Abnormality Detection in bone X-Rays

Koutsomarkos Alexandros[1] and Lakkas Ioannis[2]

[1]p3352106 , [2]p3352110
Emails: akoutsomarkos@aueb.gr , ilakkas@aueb.gr

April 18, 2023

Google Colab Pre-Trained DenseNet201
Google Colab Pre-Trained ResNet50
Google Colab CNN

1 X-Ray classification

Given a study containing X-Ray images, build a deep learning model that decides if the study is normal or abnormal. You must use at least 2 different architectures, one with your own CNN model (e.g., you can use a model similar to the CNN of the previous project) and one with a popular pre-trained CNN model (e.g., VGG-19, ResNet, etc.). Use the MURA dataset to train and evaluate your models. More information about the task and the dataset can be found at **stanford ML group**.

1.1 Dataset

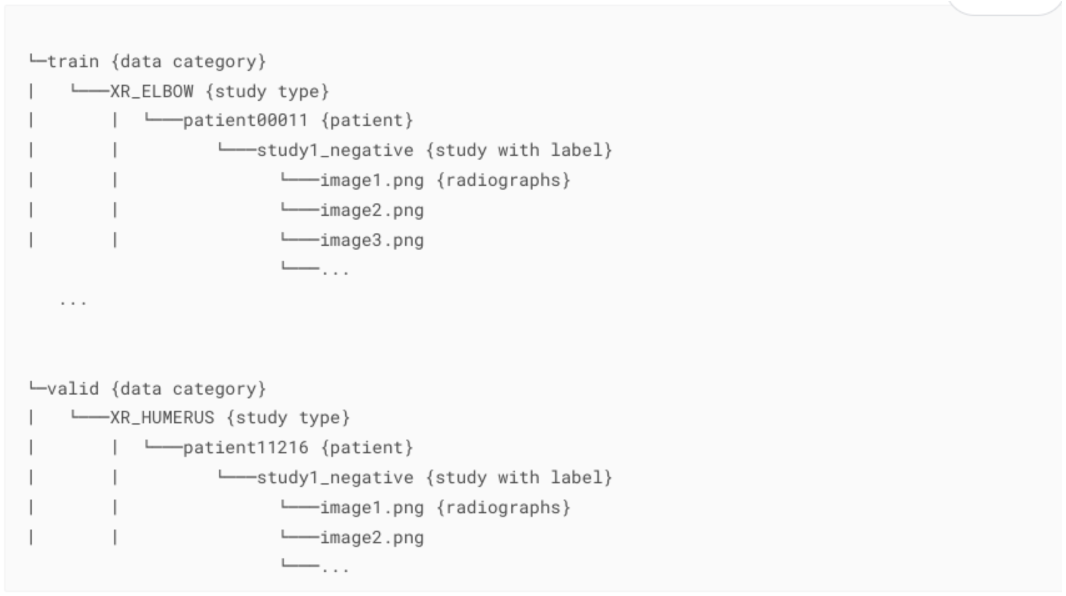
For the purpose of the exercise, we imported the MURA dataset from Stanford. MURA (musculoskeletal radiographs) is a large dataset of bone X-rays. Algorithms are tasked with determining whether an X-ray study is normal or abnormal. Musculoskeletal conditions affect more than 1.7 billion people worldwide, and are the most common cause of severe, long-term pain and disability, with 30 million emergency department visits annually and increasing. MURA is one of the largest public radiographic image datasets.

This data comes as a split dataset (training and validation in separate directories).

- There are 36,808 images in the training set.
- There are 3,197 images in the test set.
- The images have pixel values ranging from 0 to 255.

- The labels are either "positive" (1) or "negative" (0).

Each dataset such as training and validation has sub-directories as shown below.



```
└─train {data category}
|   └───XR_ELBOW {study type}
|       |  └───patient00011 {patient}
|       |         └───study1_negative {study with label}
|       |                └───image1.png {radiographs}
|       |                └───image2.png
|       |                └───image3.png
                         └───...
   ...


└─valid {data category}
|   └───XR_HUMERUS {study type}
|       |  └───patient11216 {patient}
|       |         └───study1_negative {study with label}
|       |                └───image1.png {radiographs}
|       |                └───image2.png
                         └───...
```

Figure 1: Dataset hierarchy

1.2 Exploratory Data Analysis (EDA)

In Figure (2) we can see some examples of the training dataset.

In Figure (3) we can see the same images, but this time with the text removed. For this task we used CV2 and Keras-OCR as explained in this **blog**

Regarding the distribution in our dataset we observe that there are some differences between the different body parts, although in general, we have a fair share of positive as well as negative examples for each body part. We find the smallest ratio of positives/negatives for the Hand category.

Finally in Figure (3) we can see some examples of the training dataset, along with their label.

1.3 Train/Dev/Test split

To ensure the creation of a robust model for inference, we will reserve a portion of the training data as a development set. This will enable us to tune the hyperparameters of the model architectures effectively. Since our dataset has already been split to train/test sets we will reserve a portion of the training data (10%) for the development

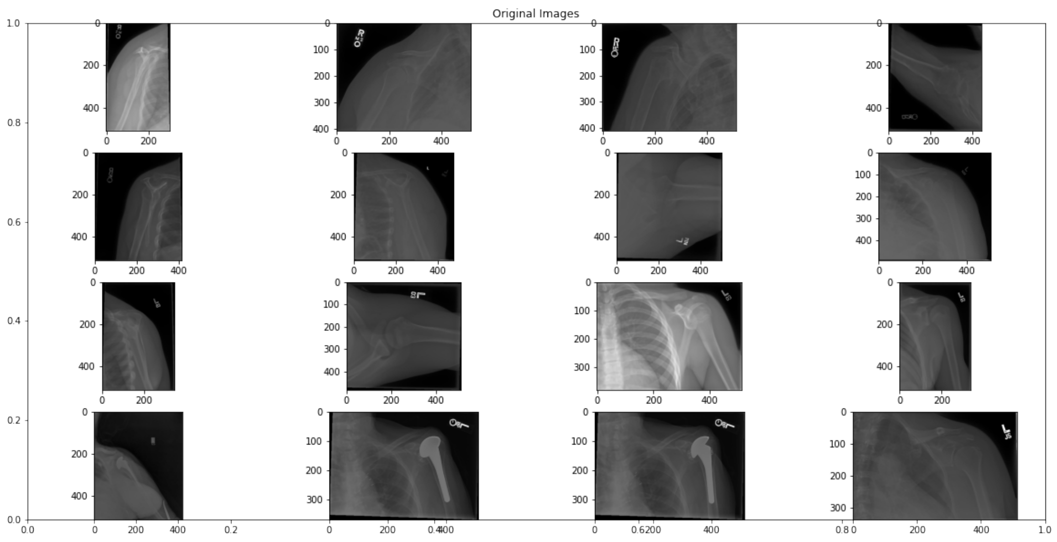


Figure 2: Images of the train dataset.

