(t)$ and $g(t)$ can be defined as,

$$ff(t) * g(t) \triangleq \overset{--}{\text{�}}\ \underset{(ff*gg)(t)}{\underset{\text{�������������}}{ff(r)g(t-r)dr}}$$

After the convolution layer, a pooling layer is used to further down sample the detected features which helps to detect the image whether it is tilted, stretched or rotated which helps provide spatial invariance to these models. Pooling reduces the parameters by 75% and also prevents overfitting. The next layer is to flatten all the pooled values and arrange them into a vertical array to be fed into an ANN (the dense layers). This step is called the full connection (FC). Finally, the values are propagated through the ANN to get an output at the output layer. Figure 4 displays a full convolutional neural network.



Software and hardware

All the various architectures of CNNs that were trained were done through Python3 with the Keras library (that uses TensorFlow backend) high level API for construction of neural networks on a workstation with Intel i5 11th generation processor with 16 GB RAM. We have used two well-known python3 interpreters that include Google Colaboratory and Jupyter Notebook. Google Colaboratory shall be our normal python interpreter that shall run the sample model of Pneumonia Detection using CNN and give us a result. On the contrary, the Jupyter Notebook had been integrated with Intel oneAPI, a hardware performance accelerator. It is a cross platform software for heterogenous computing that is available open source to be integrated for our projects made thus increasing their performance and efficiency.

Experiments and analysis

We carried out experiments with different CNN architectures pertaining to the number of convolutional layers, number of dense layers, inclusion, and exclusions of regularizations like L1, L2, Batch Norm (Batch Normalization) and Dropout, image input sizes, kernel sizes, pooling matrix sizes, and compared each architecture's performance based on the maximum accuracy achieved during training in addition to the least cross entropy loss encountered during training. Table 3 specifies the meanings of the abbreviations used in Table 4 which contains information of 15 different CNN architectures and their maximum accuracy and least loss along with the time taken in seconds to train them. One thing to note is that the number of nodes in each ANN layer is taken to be 128 except for the last output layer having 2 output nodes.

Figure 4 summarizes the statistics of each architecture based on LVCEL, MVA and TT. We have necessary diagrams that plots the training accuracy and loss achieved per each epoch for each architecture. From Table 4 we notice that architectures 10 and 11 perform poorly with respect to MVA and LVCEL. This may be due to the L1 regularization added on these architectures (which is not applied on any other case) which restricts the model from correctly learning the relationships in the dataset. We also notice a directly proportional relationship between architecture size and TT. However, TT highly depends upon IS and as we see in architectures 5, 6 and 13, when IS. TT is greatly reduced. It is helpful to note that TT is also directly proportional to the how computationally expensive training the model is.

Training neural networks takes a lot of computational power, and the longer it runs, the more intensive the task is.

Finally, we can reduce from more statistical analysis that MVA and LVCEL do not improve drastically with increasing architecture size, whereas TT increases drastically. This brings us to the simple conclusion that for a given dataset, the simpler the architecture, the better. Adding extra layers to the neural network may cause it to underfit the data as it starts to attempt to detect features which are not really existent. Based on MVA, LVCEL and TT, for our purposes, we select architecture 5 for further consideration as the most ideal model and we use this simple architecture to evaluate model performance on the CXR dataset. Using a CNN model to simply run epochs on our model, we have come to a conclusion that a normal python interpreter like Google Colaboratory takes more than 10 mins to run only 2 epochs where as the one integrated with Intel oneAPI takes only about 3 mins. This would give us a boost of about 30% in performance that changes everything when implemented in real time training of large language models. Usage of such Hardware Performance Accelerators can give us a huge boost in the performance metrics of our real time model, increasing speed and efficiency of the model.

