Deep Learning Project 2

Abnormality Detection in bone X-Rays

Doudos Panagiotis (f3352106)

Legkas Sotirios (f3352112)

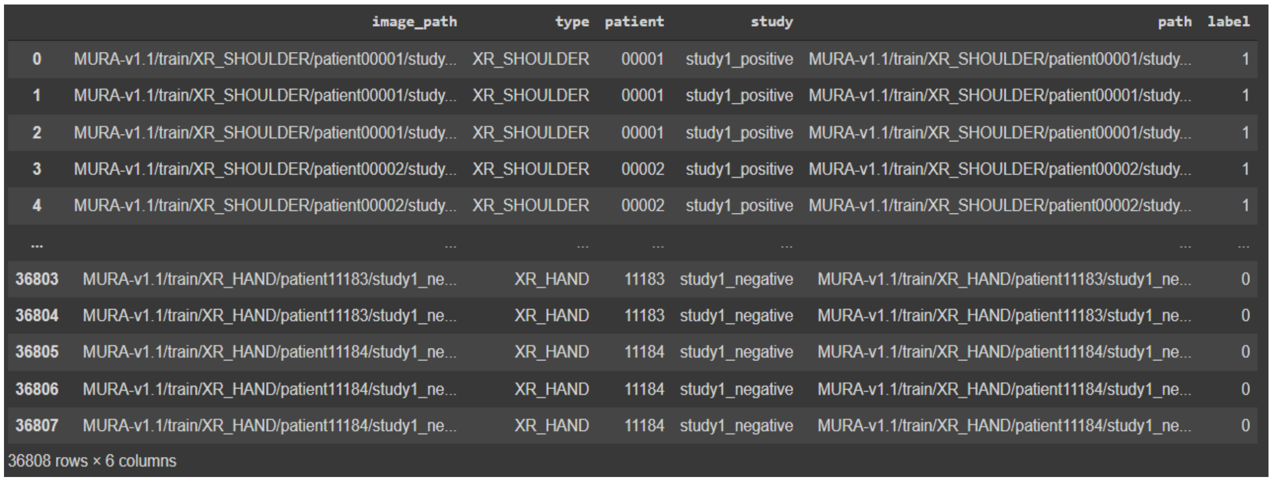
Introduction:

This Project's task was to tackle the following problem: "Given a study containing X-Ray images, build a deep learning model that decides if the study is normal or abnormal." The models had to be at least one CNN and at least one large pretrained model, to compare, contrast and understand the uses and advantages of each. The main evaluation metric is Cohen's Kappa, or how much the models agree with the human experts disregarding chance.

Data Preprocessing and Augmentation:

Regarding the data preprocessing, a DataFrame was created, containing 3 columns. One included the paths to each image, one the body part in the X-ray, in one of the 7 available categories (shoulder, hand, finger etc) and finally, one containing a second path that will be later used in merging the DataFrame with the label column from a separate file. The same process was followed to create the DataFrame for the test set, named validation set by the Project description. Both datasets were shuffled, making the data order random.

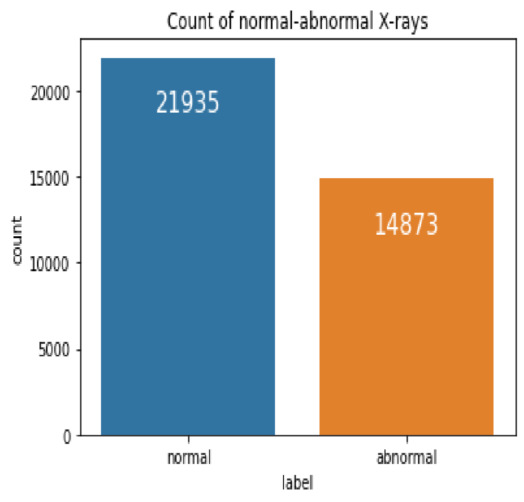
A snapshot of the DataFrame is shown below:



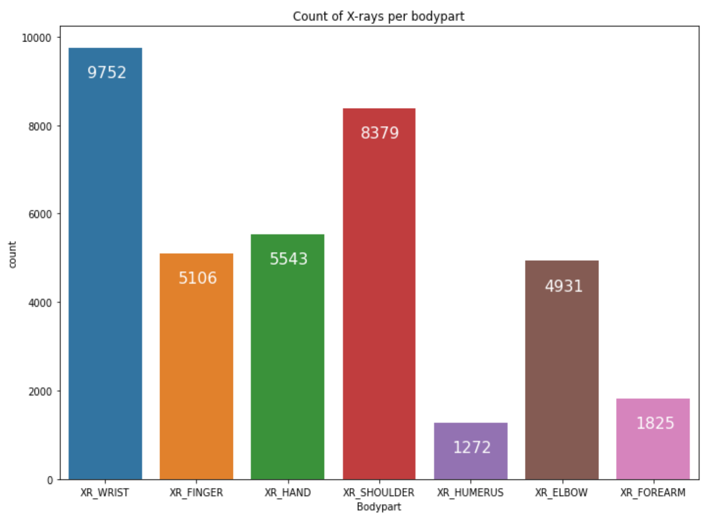
	image_path	type	patient	study	path	label
0	MURA-v1.1/train/XR_SHOULDER/patient00001/study...	XR_SHOULDER	00001	study1_positive	MURA-v1.1/train/XR_SHOULDER/patient00001/study...	1
1	MURA-v1.1/train/XR_SHOULDER/patient00001/study...	XR_SHOULDER	00001	study1_positive	MURA-v1.1/train/XR_SHOULDER/patient00001/study...	1
2	MURA-v1.1/train/XR_SHOULDER/patient00001/study...	XR_SHOULDER	00001	study1_positive	MURA-v1.1/train/XR_SHOULDER/patient00001/study...	1
3	MURA-v1.1/train/XR_SHOULDER/patient00002/study...	XR_SHOULDER	00002	study1_positive	MURA-v1.1/train/XR_SHOULDER/patient00002/study...	1
4	MURA-v1.1/train/XR_SHOULDER/patient00002/study...	XR_SHOULDER	00002	study1_positive	MURA-v1.1/train/XR_SHOULDER/patient00002/study...	1
...	...	...	...	...	...	...
36803	MURA-v1.1/train/XR_HAND/patient11183/study1_ne...	XR_HAND	11183	study1_negative	MURA-v1.1/train/XR_HAND/patient11183/study1_ne...	0
36804	MURA-v1.1/train/XR_HAND/patient11183/study1_ne...	XR_HAND	11183	study1_negative	MURA-v1.1/train/XR_HAND/patient11183/study1_ne...	0
36805	MURA-v1.1/train/XR_HAND/patient11184/study1_ne...	XR_HAND	11184	study1_negative	MURA-v1.1/train/XR_HAND/patient11184/study1_ne...	0
36806	MURA-v1.1/train/XR_HAND/patient11184/study1_ne...	XR_HAND	11184	study1_negative	MURA-v1.1/train/XR_HAND/patient11184/study1_ne...	0
36807	MURA-v1.1/train/XR_HAND/patient11184/study1_ne...	XR_HAND	11184	study1_negative	MURA-v1.1/train/XR_HAND/patient11184/study1_ne...	0