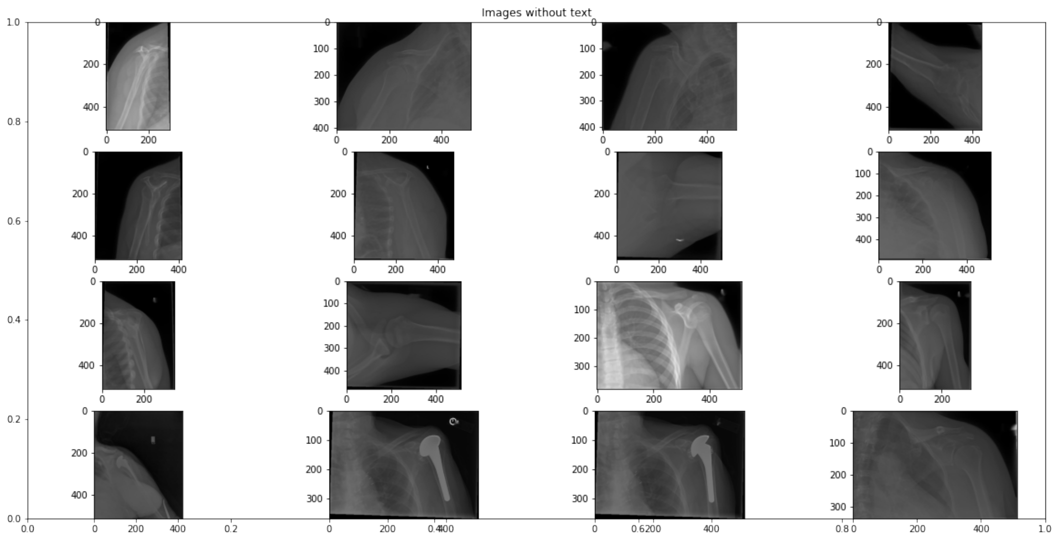
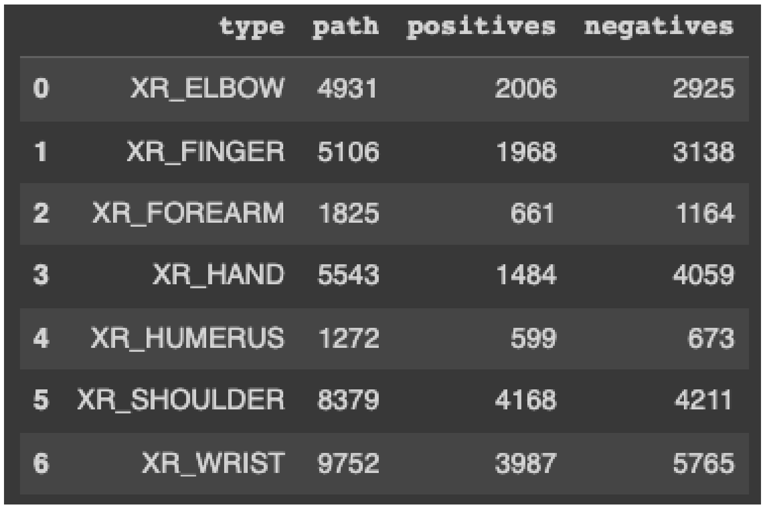


Figure 3: Images of the train dataset with text removed.

set. We shall also be extra careful to retain the uniform nature of the target class distribution so as not to introduce bias. To do this, we shall split the sets in a stratified fashion.



	type	path	positives	negatives
0	XR_ELBOW	4931	2006	2925
1	XR_FINGER	5106	1968	3138
2	XR_FOREARM	1825	661	1164
3	XR_HAND	5543	1484	4059
4	XR_HUMERUS	1272	599	673
5	XR_SHOULDER	8379	4168	4211
6	XR_WRIST	9752	3987	5765

Figure 4: Distribution of the training dataset.

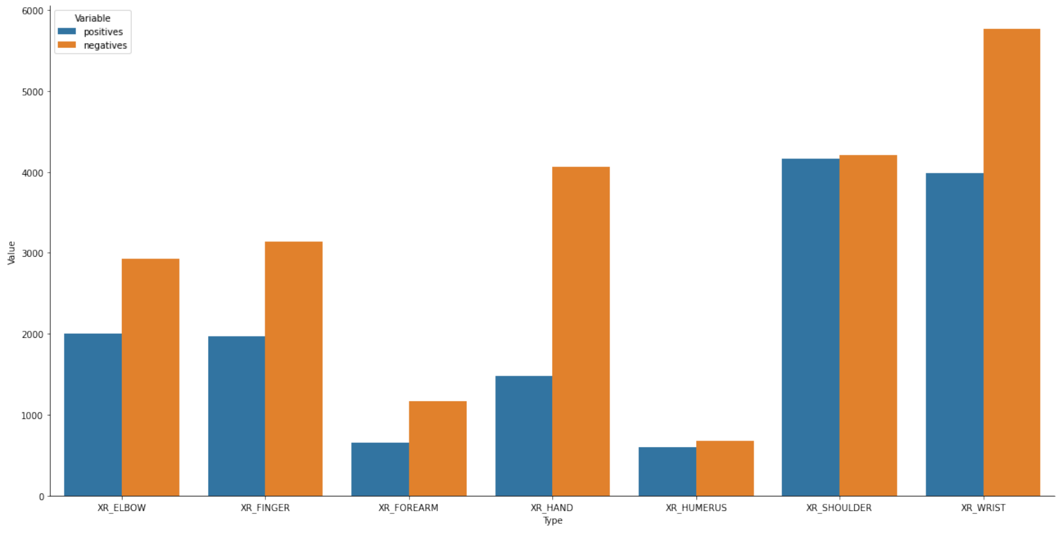


Figure 5: Distribution barplot of the training dataset.

1.4 Pre-Processing

Before proceeding with the development of the models, we had to do some data-prepossessing, using the **ImageDataGenerator**.

ImageDataGenerator is a class in the Keras deep learning library that generates batches of image data with real-time data augmentation. It is commonly used for image classification tasks, where it can be used to preprocess the images and augment the dataset in real-time during model training.

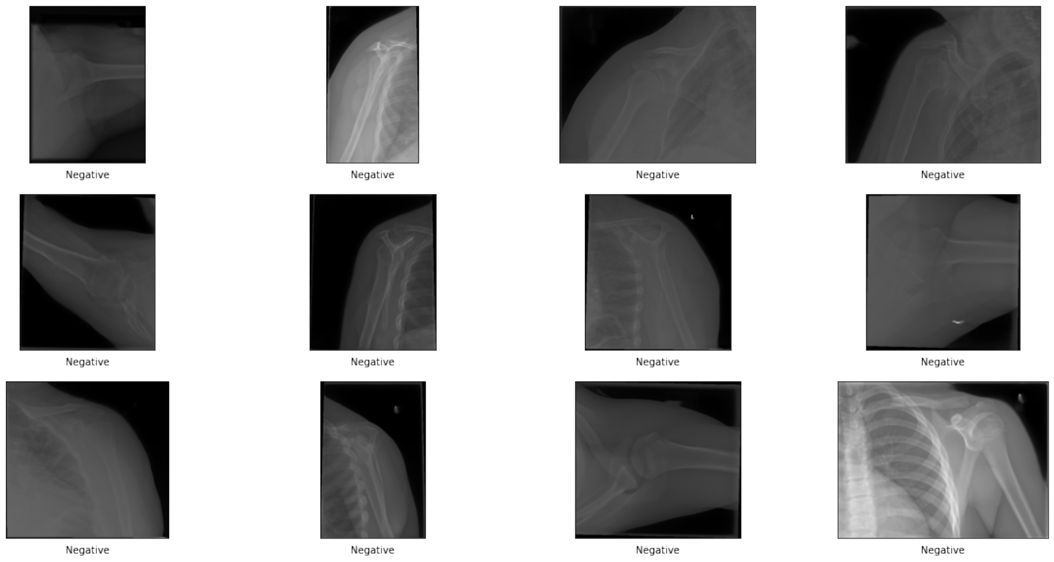


Figure 6: Images of the train dataset with text removed.

