Holon Institute of Technology

# Abnormality Detection in Musculoskeletal Radiographs

Alon Avrahami

David Chernin

Yair Hanina

# Agenda

- Motivation to solve our problem
- Introduce MURA
- Model and Architecture
- Results and Analysis
- Conclusions

# Motivation

## Why we choose this project?

Musculoskeletal conditions affect more than 1.7 billion people worldwide based on a study by Global Burden Disease, and a major cause of disability.

This is a critical radiological task, a study interpreted as normal rules out disease and can eliminate the need for patients to undergo further diagnostic procedures or interventions.

This is a common problem in the AI industry, also, this problem is related to health care, which is very important to humanity and we are always want to get the best results as we can - we can save human life.

# Objective

## Main objective

The main objective of the project is to develop a convolution neural network model that automatically detects abnormalities and normalities in musculoskeletal radiographs.

Specific objectives:

- Develop a model based on Keras DenseNet169, trained on “imagenet” dataset, and see the results using MURA dataset for train and test.
- In order to improve the base model, we will try to “Fine-Tune” DenseNet169 using image augmentations, modifying layers, using dynamic learning rate, modified loss function, etc...

# MURA

## **Mu**sculoskeletal **Ra**diographs

Stanford University- Department of Computer Science, Medicine, and Radiology introduced a public dataset MURA of musculoskeletal radiographs from Stanford Hospital which is the largest dataset with 40,561 images from 14,863 upper extremity studies.

The MURA abnormality detection task is a binary classification task, the input is an image of radiologist, with each study containing one or more images and the expected output is a binary label $y \in \{0, 1\}$ indicating whether the study is normal or abnormal, respectively.[1]

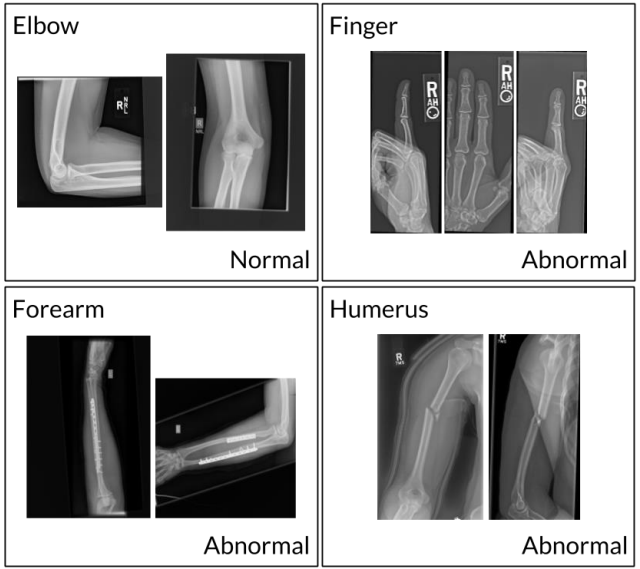
# MURA

## **Mu**sculoskeletal **Ra**diographs

Each radiographic images belongs to one of seven types of bones: elbow, finger, forearm, hand, humerus, shoulder, and wrist.

In our model we will try to focus on two types of bones due to compute power limitations.

The dataset splitted into training and validation sets, with no overlap between the datasets.



# MURA

## Performance evaluation

To evaluate our model performance and get a robust estimate of the model prediction, we will compare our results against Stanford radiologist performance.

With 9 years of experience on average, The radiologists individually retrospectively reviewed and labeled each study in the test set as a DICOM file as normal or abnormal in the clinical reading room environment using the PACS (Picture Archive and Communication System) system.

| | Radiologist 1 | Radiologist 2 | Radiologist 3 |
|---|---|---|---|
| Elbow | 0.850 (0.830, 0.871) | 0.710 (0.674, 0.745) | 0.719 (0.685, 0.752) |
| Finger | 0.304 (0.249, 0.358) | 0.403 (0.339, 0.467) | 0.410 (0.358, 0.463) |
| Forearm | 0.796 (0.772, 0.821) | 0.802 (0.779, 0.825) | 0.798 (0.774, 0.822) |
| Hand | 0.661 (0.623, 0.698) | 0.927 (0.917, 0.937) | 0.789 (0.762, 0.815) |
| Humerus | 0.867 (0.850, 0.883) | 0.733 (0.703, 0.764) | 0.933 (0.925, 0.942) |
| Shoulder | 0.864 (0.847, 0.881) | 0.791 (0.765, 0.816) | 0.864 (0.847, 0.881) |
| Wrist | 0.791 (0.766, 0.817) | 0.931 (0.922, 0.940) | 0.931 (0.922, 0.940) |
| Overall | 0.731 (0.726, 0.735) | 0.763 (0.759, 0.767) | 0.778 (0.774, 0.782) |