36808 rows × 6 columns

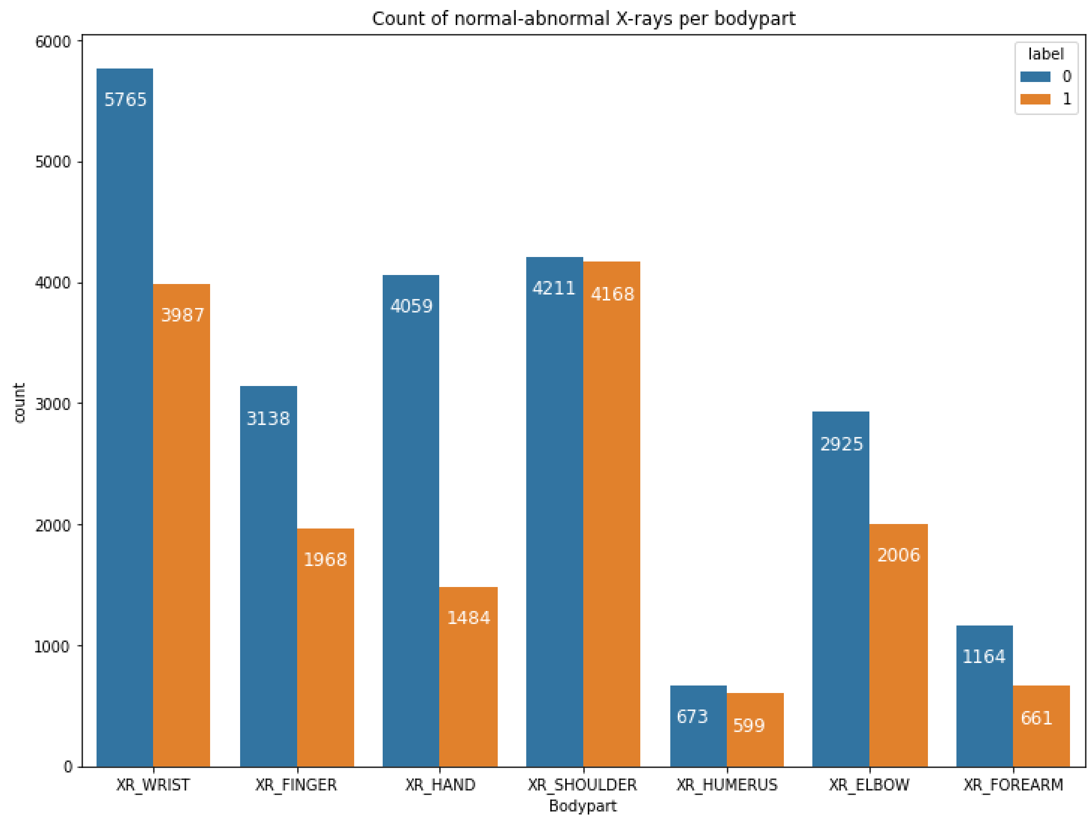
The number of normal and abnormal X-rays:



The number of X-rays per body-part:



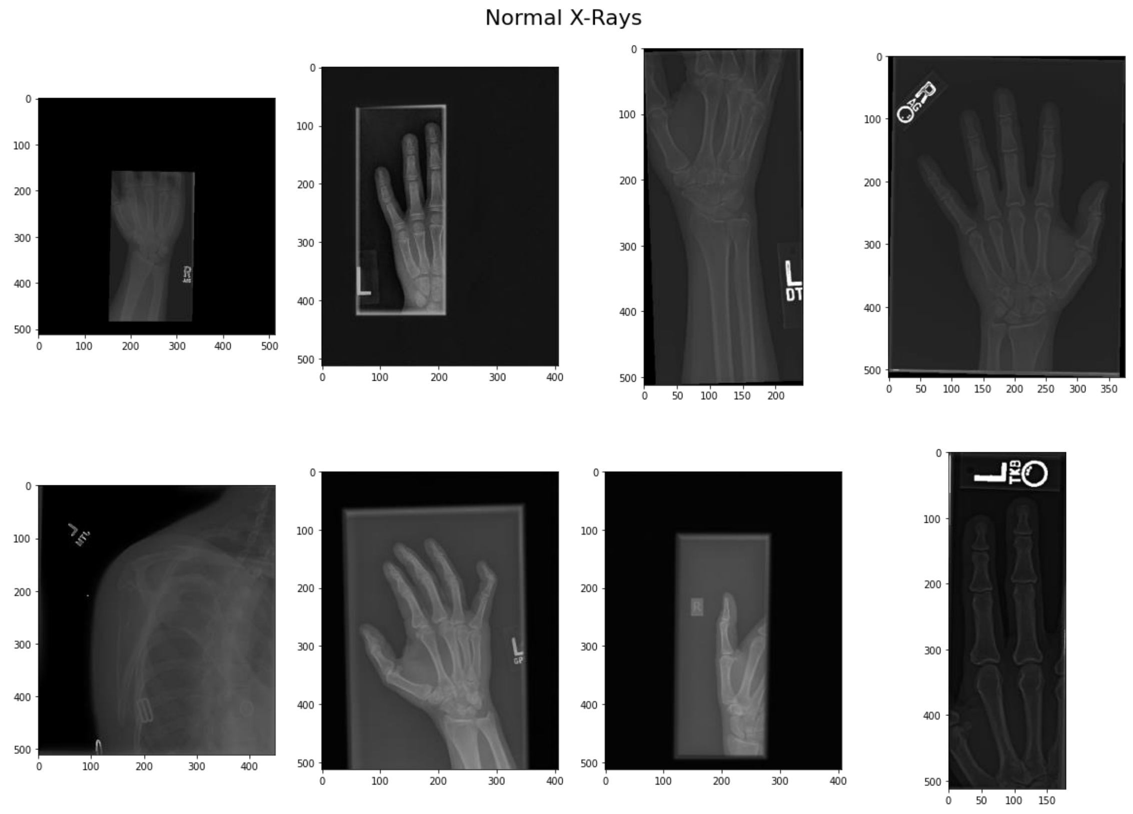
The number of normal and abnormal X-rays per bodypart:



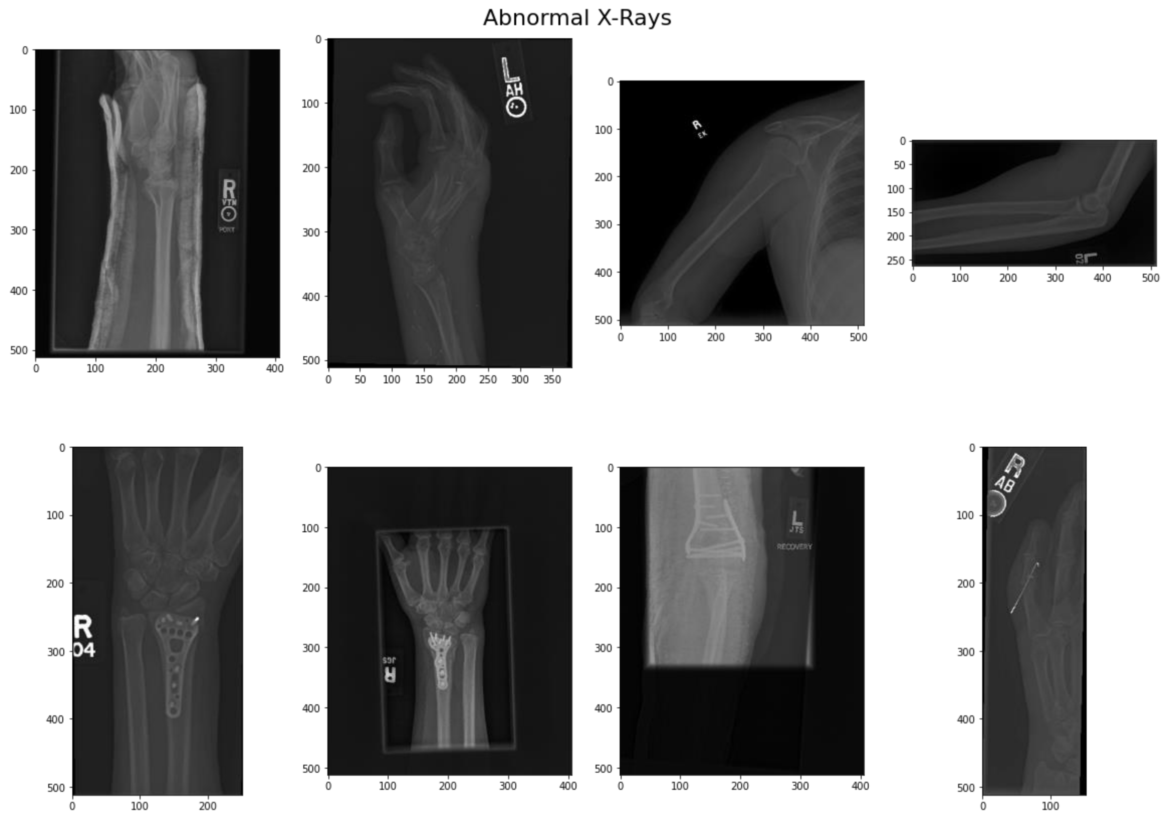
The next step in data preprocessing was to use a function utilizing ImageDataGenerator from Keras to create batches of tensor image data with real time data augmentation. The parameters of this function were defined, so that the images would be augmented by a rotation of up to 30 degrees, a possible horizontal flip, filling the outside boundaries with a constant value so that the useful image content is not altered and finally rescaling the images by dividing each pixel value with 255. This function also included a validation split parameter, which was set to 0.2, meaning that 20% of the designated “train set” will be used to validate our models during training. The same function was essentially implemented for the test data but without any augmentation apart from image rescaling.