The ImageDataGenerator class provides a variety of image augmentation techniques such as rescaling, rotation, zooming, shifting, flipping, shearing, and more. These techniques can help improve the robustness of the model by increasing the diversity of the training data, which can in turn improve the model's accuracy and generalization performance.

In addition to data augmentation, the ImageDataGenerator class also supports other data preprocessing techniques such as data normalization and resizing. It can be used in conjunction with the Keras Sequential or Functional API to train convolutional neural networks (CNNs) and other deep learning models for image classification, segmentation, and other related tasks.

If we inspect the training set, we see that the pixel values fall in the range of 0 to 255. So we normalize them to [0, 1].

Then we create four datasets with the ImageDataGenerator.

dataset	configuration
train original	224x224 pixels, batch size = 64, shuffle
train augmented	224x224 pixels, batch size = 64, shuffle, rotation, flip
validation	224x224 pixels, batch size = 64, shuffle
test	224x224 pixels, batch size = 64, shuffle

Table 1: Datasets

1.5 Pre-Trained Models

We start by building models with transfer learning based on pre-trained models. For this purpose we chose two different pre-trained models i.e. **DenseNet201** and **ResNet50**. The task is to transfer the learning of a ResNet50 and DenseNet201 trained with Imagenet to a model that classify images from MURA dataset.

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
ResNet50	98	74.9%	92.1%	25.6M	107
DenseNet201	80	77.3%	93.6%	20.2M	402

Table 2: Pre-trained Models available in Keras (The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset)

For each model we use four variations, namely keeping the pre-trained model frozen (we do not train the weights of the base model) or not, and adding more Dense layers on top or not. We have a dropout layer after the base model and also after each dense layer with a dropout rate of 20%.

base model	trainable	dense layers	dense layer units
DenseNet201	no	0	0
DenseNet201	yes	0	0
DenseNet201	no	3	128, 128, 64
DenseNet201	yes	3	128, 128, 64
ResNet50	no	0	0
ResNet50	yes	0	0
ResNet50	no	3	128, 128, 64
ResNet50	yes	3	128, 128, 64

Table 3: Model Variations

1.5.1 Model Training

For the training part we chose to use Adam optimizer. The optimizer is imported form **keras** library. For all training variations we chose to configure 10 epochs and early stopping based on validation Cohen Kappa.

In order to fine tune our models we reduce the learning rate from 0.0001 for the frozen models to 0.00001 for the unfrozen models. This is because we need small learning rate to avoid catastrophic forgetting for the base models. Moreover for the unfrozend models we use **ReduceLROnPlateau** which reduces learning rate when a metric has stopped improving. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

DenseNet201

We will opt for max pooling for feature extraction, since we want to "emphasize" any sharp features on the image (e.g hardware placed on human body that most definitely suggests abnormality).

include top : False
weights : imagenet
input shape : (224, 224, 3)
pooling : max
classes : 2

ResNet50

We will opt for max pooling for feature extraction, since we want to "emphasize" any sharp features on the image (e.g hardware placed on human body that most definitely suggests abnormality).

include top : False
weights : imagenet
input shape : (256, 256, 3)
pooling : max
classes : 2

1.5.2 Training Results

DenseNet201

In the below Figures we can see how each model performed, using the different variations. Available are graphs for the training loss, and validation loss, the training accuracy and validation accuracy, the training AUC and validation AUC, the training CohenKappa and validation CohenKappa. Also there are confusion matrices which were produced during the evalution of the models on the test dataset.

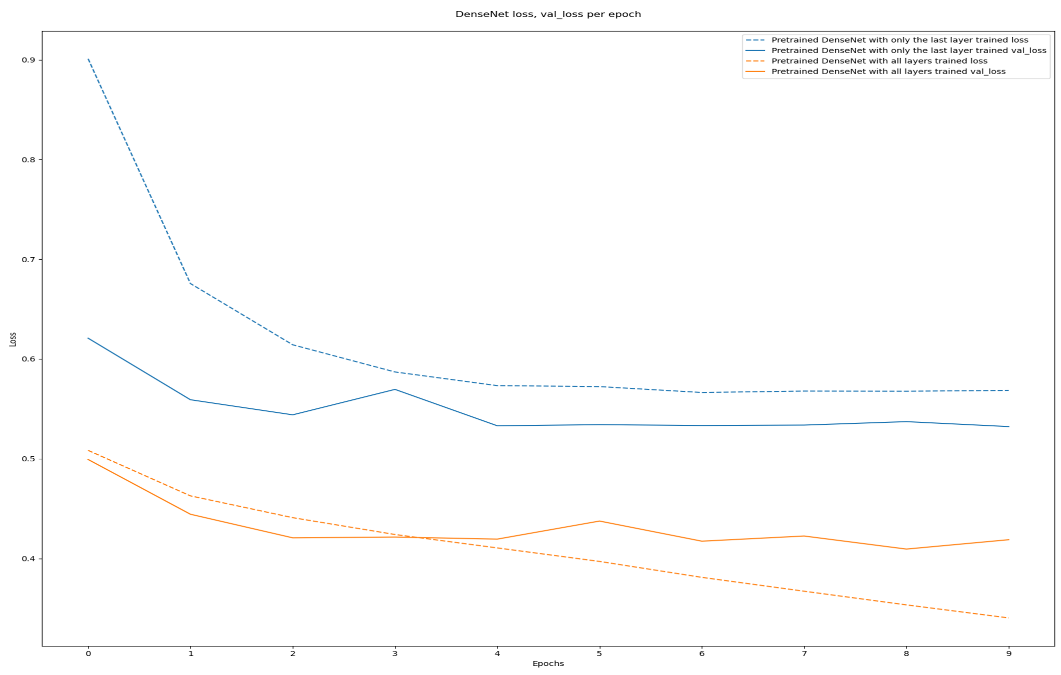


Figure 7: Loss for DenseNet without extra dense layers.