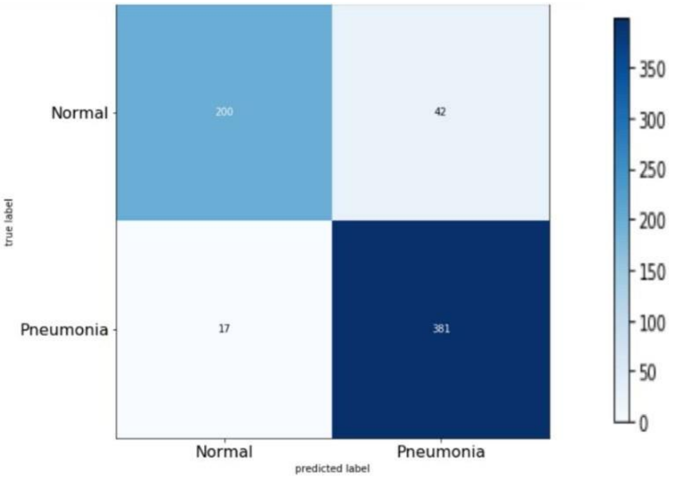
Results of selected architecture

We evaluate the model performance of architecture 5 on the test set specified by Table 2. As the evaluation metrics, we use confusion matrix and calculate the Sensitivity and Specificity as follows,

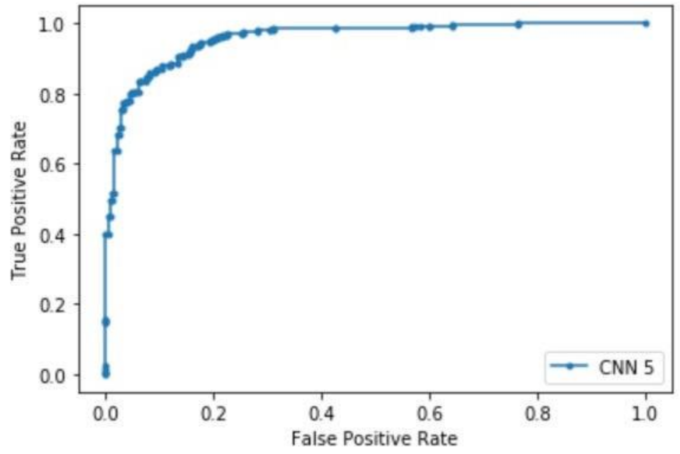
$$Sensiitiiviity = {}^{TP}\text{�}_{(TP + FN)}$$

$$Speciiffiiciity = \ {}^{TN}\text{�}_{(TN + FP)}$$

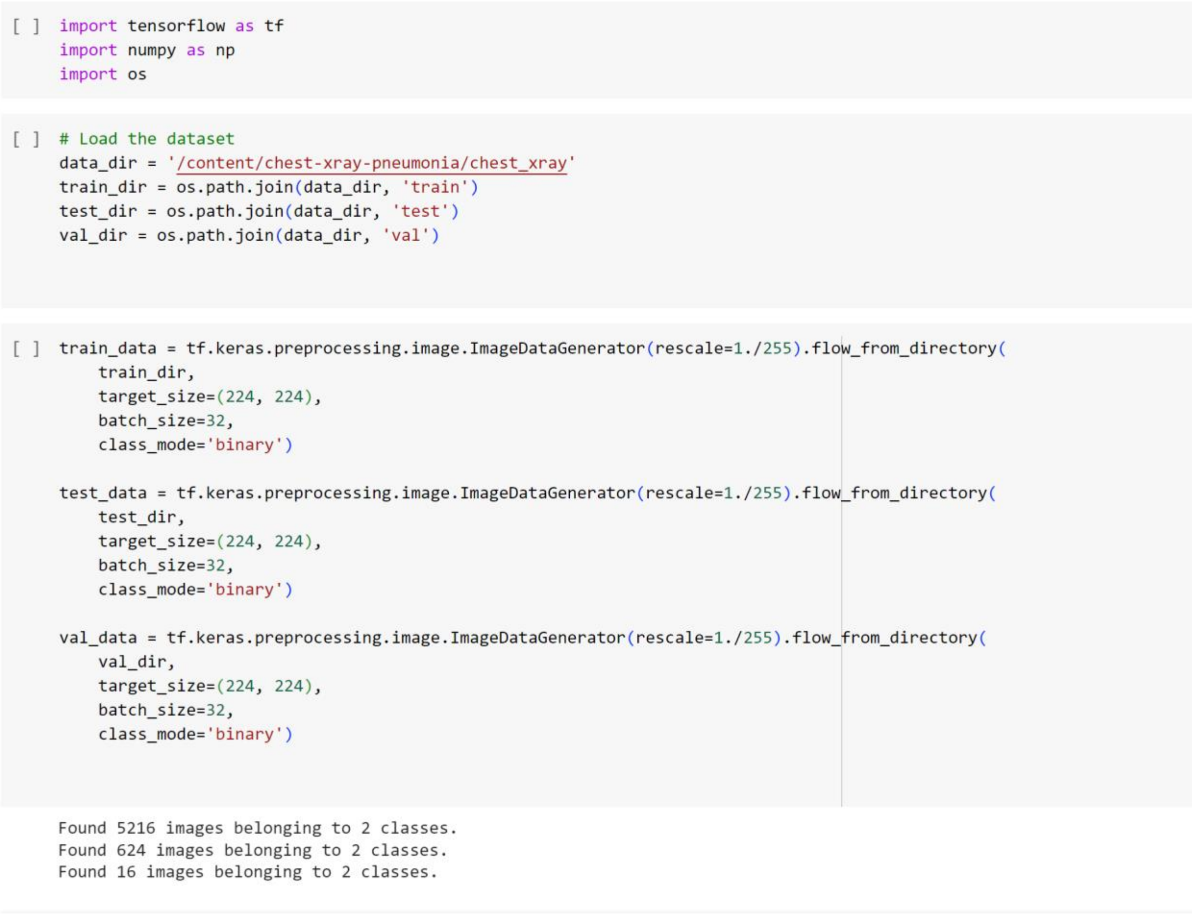
where TP, TN, FN, FP refer to True Positive, True Negative, False Negative, and False Positive, respectively.