# Model

## Components

Before we are going to talk about our model architecture, we want to describe some layers and techniques that are used in our networks.

- Batch normalization
- Dropout
- Pooling
- Loss functions
- Adam optimization

# Model

## Batch Normalization

Batch Normalization [4] is a technique to accelerate deep network training by normalizing the activations throughout a neural network to take on a unit Gaussian distribution.

This reduces the covariance shift where the input distribution to each layer changes as the parameters of its previous layers are updated.
The normalization is done with respect to a minibatch of data.

# Model

## Dropout

Dropout layer [5] is a technique to deal with overfitting in neural networks.

At training time, some neurons are randomly dropped, along with their connections, during every forward pass.

Given a n-dimensional input vector $(X_1, \ldots, X_n)$ , the output from the dropout layer is

The d-dimensional vector $(Y_1, \ldots, Y_n)$ given by:

$$y_k = \mathrm{Bern}(p) x_k$$

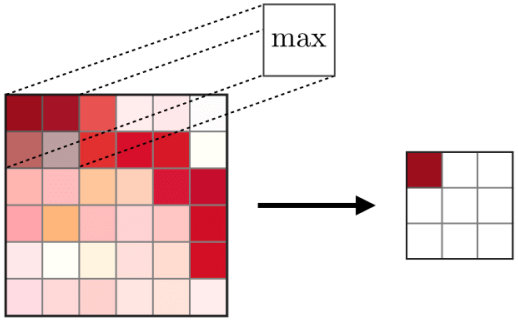
Where $p \in [0, 1]$ is the dropout parameter.

# Model

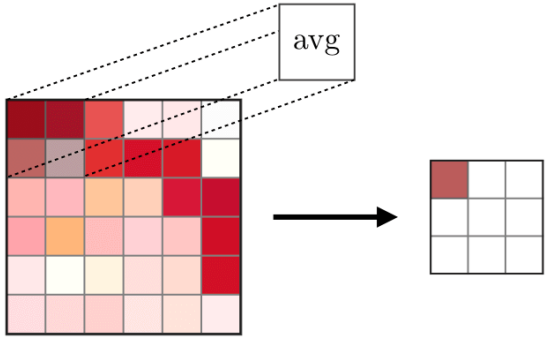
## Pooling

The pooling layer is a downsampling operation, typically applied after a convolution layer, which does some spatial invariance. In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively.

Each pooling operation selects the maximum value of the current view




Each pooling operation averages the values of the current view



# Model

## Loss Functions - Binary Cross Entropy Loss

Our approach to calculate the loss is to discretize our output into d-bins, and make our networks predict d-values, interpreting each output as the probability that the true value lies in the d-th bin.

We compare this with the true probability distribution, and measure the deviation as :

$$L = \frac{1}{N} \sum_{i=1}^{n} \left( - \sum_{j=1}^{d} y_d \log(\hat{y}_d) \right)$$

# Model

## Adam Optimization

We used Adam optimizer [6] as a default for our experiments.
Adam is a stochastic gradient descent optimization algorithm which works very well.
Concretely, the update is done using the gradient and the learning rate.

The Adam optimizer parameters in our model were applied as follows :
learning rate = 0.0001, beta_1=0.9, beta_2=0.999.

In addition, Adam is computationally efficient, has lower memory requirement, and favourable for problems with large data and parameters.

# Model

## DenseNet - Background

Convolutional Neural Networks caused a problem of decreasing the feature-map when passing through many layers, in order to solve this problem, at year 2017, Gao Huang published the article about DenseNet - Densely Connected Convolutional Network [1].