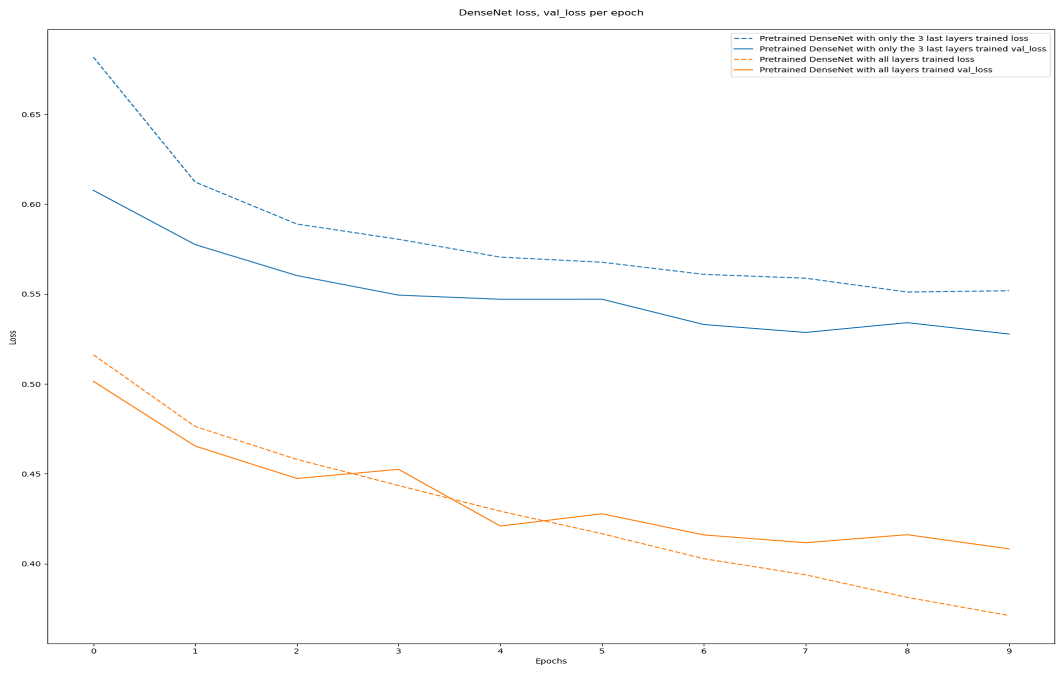


Figure 8: Loss for DenseNet with extra dense layers.

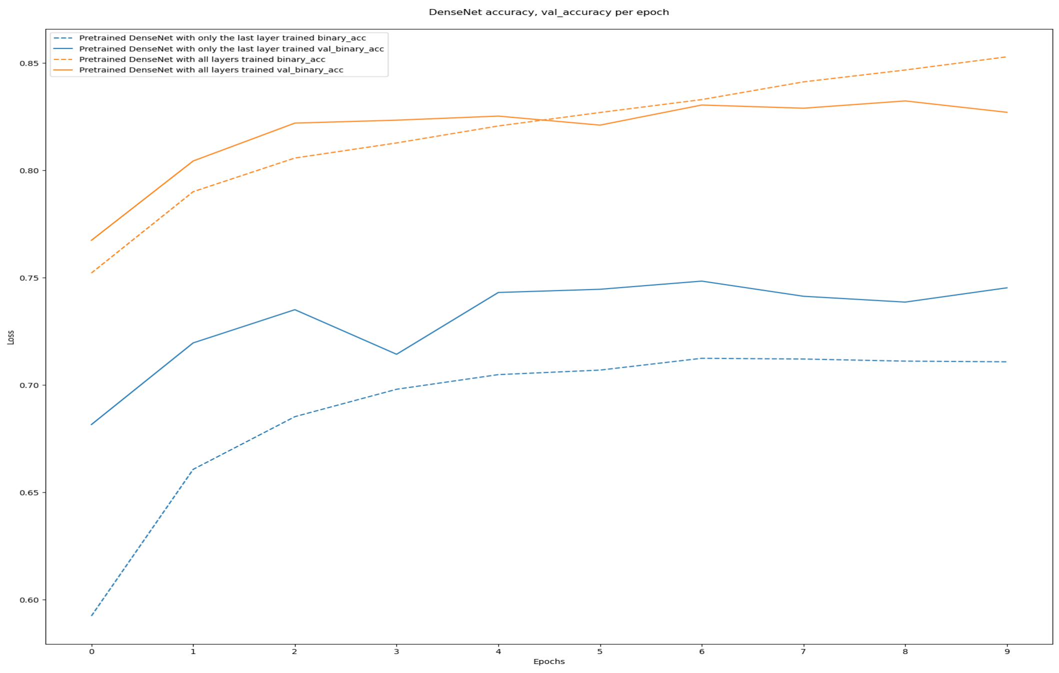


Figure 9: Accuracy for DenseNet without extra dense layers.

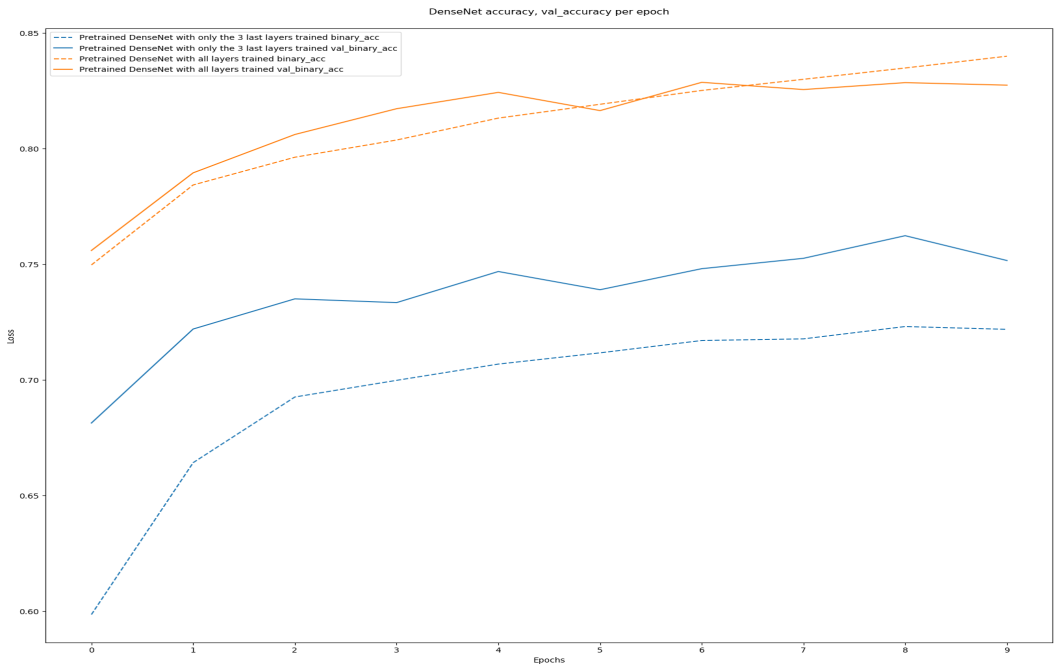


Figure 10: Accuracy for DenseNet with extra dense layers.