Next, the aforementioned function was used along with “flow_to_dataframe “, which takes the DataFrame and the directory paths of the images created and generates batches with the augmented/normalized data that can be used in the models. The batch size was set to 32 and the image size to (224x224). The same procedure was applied to the test data as well.

Some normal X-rays are shown below:

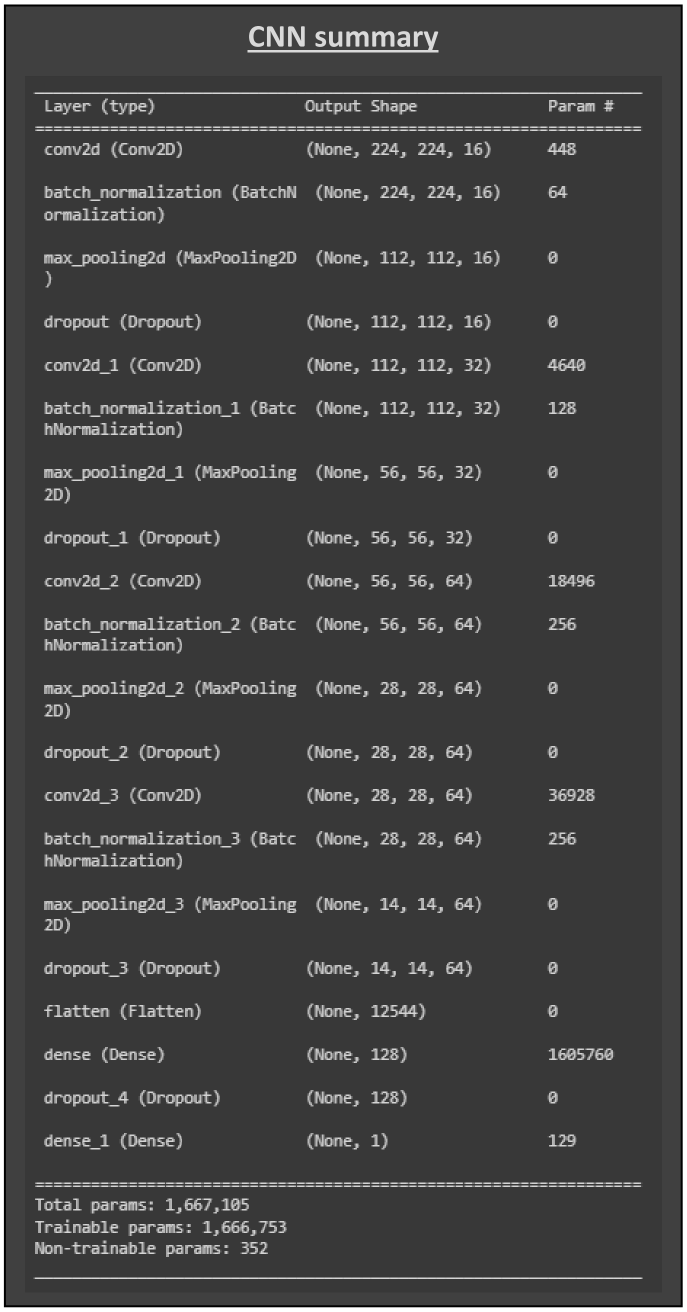


Some abnormal X-rays are shown below:



CNN:

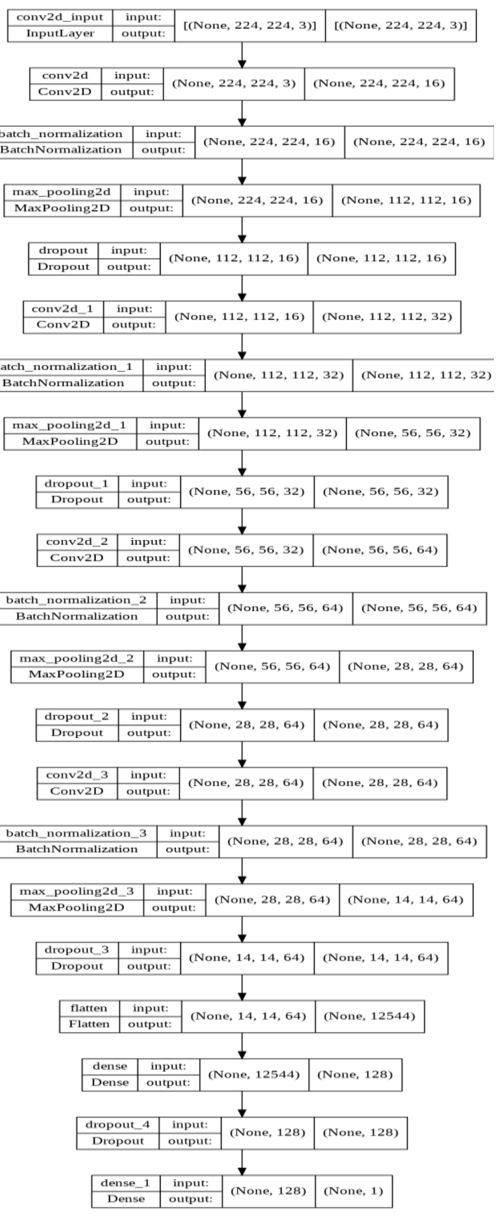
First, a CNN was created. Unlike the previous Project, this time the focus was not on fine-tuning details, but rather on efficiently making a model that works relatively well, to save time and resources for the bigger models. Therefore, the tuning was mostly manual, as in trying different values when the kappa returned was not satisfactory. The CNN was built starting with the input layer, accepting 224 by 224 images, followed by 4 convolutional layers, each with batch normalization, max pooling and dropout -with a rate of 0.25-. The filters used were scaled to increase every layer, starting with 16, moving on to 32 and then having 64 for the last two layers. The kernel size was set to 3 by 3, with Relu as the activation function. After the CNN's convolutional layers, a dense layer was added, with 128 nodes, activated with Relu and followed by a 0.25 dropout once again. Finally, a single node sigmoid output layer gives the final result. On training the model, the loss function was set to binary cross entropy, as the problem is binary by nature. A fairly standard learning rate was selected at 0.0001 and the metrics observed were accuracy and the kappa score.



CNN summary

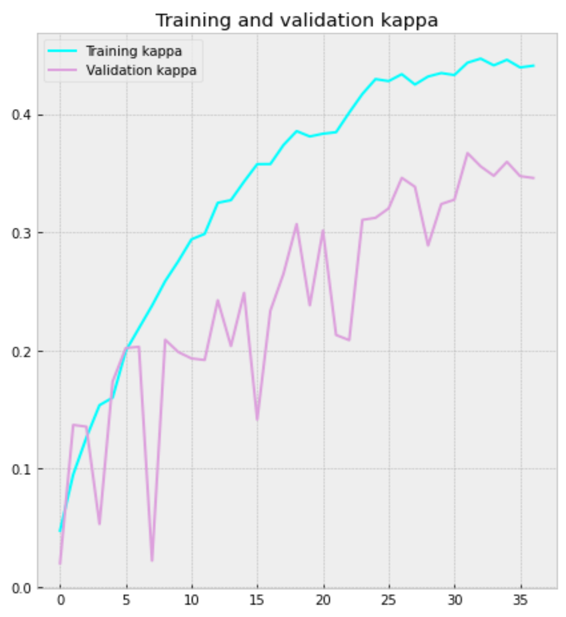
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 16)      448

 batch_normalization (BatchN  (None, 224, 224, 16)     64
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 112, 112, 16)     0
 )

 dropout (Dropout)           (None, 112, 112, 16)      0

 conv2d_1 (Conv2D)           (None, 112, 112, 32)      4640

 batch_normalization_1 (Batc  (None, 112, 112, 32)     128
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 56, 56, 32)        0

 conv2d_2 (Conv2D)           (None, 56, 56, 64)        18496

 batch_normalization_2 (Batc  (None, 56, 56, 64)       256
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 28, 28, 64)       0
 2D)

 dropout_2 (Dropout)         (None, 28, 28, 64)        0

 conv2d_3 (Conv2D)           (None, 28, 28, 64)        36928

 batch_normalization_3 (Batc  (None, 28, 28, 64)       256
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 dropout_3 (Dropout)         (None, 14, 14, 64)        0