DenseNet solve this problem in a way that each layer in a dense block receives feature maps from all the preceding layers, and passes its output to all next layers.

# Model

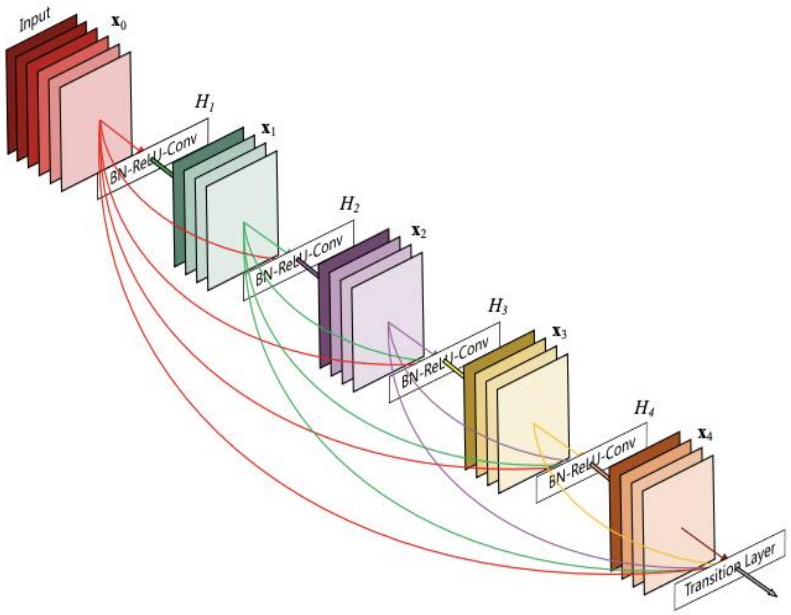
## DenseNet - Dense Block

Each $H_\ell$ has defined operations:

BN - ReLU - Conv(1×1) - BN - ReLU - Conv(3×3).

1x1 convolution allows us to compress the data of the input volume into a smaller volume before performing the more expensive convolution. This way, we encourage the weights to find a more efficient representation of the data.

The design was found to be especially effective for DenseNet and improves computational efficiency.

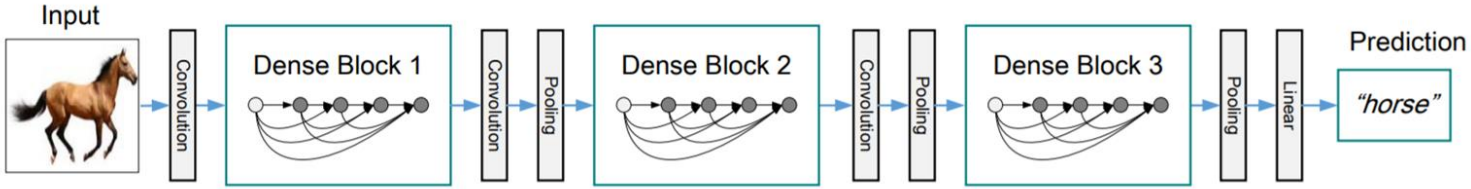


# Model

## DenseNet - Transition Layer

An important part of convolutional networks is pooling layers that change the size of feature-maps. To enable pooling in the model, it was divided into multiple densely connected dense blocks .

The layers between the dense blocks are called transition layers, which do 1x1 convolution and 2x2 average pooling with stride 2. Furthermore, to perform down-sampling in a DenseNet, it is inefficient to use expensive 3×3 convolution with stride 2.



# Model

## DenseNet - Example of use

For a given image X, we want to pass it through a convolutional network.

The DenseNet include $\ell$ layers, each of them implements a non-linear transformation $H_\ell$.

$H_\ell$ can be a compositions of operations such as l

function, pooling or convolution [1].

We symbolize the result (output) of the

$\ell$-th layer as $X_\ell$: $\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}])$

This introduces $\ell(\ell+1)/2$ connections in

an $\ell$-layer network.

# Model

## DenseNet - Dense connectivity

The network has different variations depending on the number of layers it has. In the case of our model we using  the 169-layer variation.