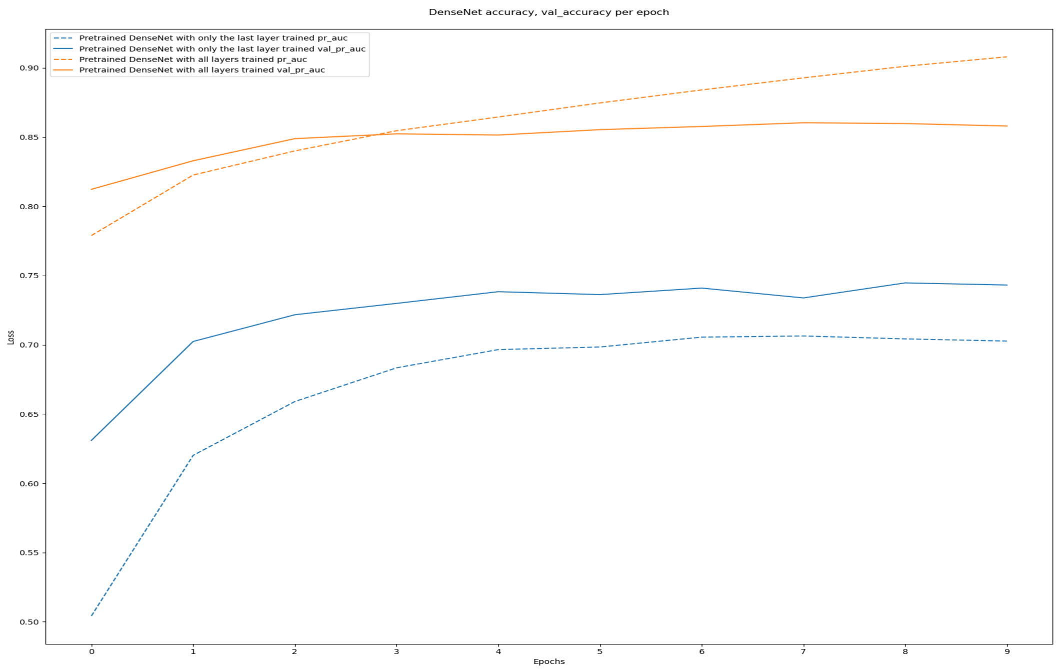


Figure 11: AUC for DenseNet without extra dense layers.

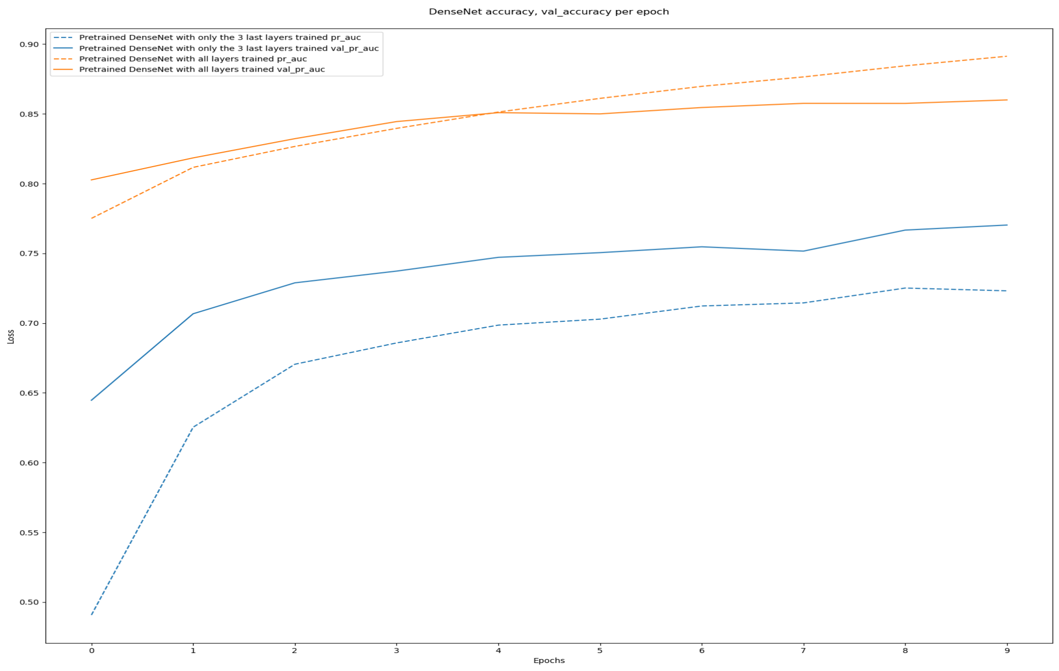


Figure 12: AUC for DenseNet with extra dense layers.

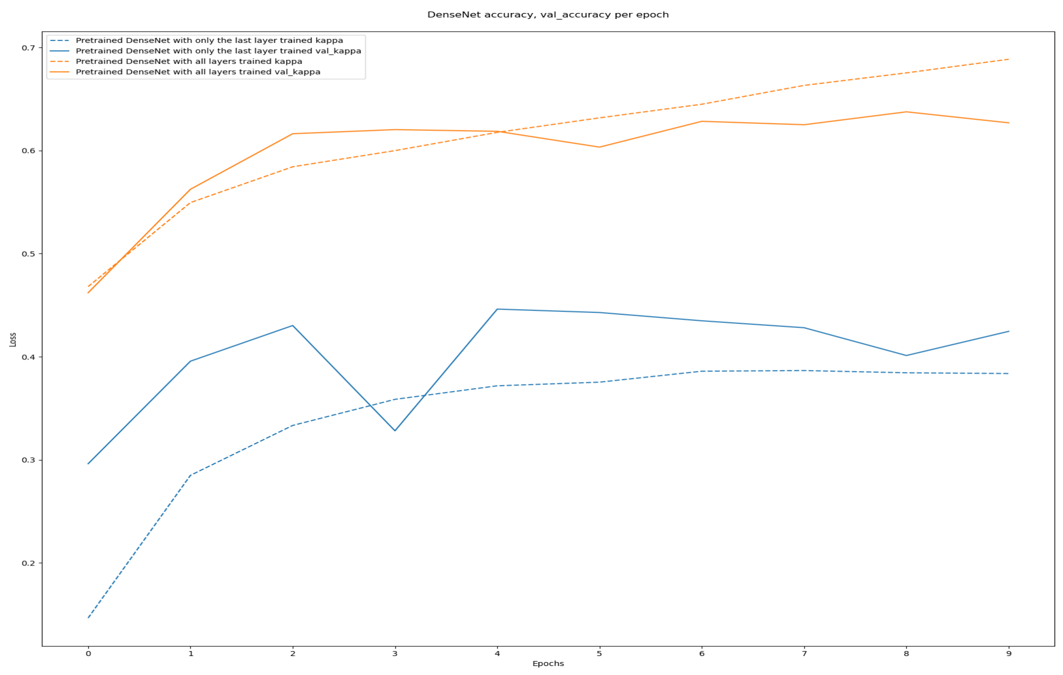


Figure 13: Cohen's Kappa for DenseNet without extra dense layers.