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 128)               1605760

 dropout_4 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 1,667,105
Trainable params: 1,666,753
Non-trainable params: 352
_________________________________________________________________
```

CNN Architecture

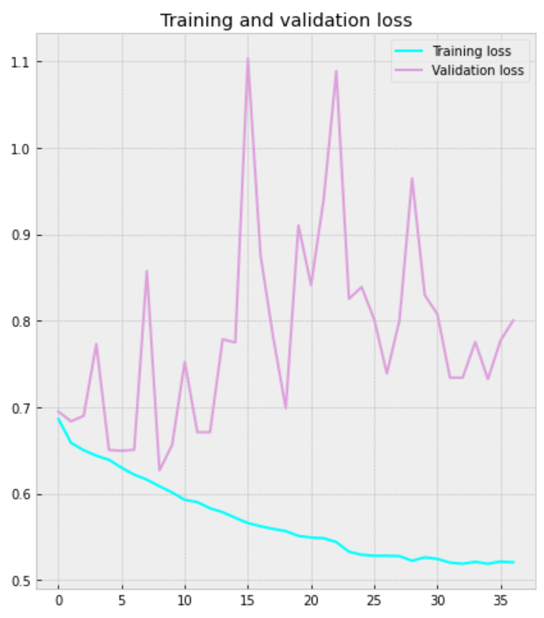

conv2d_input | InputLayer | input: [(None, 224, 224, 3)] | output: [(None, 224, 224, 3)]
conv2d | Conv2D | input: (None, 224, 224, 3) | output: (None, 224, 224, 16)
batch_normalization | BatchNormalization | input: (None, 224, 224, 16) | output: (None, 224, 224, 16)
max_pooling2d | MaxPooling2D | input: (None, 224, 224, 16) | output: (None, 112, 112, 16)
dropout | Dropout | input: (None, 112, 112, 16) | output: (None, 112, 112, 16)
conv2d_1 | Conv2D | input: (None, 112, 112, 16) | output: (None, 112, 112, 32)
batch_normalization_1 | BatchNormalization | input: (None, 112, 112, 32) | output: (None, 112, 112, 32)
max_pooling2d_1 | MaxPooling2D | input: (None, 112, 112, 32) | output: (None, 56, 56, 32)
dropout_1 | Dropout | input: (None, 56, 56, 32) | output: (None, 56, 56, 32)
conv2d_2 | Conv2D | input: (None, 56, 56, 32) | output: (None, 56, 56, 64)
batch_normalization_2 | BatchNormalization | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
max_pooling2d_2 | MaxPooling2D | input: (None, 56, 56, 64) | output: (None, 28, 28, 64)
dropout_2 | Dropout | input: (None, 28, 28, 64) | output: (None, 28, 28, 64)
conv2d_3 | Conv2D | input: (None, 28, 28, 64) | output: (None, 28, 28, 64)
batch_normalization_3 | BatchNormalization | input: (None, 28, 28, 64) | output: (None, 28, 28, 64)
max_pooling2d_3 | MaxPooling2D | input: (None, 28, 28, 64) | output: (None, 14, 14, 64)
dropout_3 | Dropout | input: (None, 14, 14, 64) | output: (None, 14, 14, 64)
flatten | Flatten | input: (None, 14, 14, 64) | output: (None, 12544)
dense | Dense | input: (None, 12544) | output: (None, 128)
dropout_4 | Dropout | input: (None, 128) | output: (None, 128)
dense_1 | Dense | input: (None, 128) | output: (None, 1)

The above model had a total of 1,666,753 trainable parameters. The training was performed with early stopping – with 5 epochs of patience- and learning rate reduction on plateau until 10^(-6), both monitoring the validation set's Cohens kappa, with 4 epochs of patience. 8 workers were set for a maximum of 50 epochs. The results of the CNN are shown extensively below:

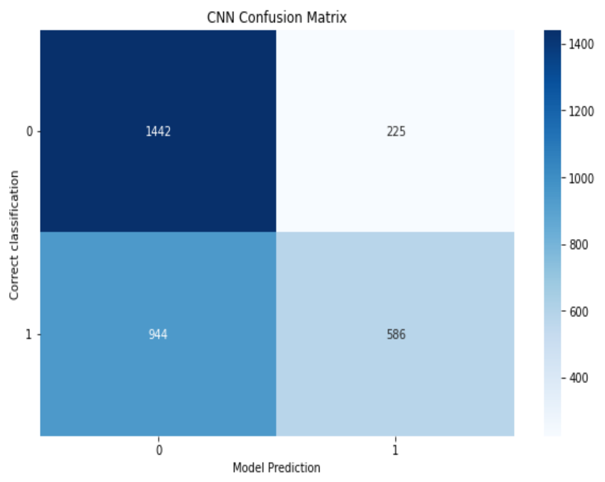
Training and validation Cohen's Kappa



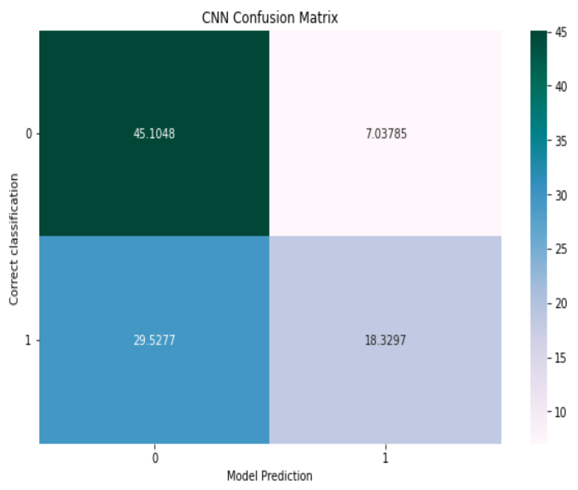
Training and validation Loss

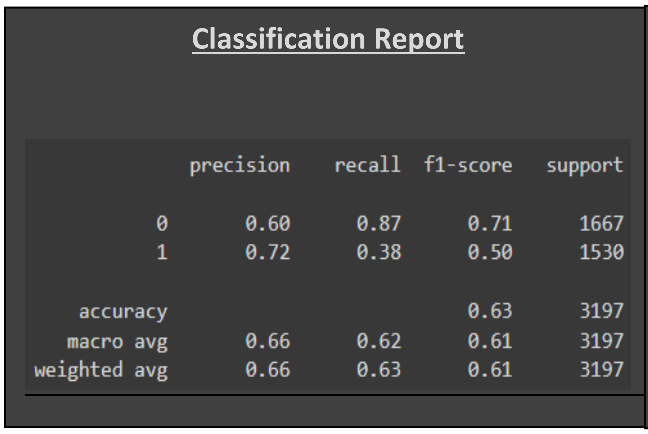


CNN Confusion Matrix

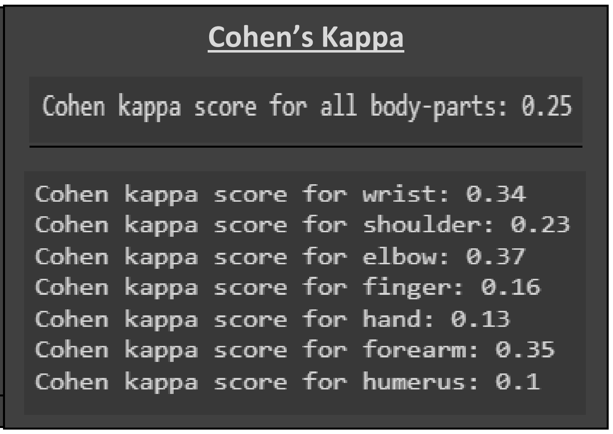


CNN Confusion Matrix (%)




Classification Report

```
              precision    recall  f1-score   support

           0       0.60      0.87      0.71      1667
           1       0.72      0.38      0.50      1530

    accuracy                           0.63      3197
   macro avg       0.66      0.62      0.61      3197
weighted avg       0.66      0.63      0.61      3197
```


Cohen's Kappa