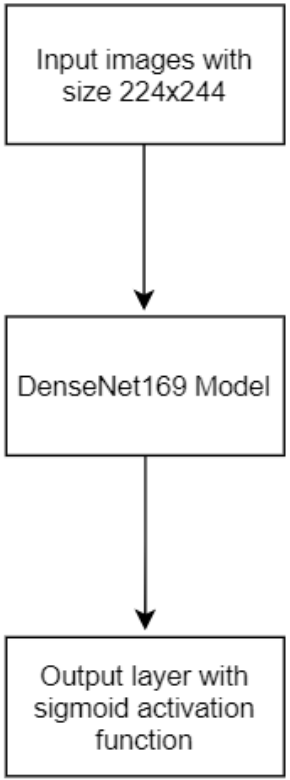
| Layers | Output Size | DenseNet-121 | DenseNet-169 | DenseNet-201 | DenseNet-265 |
|---|---|---|---|---|---|
| Convolution | $112 \times 112$ | $7 \times 7$ conv, stride 2 | | | |
| Pooling | $56 \times 56$ | $3 \times 3$ max pool, stride 2 | | | |
| Dense Block (1) | $56 \times 56$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$ |
| Transition Layer (1) | $56 \times 56$ | $1 \times 1$ conv | | | |
| | $28 \times 28$ | $2 \times 2$ average pool, stride 2 | | | |
| Dense Block (2) | $28 \times 28$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$ |
| Transition Layer (2) | $28 \times 28$ | $1 \times 1$ conv | | | |
| | $14 \times 14$ | $2 \times 2$ average pool, stride 2 | | | |
| Dense Block (3) | $14 \times 14$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$ |
| Transition Layer (3) | $14 \times 14$ | $1 \times 1$ conv | | | |
| | $7 \times 7$ | $2 \times 2$ average pool, stride 2 | | | |
| Dense Block (4) | $7 \times 7$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$ |
| Classification Layer | $1 \times 1$ | $7 \times 7$ global average pool | | | |
| | | 1000D fully-connected, softmax | | | |

# Model

## Base DenseNet169 Model

The base DenseNet169 model trained on ImageNet dataset, takes as an input images with size of 224x224x3, using 2x2 average pooling.
We changed the original DenseNet169 output layer to dense layer with 1 neuron that uses Sigmoid activation function, to predict binary result (Normal - 0 / Abnormal - 1).

Both the training and validation set uses batch size of 8, images were scaled to 224x224,  Adam optimizer with initial learning rate of 0.0001, 10 epochs.
The output layer, based on the sigmoid function, converts the input value to a value between 0 and 1 - probability.



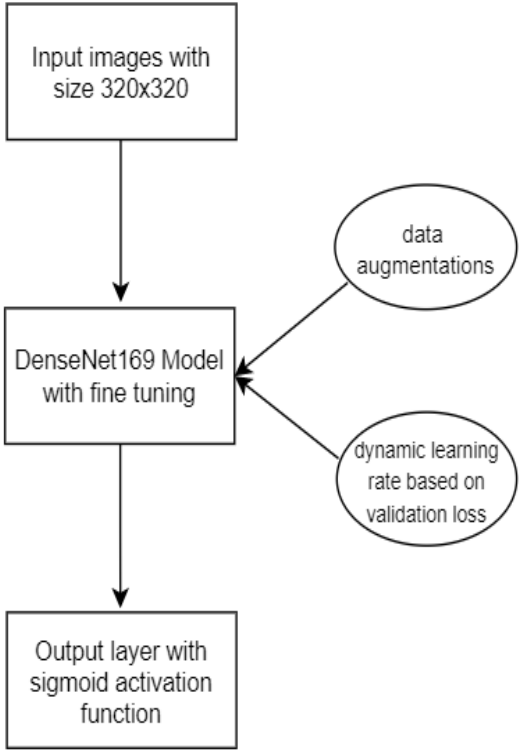
# Model

## Modified DenseNet169 Model

We modified the input layer, which takes as an input images with size of 320x320x3, and applied images data augmentation.

Same as the original DenseNet169 model, we changed the output layer to handle our study.

Training was the same as the original model, but the scaled and modified images which gives us higher feature-map.
We used Adam optimizer, but we implemented a model callback that reduce the learning rate when it recognise plateau on the validation loss metric.