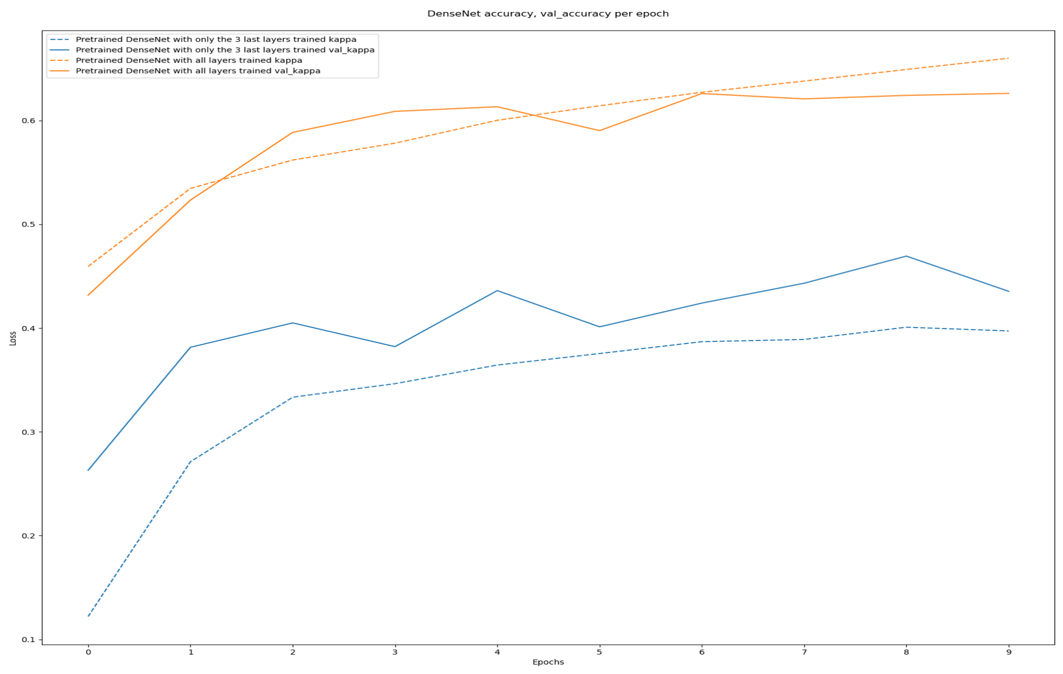


Figure 14: Cohen's Kappa for DenseNet with extra dense layers.

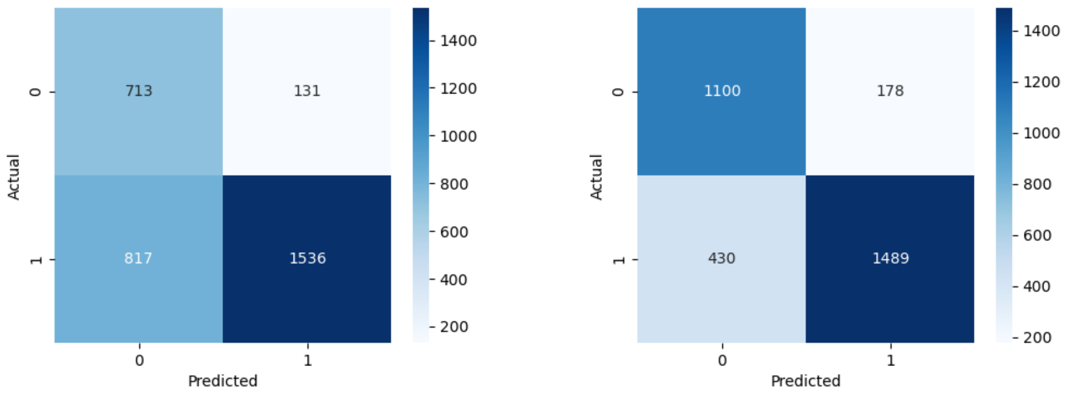


Figure 15: Confusion Matrices for DenseNet without extra dense layers on the test dataset (frozen and unfrozen).

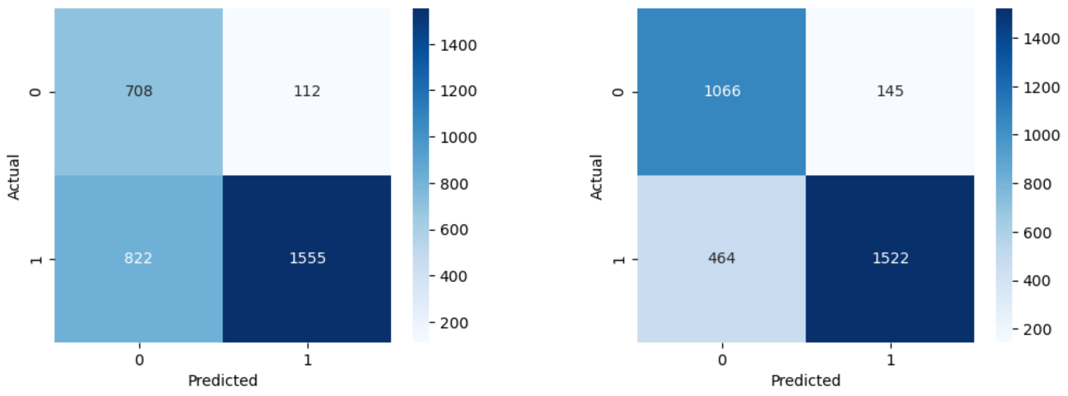


Figure 16: Confusion Matrices for DenseNet with extra dense layers on the test dataset (frozen and unfrozen).

ResNet50

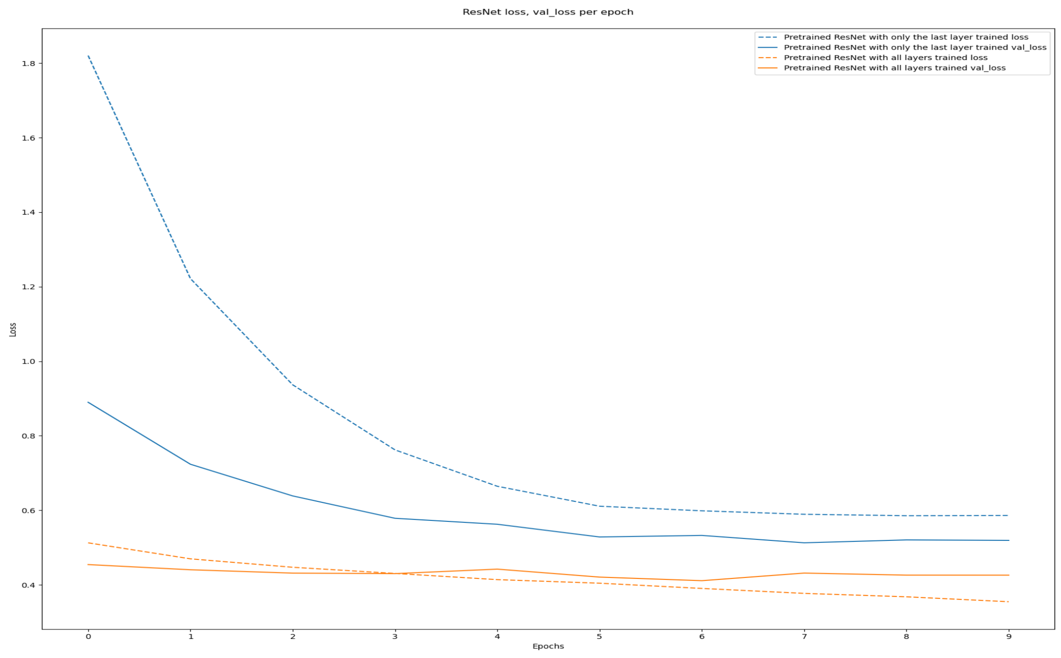


Figure 17: Loss for ResNet without extra dense layers.