```
Cohen kappa score for all body-parts: 0.25
```

```
Cohen kappa score for wrist: 0.34
Cohen kappa score for shoulder: 0.23
Cohen kappa score for elbow: 0.37
Cohen kappa score for finger: 0.16
Cohen kappa score for hand: 0.13
Cohen kappa score for forearm: 0.35
Cohen kappa score for humerus: 0.1
```

The Cohen's kappa given by the CNN is not as satisfactory as expected. Maybe the pretrained models can raise that bar. An interesting thing however was the shape of the learning curves. The validation set's metrics had very abrupt hills and valleys as epochs went by. This should be a result of the different types of images, which are misbalanced in their labels and as such, it takes a lot of trial and error for the model to understand them depending on what it receives each batch. The above hypothesis can also be clearly seen on the vastly different kappa scores for each body part.

Pretrained models: Introduction

To investigate the effect of pretraining and how training the layers affects the models, the pretrained models were run in 2 different states regarding the freezing of their layers. Once with only the added layers on top being trainable and the actual model being frozen and once where the model's upper layers are trained first with the pretrained layers being frozen, and then, with a smaller learning rate, the whole model is retrained, this time being fully trainable. Due to the computationally demanding nature of the pretrained models, not many epochs were used. The minimal number of epochs however is also important in keeping the pretrained layers relatively intact and altering them only as necessary to keep their innate advantages.

Some extra layers are added to the pretrained models. Before the model, an input layer is added to receive the image matrices. Then, on top of each pretrained model, after removing its top, 6 layers are added. First a dropout layer with 0.2 rate, then a dense layer with 1024 nodes, activated by Relu, followed by a batch normalization layer. Another dropout layer is enforced, this time with 0.3 dropout rate and finally the output layer with a sigmoid activation function since the classification is binary.

For the parameters inside the model function, the following were applied. First of all, since the task is different than the original, include_top was set equal to False. Then the weights were set to "imagenet" due to the nature of the task and the necessity of the pretraining. Finally, the correct input shape was selected, the pooling was set equal to max, as we needed feature extraction and 2 classes were selected as our model is binary.

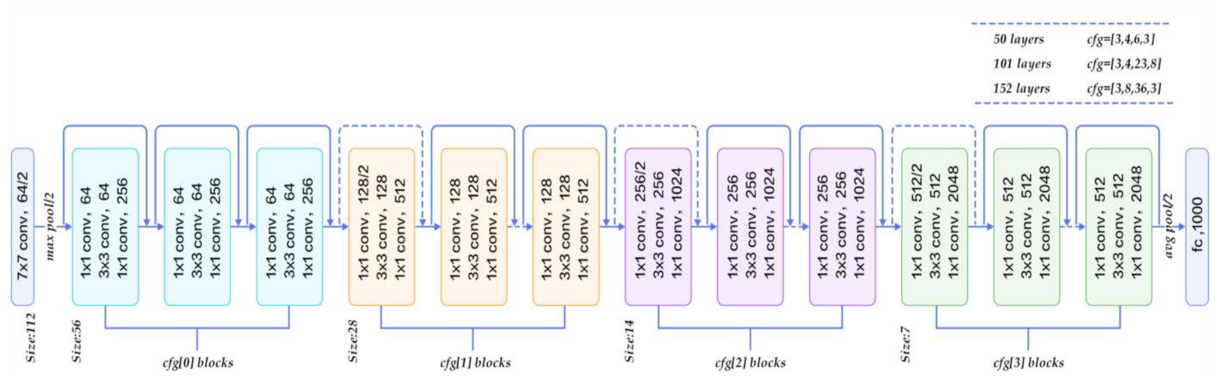
When only the upper layers are trained, for the fit, the optimizer selected was Adam, with a 0.0001 learning rate to relatively quickly train the upper layers, the loss function is Binary cross entropy and early stopping was applied with a patience of 5 epochs. The model was set to run for 12 epochs and the Cohen's Kappa of the validation set was the metric on which the model is assessed. Loss and Accuracy is also observed.

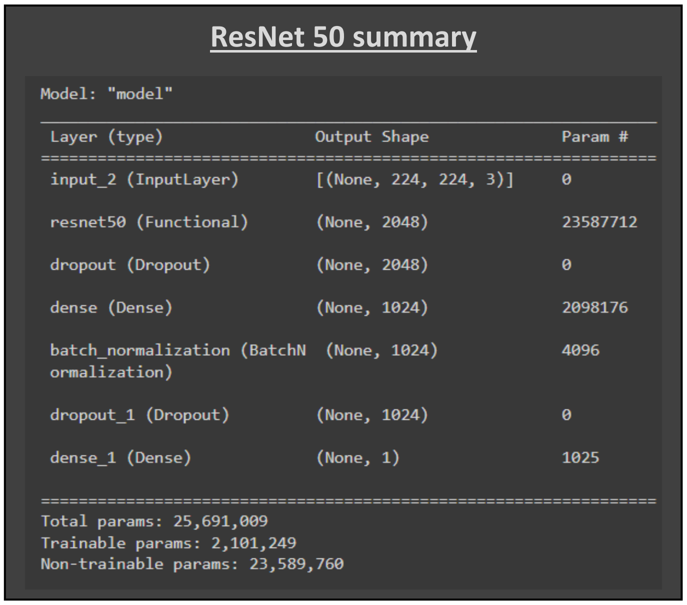
For the retraining of the models, this time with all layers being trainable, the learning rate was adjusted to 10^(-5), because a high learning rate applied to the whole of the pretrained model could be detrimental to the nature of the pretrained. The adjustment of learning rate when hitting plateau was also added, multiplying the learning rate by 0.1 when necessary (the patience was set to 8), with a minimum of 10^(-9). More epochs were allowed, this time having up to 15. The metrics used were retained.

Pretrained models: (1) ResNet50

The first pretrained model used was ResNet50. An improvement on ResNet, an overall trustworthy model, being one of the first to use skip connection. ResNet50 seemed like a good choice as it is one of the models proposed by Keras for image classification and is not too large of a model for our available resources. The training was performed as described above, which is how both of our pretrained models were handled. The results for both the frozen and trainable ResNet50 are shown below. Emphasis is given to the better model, that being the one trained twice.

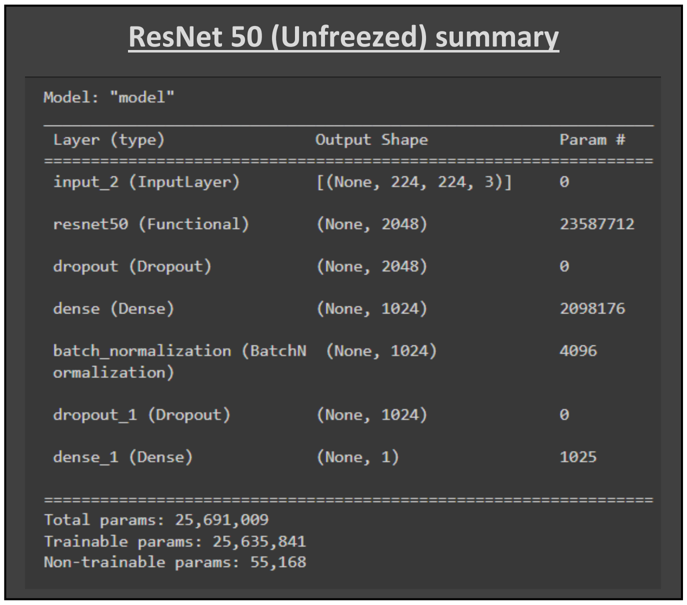
Resnet 50 Architecture





ResNet 50 summary

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 resnet50 (Functional)       (None, 2048)              23587712

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1024)              2098176

 batch_normalization (BatchN  (None, 1024)             4096
 ormalization)

 dropout_1 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 1)                 1025

=================================================================
Total params: 25,691,009
Trainable params: 2,101,249
Non-trainable params: 23,589,760
```