# Model

## Model improvements and Fine-Tuning

The first improvement we applied to our model is the data augmentations.
As part of the data augmentations, we resize the images to 320x320 instead of 224x224 in the base DenseNet model. In addition, we applied another 2 image modifications:

- Image horizontal flip so the dataset can have a symmetric image of the study. This will enforce the model take into account at training both sides of the observation.
- Image random rotation of 30 degrees so that the model can take into account the small variations that could be present in normal radiographic studies.

We used those techniques to avoid over-fitting while training our network, and also, using this allow the model to handle some human differences.

# Model

## Model improvements and Fine-Tuning

Fine-tuning leverages a previously trained convolutional layer (in our case DenseNet model trained on ImageNet dataset), and makes such layers part of the learning process as well, by updating the model weights.

As part of the model Fine-Tuning, we used a dynamic learning rate, that reduces the learning rate each time it recognizes a Plateau of the validation set loss metric.

In other words, it means that we reduce the learning rate when a metric (validation loss) has stopped improving.

We implemented these using CallBacks that help to modify the model while its running, between epochs, using TensorFlow ReduceLROnPlateau (tf.keras.callbacks.ReduceLROnPlateau).

# Model

## Model improvements and Fine-Tuning

Fine-tuning implementation in our model:

```
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=1, verbose=1)
```

Reduce_lr is our callback to adjust the LR when training the network, it find the minimum validation loss value between the epochs, and reduce the learning rate by factor of 10 if it recognizes that the validation loss is increasing in each epoch ( New_lr = lr * 0.1 ).

Configure the LR callback in the model with initialized Learning rate of 0.0001:

```
Opt = tf.keras.optimizers.Adam(learning_rate=0.0001, beta_1=0.9, beta_2=0.999)
modelDenseNet169.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'],
callbacks=[reduce_lr])
```

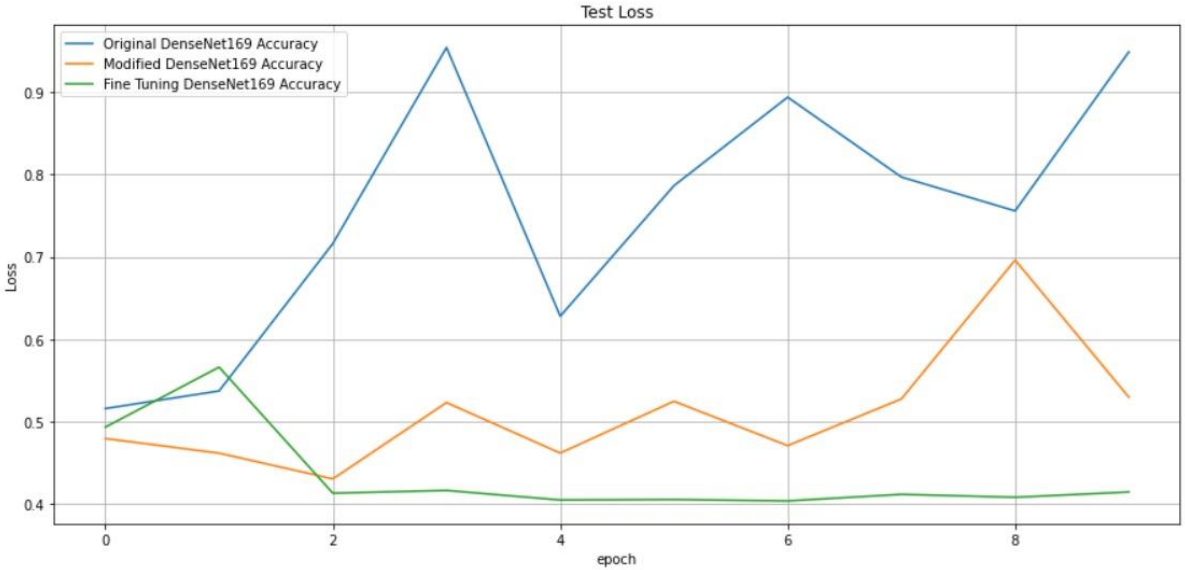
# Results & Analysis

## Loss

As we can see, the base model loss value (blue) is the most unstable in the graph, with the highest loss compared to the other model.