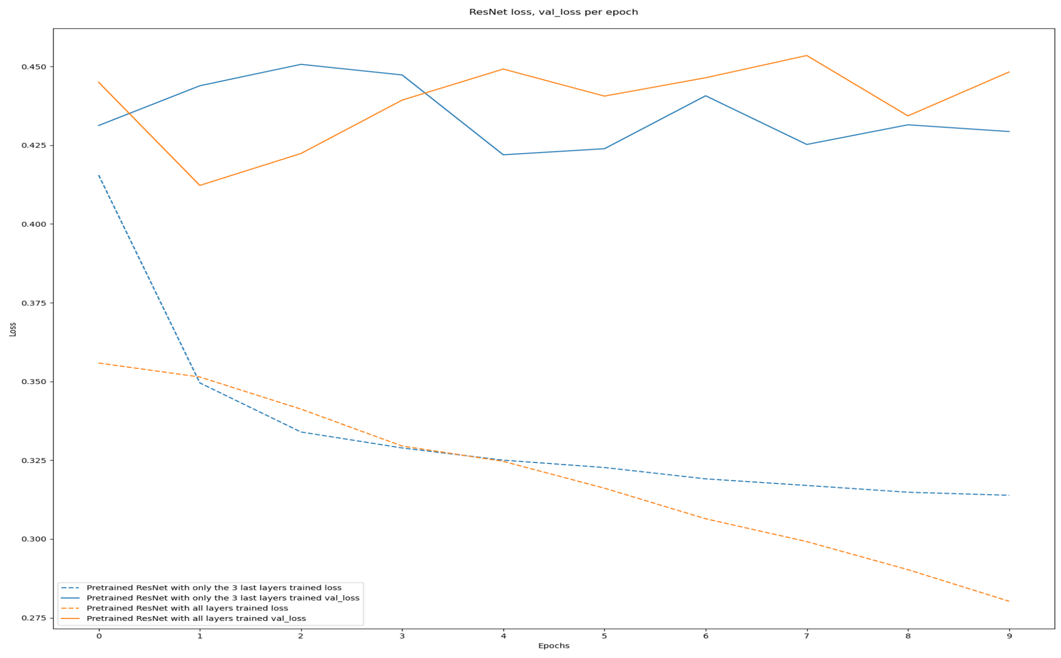


Figure 18: Loss for ResNet with extra dense layers.

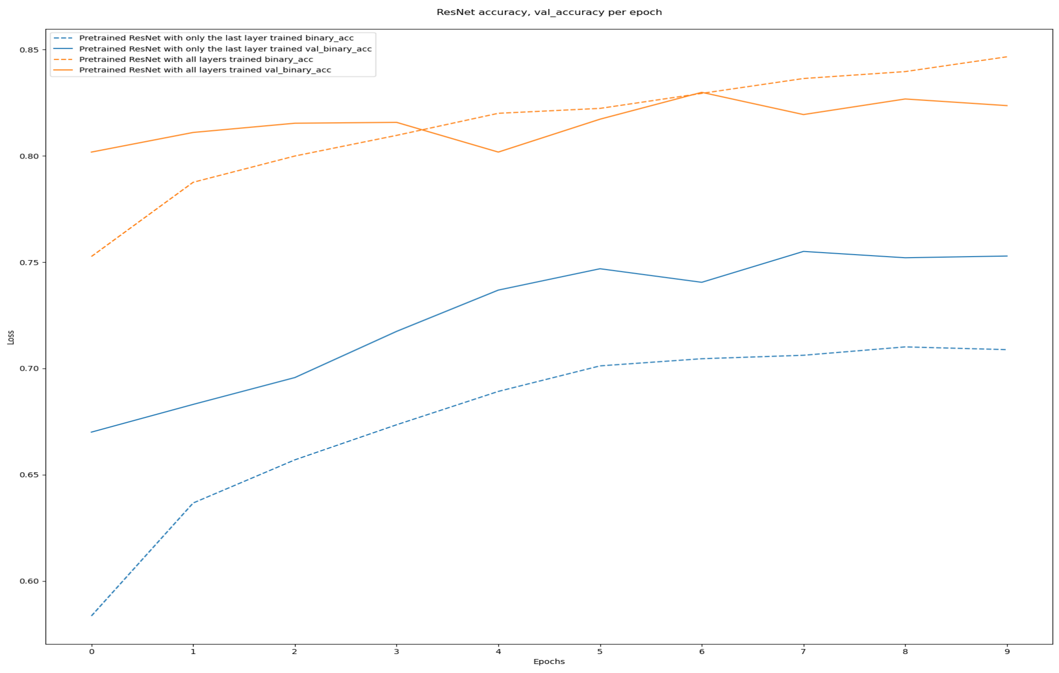


Figure 19: Accuracy for ResNet without extra dense layers.

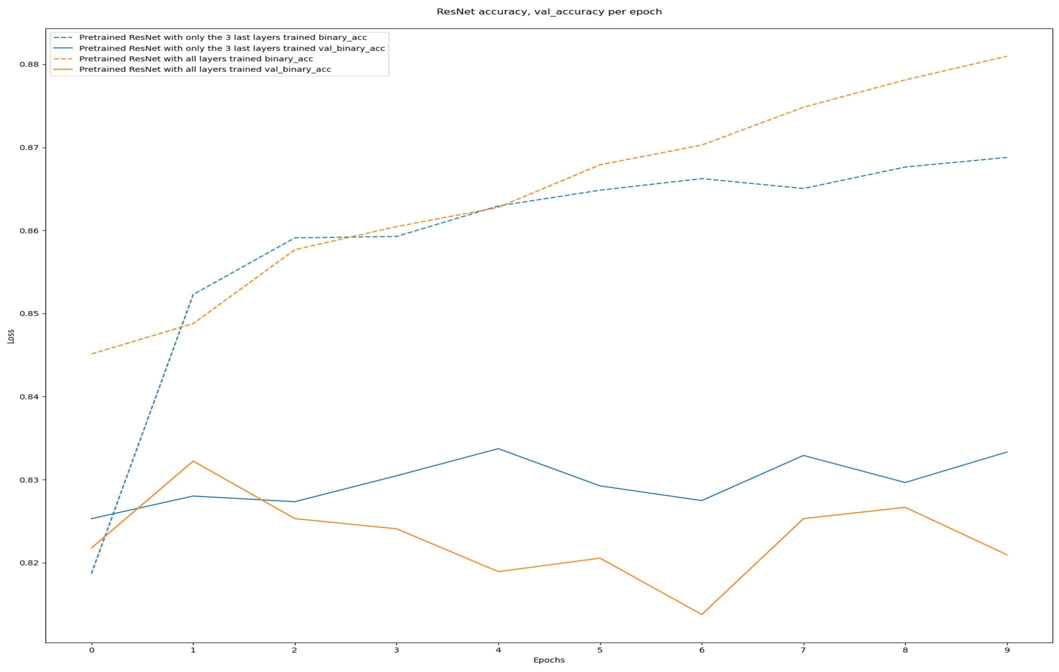


Figure 20: Accuracy for ResNet with extra dense layers.