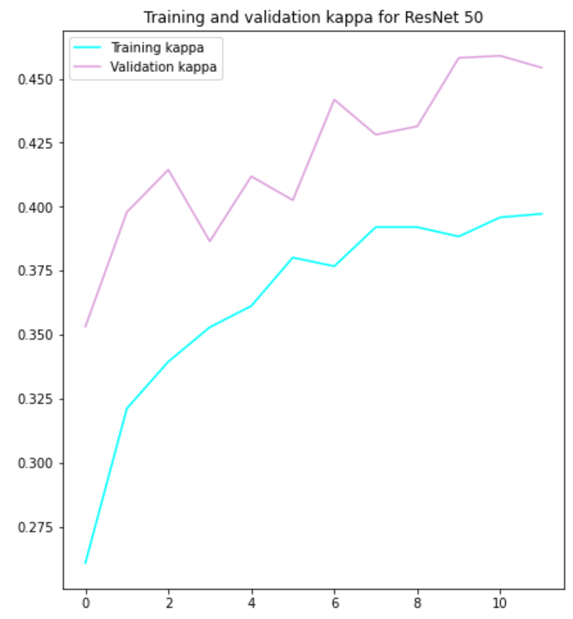


ResNet 50 (Unfreezed) summary

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 resnet50 (Functional)       (None, 2048)              23587712

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1024)              2098176

 batch_normalization (BatchN  (None, 1024)             4096
 ormalization)

 dropout_1 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 1)                 1025

=================================================================
Total params: 25,691,009
Trainable params: 25,635,841
Non-trainable params: 55,168
```

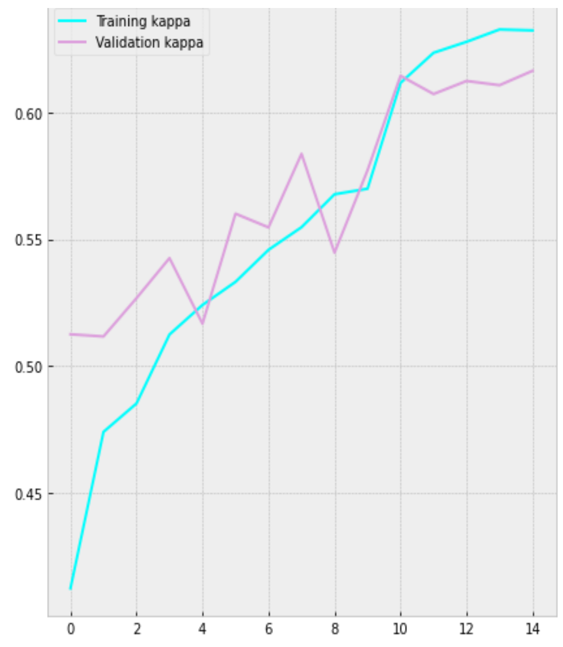
Resnet 50

Training and validation Cohen's Kappa



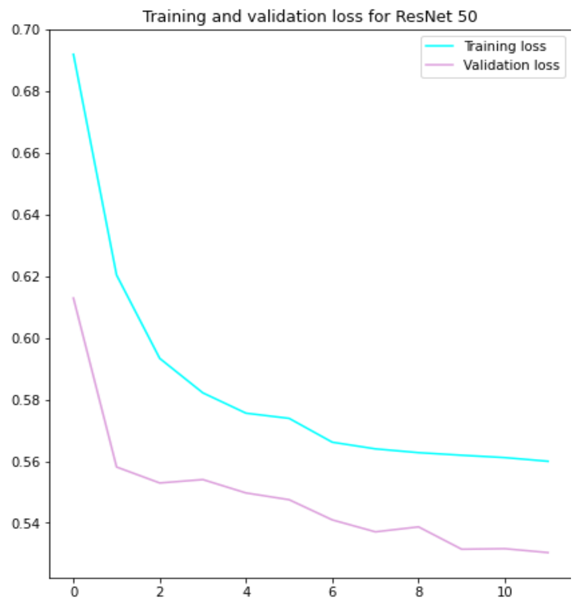
Resnet 50 (Unfreezed)

Training and validation Cohen's Kappa



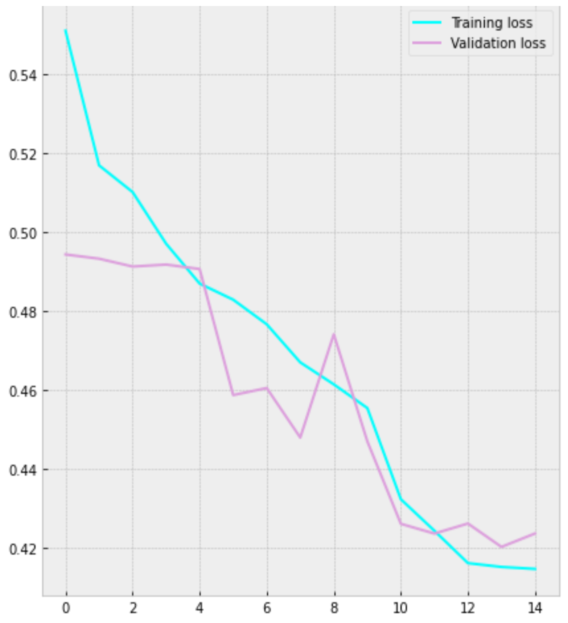
Resnet 50

Training and validation Loss



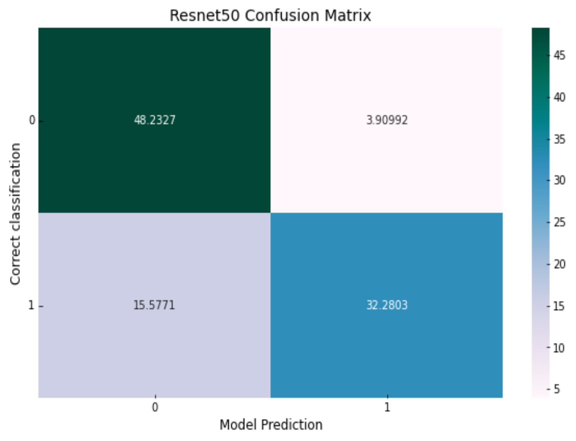
Resnet 50 (Unfreezed)

Training and validation Loss



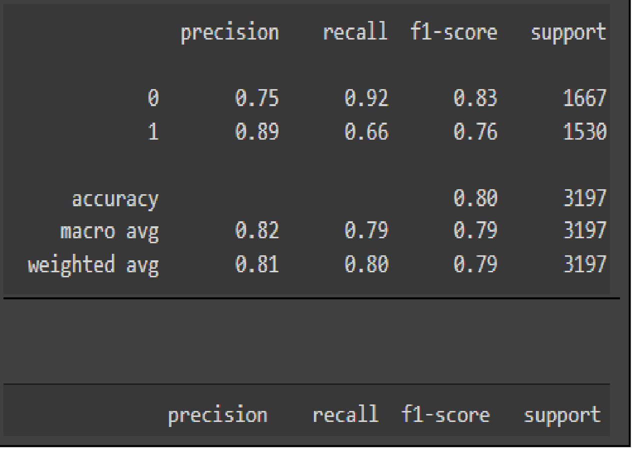
ResNet 50 (Unfreezed)

Confusion Matrix (%)

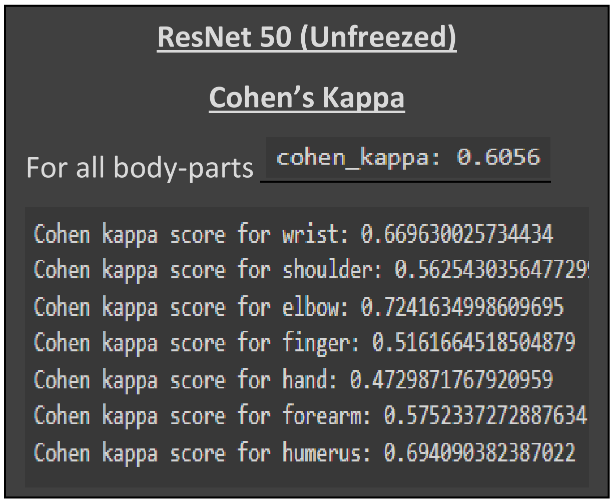


ResNet 50 (Unfreezed)

Classification Report