Our modified (yellow) and Fine-Tuned (green) models show more stable results, and overall lower loss.

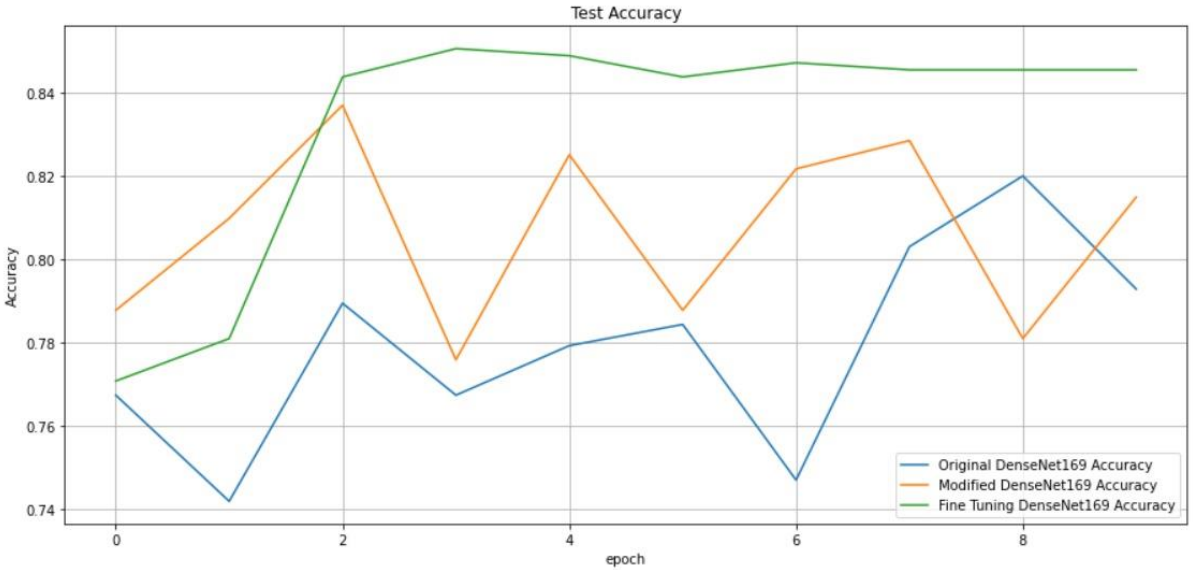
But the most interesting result here is the Fine-Tuned model, which have the lowest loss, with the highest stability and how it converges within the first



# Results & Analysis

## Accuracy

As we can see, the base model Accuracy value (blue) is the lowest in the graph, the modified (yellow) has a little bit better results in the beginning, but it's possible to see that both models Gets to almost the same Accuracy at the last epochs. The Fine-Tuned (green) model shows the best results, with much higher Accuracy compared to 2 other models. Also, the Fine-Tuned model accuracy have the highest stability and it converges within the first epochs.



# Conclusions

## The Conclusions

- In this project, we studied the effect of using techniques to well-known models to detect musculoskeletal abnormalities using the MURA dataset. In particular, we proposed the use of fine-tuning technique on top of a base model.

- The baseline network proved to achieve high accuracy results, but we noticed that the trained model using the Tine-Tuning approach had a better performance.

# Conclusions

## The Conclusions

- Thought we get a small improvement of the results, when taking into consideration that we are working on a medical research, every percent is matter, as this could be a life changer in terms of medical service.

- These models for diagnostic with computer may not currently achieve the accuracy needed to replace human interpretation of radiographic studies, but they are absolutely could save some time, reduce the human error and increase the medical team efficiency.

# References


- [1] - Densely Connected Convolutional Networks (2017)
- [2] - MURA: Large Dataset for Abnormality Detection in Musculoskeletal Radiographs (2018)
- [3] - Abnormality Detection in Musculoskeletal Radiographs with Convolutional Neural Networks and Performance Optimization (2019)
- [4] - Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift (2015)
- [5] - Dropout: A Simple Way to Prevent Neural Networks from Overfitting (2014)
- [6] - ADAM: A method for stochastic optimization (2017)