Hence, we calculate $Sensiitiiviity$ = 0.9007 and $Speciif fiiciity$ = 0.9216. Both these metrics are above 90% which is a good indicator that the model performs excellently in both, having the ability to correctly identify most of the positive pneumonia cases, and also the ability of ruling out the negative cases. Moreover, the ROC curve illustrated in whose area under the curve (AUC) is 0.9582 which is high.



Reference images of the code:



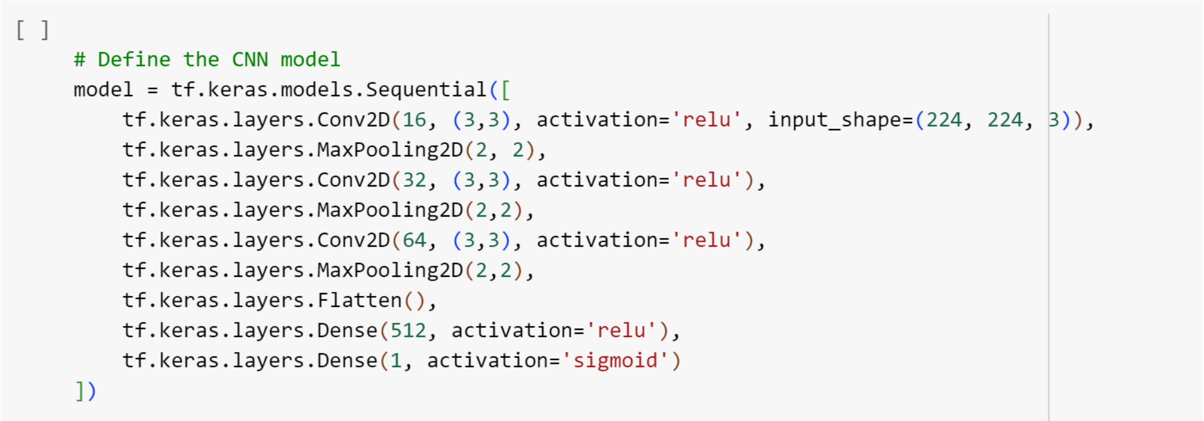
```
[ ] import tensorflow as tf
    import numpy as np
    import os
```

```
[ ] # Load the dataset
    data_dir = '/content/chest-xray-pneumonia/chest_xray'
    train_dir = os.path.join(data_dir, 'train')
    test_dir = os.path.join(data_dir, 'test')
    val_dir = os.path.join(data_dir, 'val')
```

```
[ ] train_data = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255).flow_from_directory(
        train_dir,
        target_size=(224, 224),
        batch_size=32,
        class_mode='binary')

    test_data = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255).flow_from_directory(
        test_dir,
        target_size=(224, 224),
        batch_size=32,
        class_mode='binary')

    val_data = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255).flow_from_directory(
        val_dir,
        target_size=(224, 224),
        batch_size=32,
        class_mode='binary')
```