```
              precision    recall  f1-score   support

           0       0.75      0.92      0.83      1667
           1       0.89      0.66      0.76      1530

    accuracy                           0.80      3197
   macro avg       0.82      0.79      0.79      3197
weighted avg       0.81      0.80      0.79      3197

              precision    recall  f1-score   support
```



ResNet 50 (Unfreezed)

Cohen's Kappa

For all body-parts

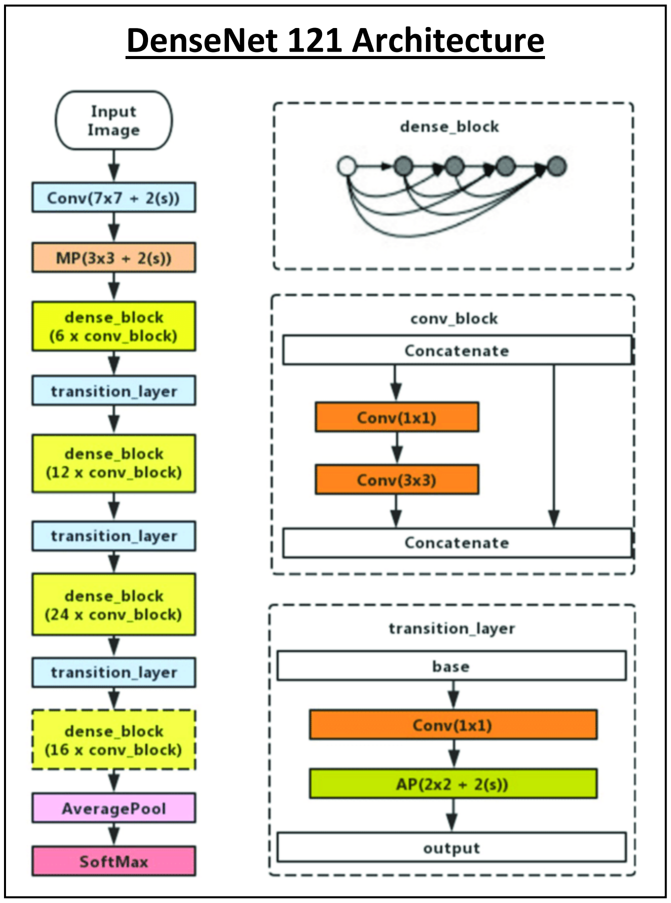
```
cohen_kappa: 0.6056
```

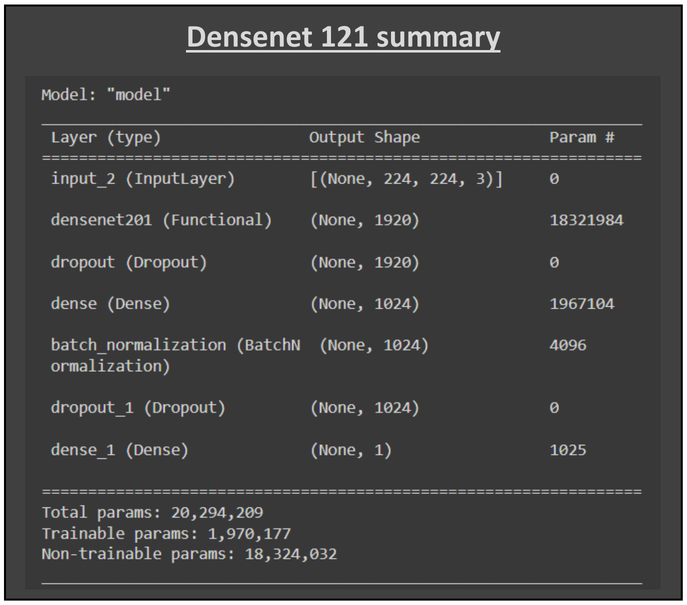
```
Cohen kappa score for wrist: 0.669630025734434
Cohen kappa score for shoulder: 0.56254303564772
Cohen kappa score for elbow: 0.7241634998609695
Cohen kappa score for finger: 0.5161664518504879
Cohen kappa score for hand: 0.4729871767920959
Cohen kappa score for forearm: 0.5752337272887634
Cohen kappa score for humerus: 0.694090382387022
```

When trained through freezing and unfreezing, the model vastly improved its Cohen's kappa from 0.45 to 0.58, showing how efficiently a correct training approach can focus and improve a pretrained models results to a given task. Evidently, the twice-trained ResNet50 based model did much better than the CNN as well, reaching a fairly satisfactory Cohen's Kappa of X and accuracy of Y in the test set. This was to be expected given it was a pretrained model. How does it fair against another pretrained model however?

Pretrained models: (2) DenseNet121

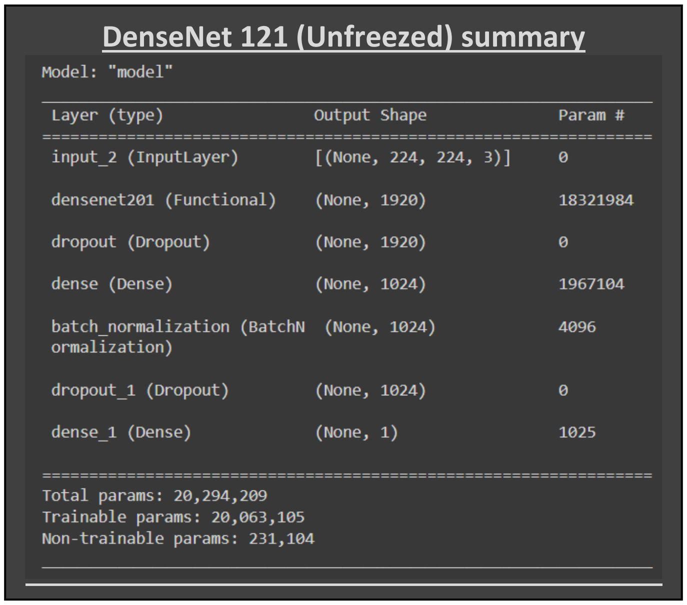
The second pretrained model selected was DenseNet121. The inspiration for this model came from its wide usage in lung X-Ray classification papers during CoViD-19. The model is also of a moderate size for the GPU capacity given. The DenseNet family of models seems to be one of the more popular in general for image classification problems, because of the way they use "dense" convolutional blocks to narrow down on image details and features. It is a larger model in size than ResNet50, leading to the hypothesis that, if correctly trained, it should be able to learn to distinguish the data more efficiently. The training pipeline was once again the same and the model's metrics are shown below:





Densenet 121 summary

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 densenet201 (Functional)    (None, 1920)              18321984

 dropout (Dropout)           (None, 1920)              0

 dense (Dense)               (None, 1024)              1967104

 batch_normalization (BatchN  (None, 1024)             4096
 ormalization)

 dropout_1 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 1)                 1025

=================================================================
Total params: 20,294,209
Trainable params: 1,970,177
Non-trainable params: 18,324,032
_________________________________________________________________
```

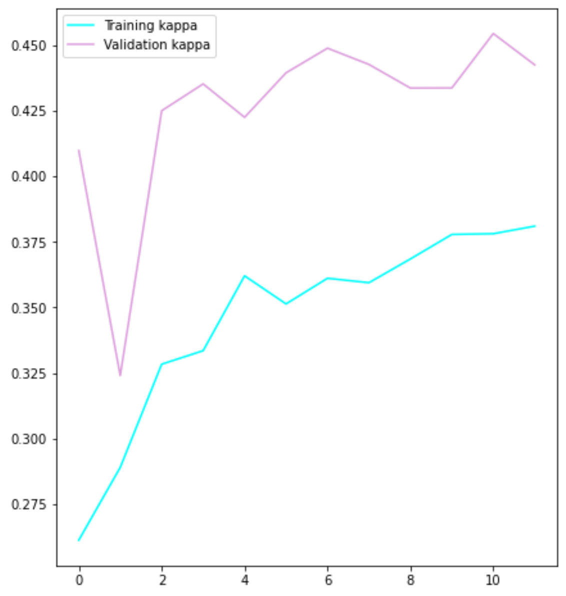


DenseNet 121 (Unfreezed) summary

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 densenet201 (Functional)    (None, 1920)              18321984

 dropout (Dropout)           (None, 1920)              0

 dense (Dense)               (None, 1024)              1967104

 batch_normalization (BatchN  (None, 1024)             4096
 ormalization)

 dropout_1 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 1)                 1025

=================================================================
Total params: 20,294,209
Trainable params: 20,063,105
Non-trainable params: 231,104
_________________________________________________________________
```

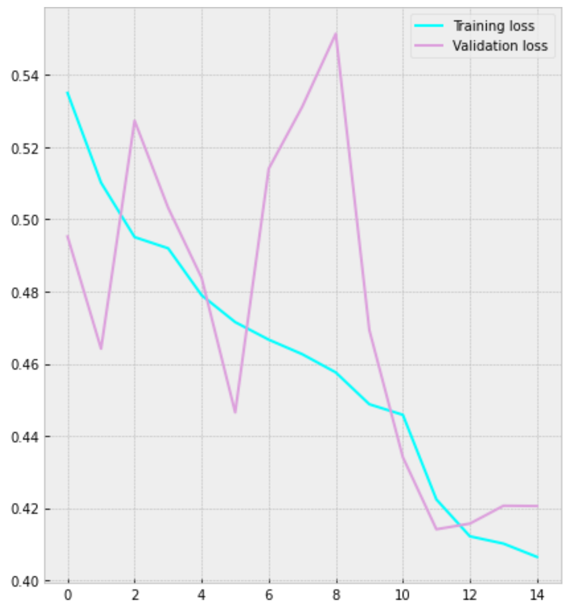
DenseNet 121

Training and validation Cohen's Kappa



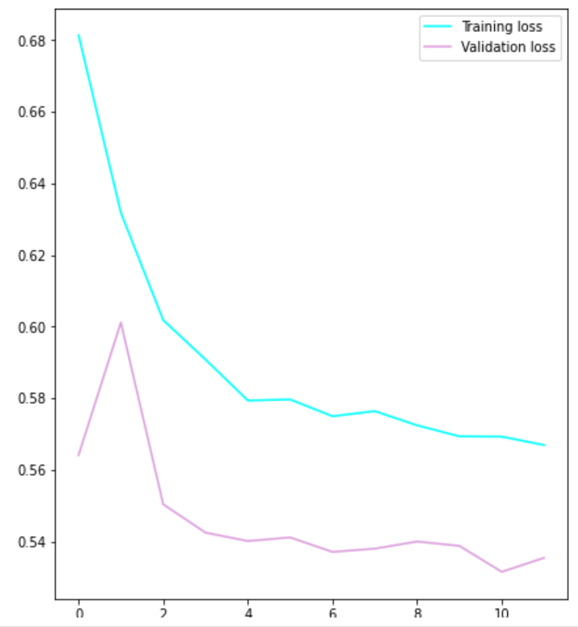
DenseNet 121 (Unfreezed)

Training and validation Cohen's Kappa



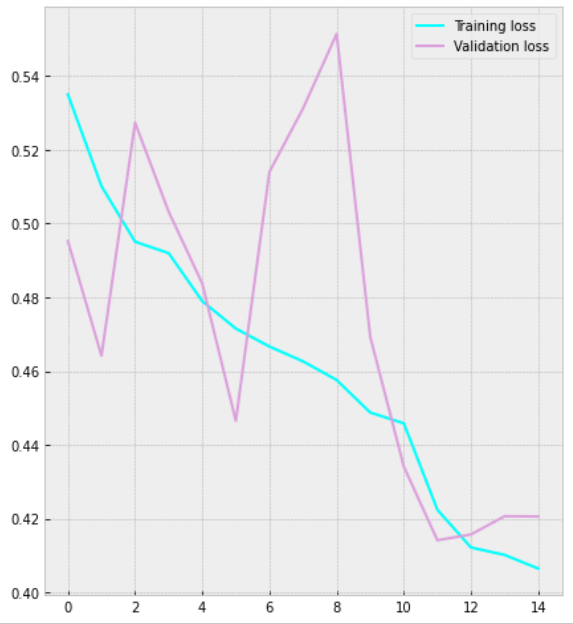
DenseNet 121

Training and validation Loss



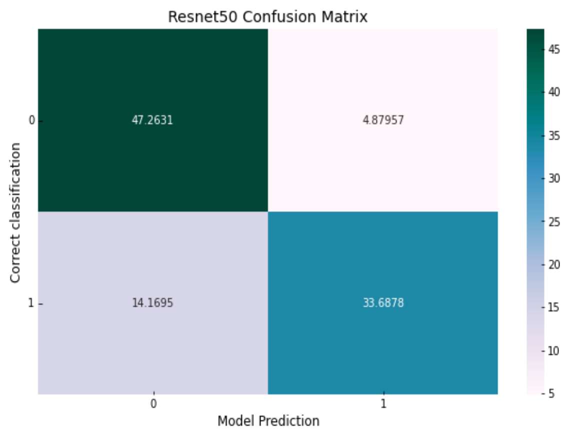
DenseNet 121 (Unfreezed)

Training and validation Loss



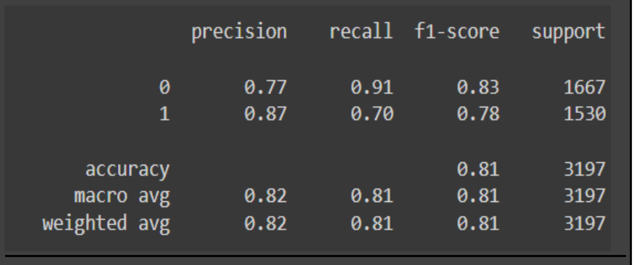
DenseNet 121 (Unfreezed)

Confusion Matrix (%)

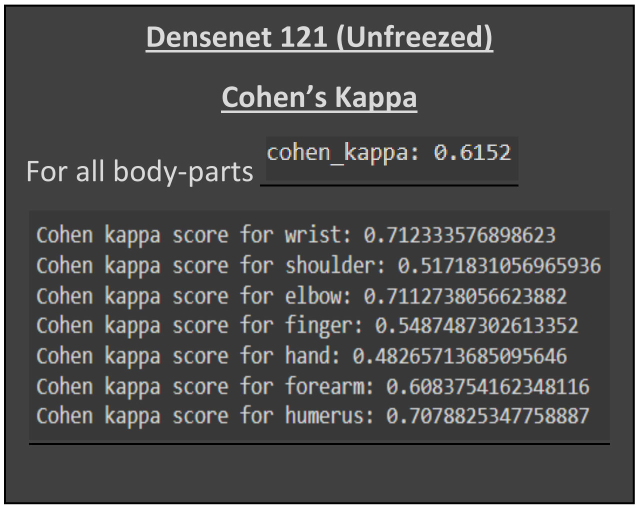


DenseNet 121 (Unfreezed)

Classification Report



```
              precision    recall  f1-score   support

           0       0.77      0.91      0.83      1667
           1       0.87      0.70      0.78      1530

    accuracy                           0.81      3197
   macro avg       0.82      0.81      0.81      3197
weighted avg       0.82      0.81      0.81      3197
```


Densenet 121 (Unfreezed)

Cohen's Kappa

For all body-parts
```
cohen_kappa: 0.6152
```
```
Cohen kappa score for wrist: 0.712333576898623
Cohen kappa score for shoulder: 0.5171831056965936
Cohen kappa score for elbow: 0.7112738056623882
Cohen kappa score for finger: 0.5487487302613352
Cohen kappa score for hand: 0.48265713685095646
Cohen kappa score for forearm: 0.6083754162348116
Cohen kappa score for humerus: 0.7078825347758887
```

An even better performing model seeing how it further improved ResNet50's Cohen's kappa score, and therefore the hypothesis made in its selection seems to be correct. It should be noted however, that when DenseNet121 is judging shoulder or elbow X-Rays specifically, it seems to be quite inferior to ResNet. Maybe we could combine them.

If-Model, a hypothetical combination of the two (or more):

The way the data are annotated, it can always be known which body part is being checked. If it is made sure that there is a human in the loop annotating the body parts, it could be possible to make an if-clause that always selects the better performing model on that specific body part. Continuing that thought, the models could even be trained on those body parts specifically and selected as such. This could be the way to some of the much higher Cohen kappas shown on the leaderboard.

Conclusion

This Project's main focus was on tackling a realistic deep learning task, using models that are currently still at use by several organizations and data scientists. With limited computational resources, the selection of which models to use and what to focus on was not only a necessary crash-land to reality from the academic ideal of infinite computational power, but also a chance to acquire a skill that can only be useful.

Google Colab Link:

The models were run from many different accounts on many different Colab notebooks throughout the preparation period. There is not a single notebook containing the ran version of all code, but all of the commands are included here:

https://colab.research.google.com/drive/1qQtxSZUD69069SJHRgvWRGh9CN4rONZ-?usp=sharing