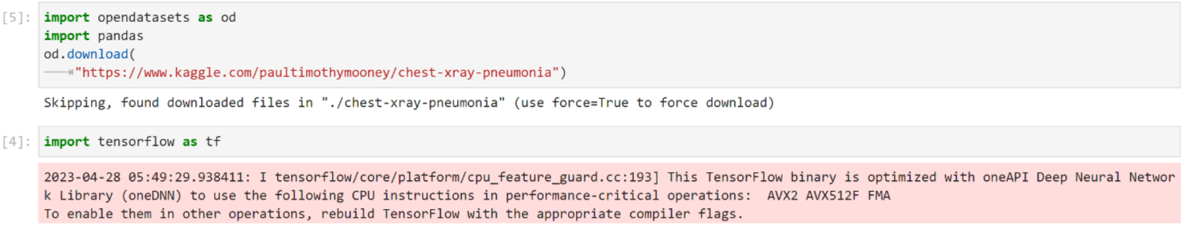
```
    Found 5216 images belonging to 2 classes.
    Found 624 images belonging to 2 classes.
    Found 16 images belonging to 2 classes.
```

The CNN MODEL used:



```
[ ]
    # Define the CNN model
    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(224, 224, 3)),
        tf.keras.layers.MaxPooling2D(2, 2),
        tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(512, activation='relu'),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ])
```

Integration of Intel oneAPI on Jupyter Notebook



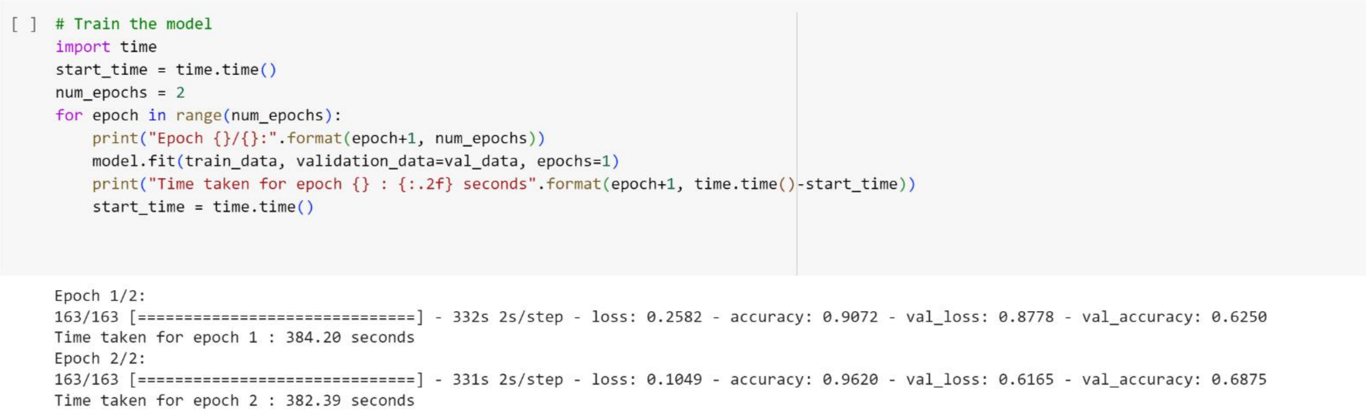
```
[5]: import opendatasets as od
     import pandas
     od.download(
         "https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia")

     Skipping, found downloaded files in "./chest-xray-pneumonia" (use force=True to force download)

[4]: import tensorflow as tf

     2023-04-28 05:49:29.938411: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Networ
     k Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX2 AVX512F FMA
     To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

Intel oneAPI

oneAPI is an open standard, adopted by Intel, for a unified application programming interface (API) intended to be used across different computing accelerator (coprocessor) architectures, including GPUs, AI accelerators and field-programmable gate arrays. It is built using DPC++ and Khronos SYCL and is integrated by the cloud console of Intel. The jupyter notebook that we have used here is directly connected with the Intel Developer Cloud and through it, we shall be integrating the software of Intel's oneAPI through the Intel oneAPI base toolkit and Intel oneAPI's oneDAL [Data Analytics Library].

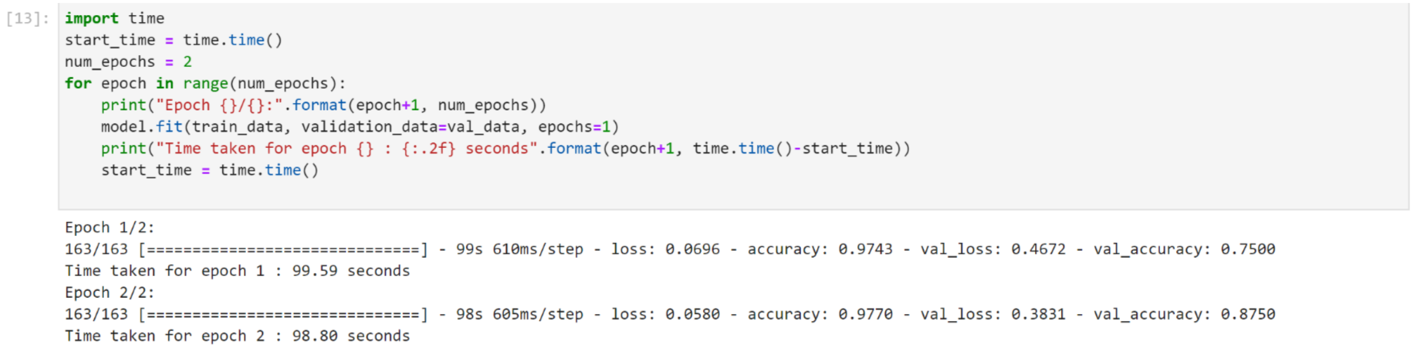
Performance comparison

Google colaboratory:



```
[ ] # Train the model
    import time
    start_time = time.time()
    num_epochs = 2
    for epoch in range(num_epochs):
        print("Epoch {}/{}:".format(epoch+1, num_epochs))
        model.fit(train_data, validation_data=val_data, epochs=1)
        print("Time taken for epoch {} : {:.2f} seconds".format(epoch+1, time.time()-start_time))
        start_time = time.time()

    Epoch 1/2:
    163/163 [==============================] - 332s 2s/step - loss: 0.2582 - accuracy: 0.9072 - val_loss: 0.8778 - val_accuracy: 0.6250
    Time taken for epoch 1 : 384.20 seconds
    Epoch 2/2:
    163/163 [==============================] - 331s 2s/step - loss: 0.1049 - accuracy: 0.9620 - val_loss: 0.6165 - val_accuracy: 0.6875
    Time taken for epoch 2 : 382.39 seconds
```

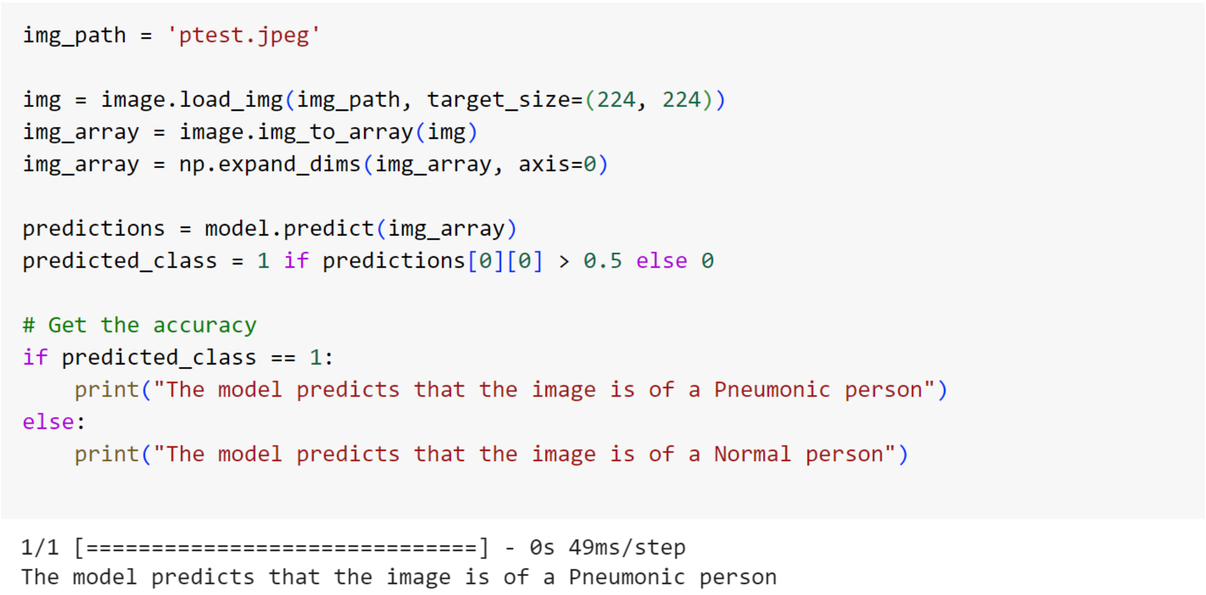
Jupyter Notebook integrated with Intel oneAPI:



```
[13]: import time
      start_time = time.time()
      num_epochs = 2
      for epoch in range(num_epochs):
          print("Epoch {}/{}:".format(epoch+1, num_epochs))
          model.fit(train_data, validation_data=val_data, epochs=1)
          print("Time taken for epoch {} : {:.2f} seconds".format(epoch+1, time.time()-start_time))
          start_time = time.time()

      Epoch 1/2:
      163/163 [==============================] - 99s 610ms/step - loss: 0.0696 - accuracy: 0.9743 - val_loss: 0.4672 - val_accuracy: 0.7500
      Time taken for epoch 1 : 99.59 seconds
      Epoch 2/2:
      163/163 [==============================] - 98s 605ms/step - loss: 0.0580 - accuracy: 0.9770 - val_loss: 0.3831 - val_accuracy: 0.8750
      Time taken for epoch 2 : 98.80 seconds
```

As we can notice from the above images, we can see more than 15% increase In the testing model's accuracy when integrated with Intel. Also, the time taken to run the epochs or one complete cycle in training the dataset in whole is clearly notable.

Sample test code:



```
img_path = 'ptest.jpeg'

img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)

predictions = model.predict(img_array)
predicted_class = 1 if predictions[0][0] > 0.5 else 0

# Get the accuracy
if predicted_class == 1:
    print("The model predicts that the image is of a Pneumonic person")
else:
    print("The model predicts that the image is of a Normal person")

1/1 [==============================] - 0s 49ms/step
The model predicts that the image is of a Pneumonic person
```

Conclusion:

By this project, we can show the real time implementation of Hardware Performance Accelerators on a real time Deep Learning model and thus, prove how efficient these softwares may be when they are integrated in solving real time problems. We attempt to find a simpler approach for pneumonia detection based on CXRs by comparing the performances of 15 different CNN architectures trained on the same dataset. Based on our findings, we select the most ideal model which is easy to train (less computationally expensive and quicker), intelligible, and has one of the best performance metrics. The metrics of the selected architecture compare to some of the state-of-the-art architectures trained on

CXRs that goes ahead to prove that striving for the simplification of CNN architectures is crucial for intelligibility without compromising accuracy and quality of performance. As learnt from the findings in this research work, we recommend that there be more research into fine-tuning simpler architectures to gain even higher levels of accuracy. We admit that the simple architecture 5 that was chosen after thorough experimentation can perform even better if fine-tuned and experimented upon more. Future work on detection of pneumonia may also be done through multimodal learning where symptoms of the patients as text could be taken by the model along with the CXR image for even better diagnosis. We propose to demonstrate the effectiveness of our Pneumonia Detection model by integrating Intel oneAPI, a hardware performance accelerator, alongside the traditional CNN model. This approach aims to provide compelling evidence of the software's efficiency by highlighting a clear contrast between the two implementations.