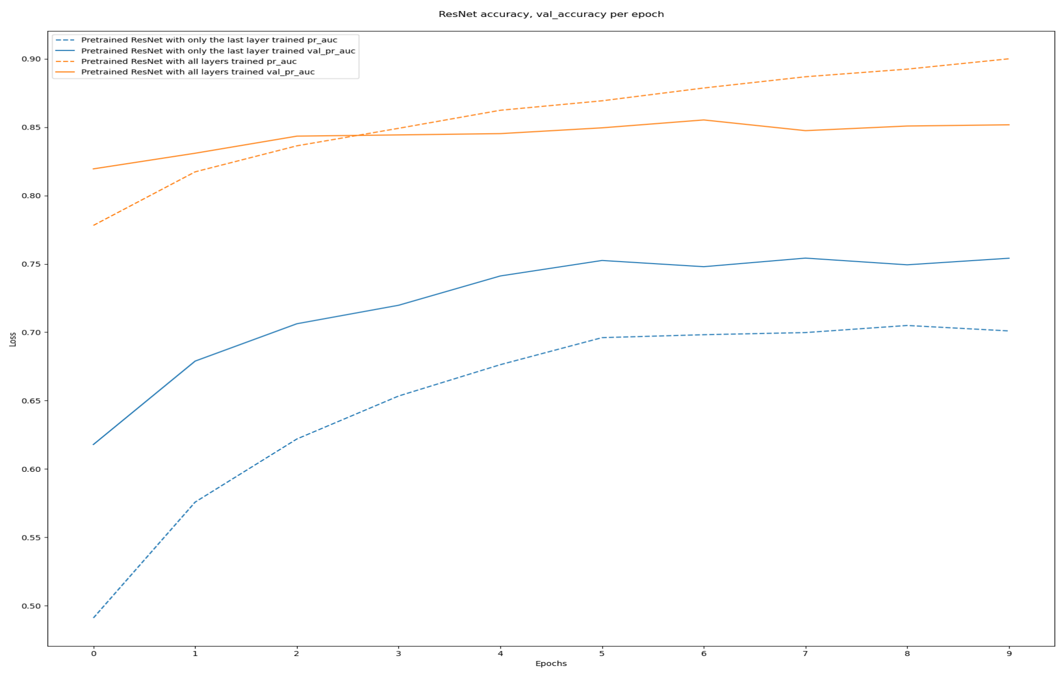


Figure 21: AUC for ResNet without extra dense layers.

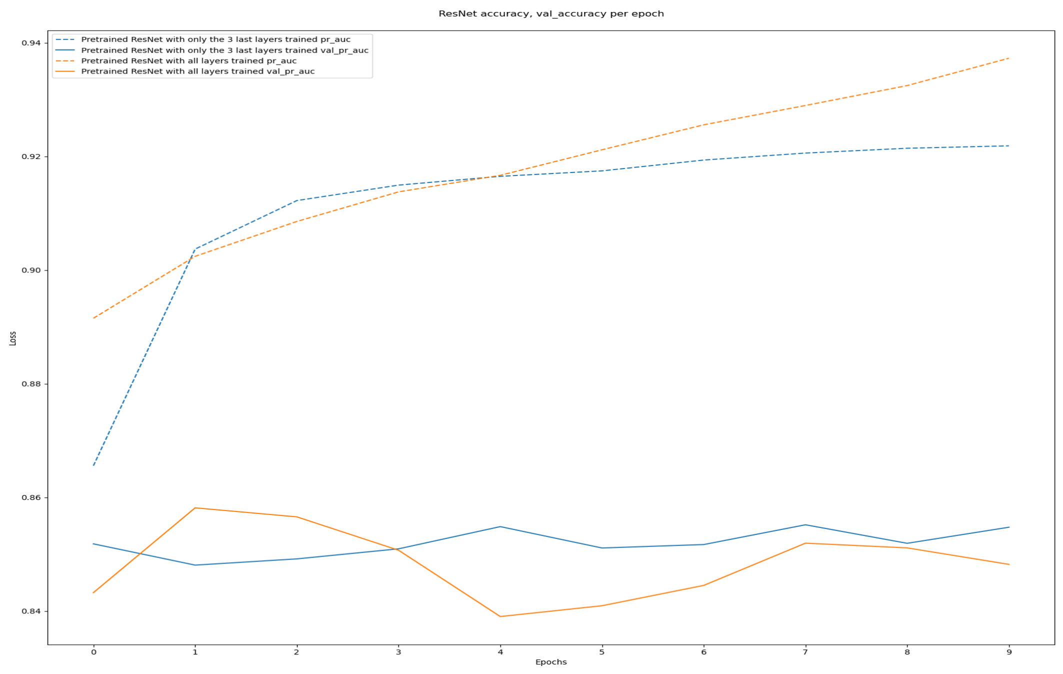


Figure 22: AUC for ResNet with extra dense layers.

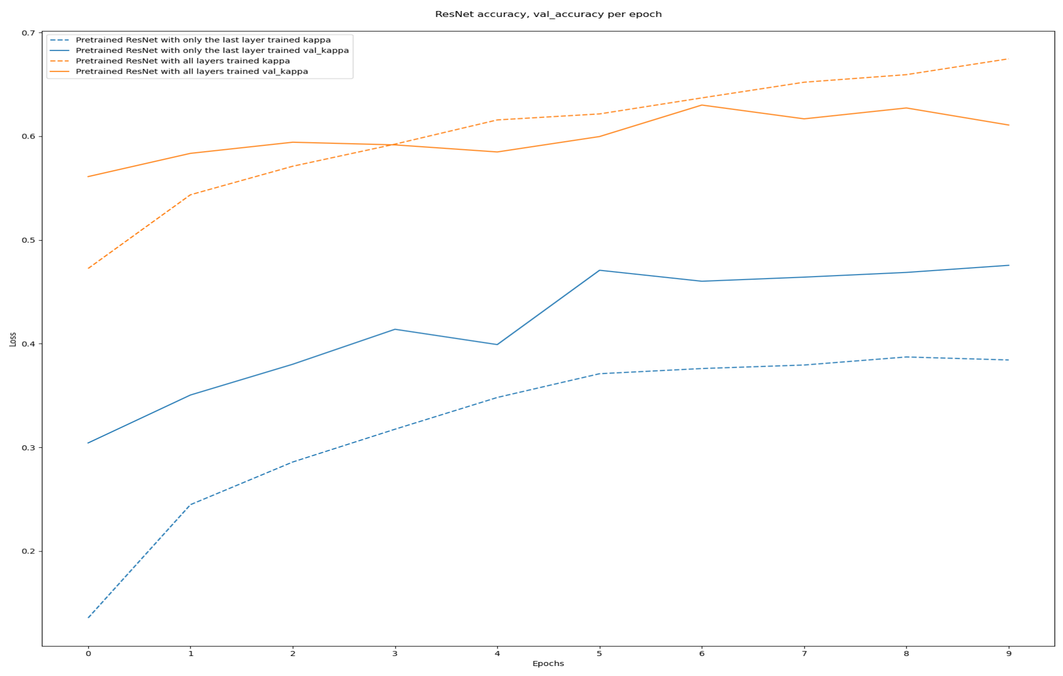


Figure 23: Cohen's Kappa for ResNet without extra dense layers.

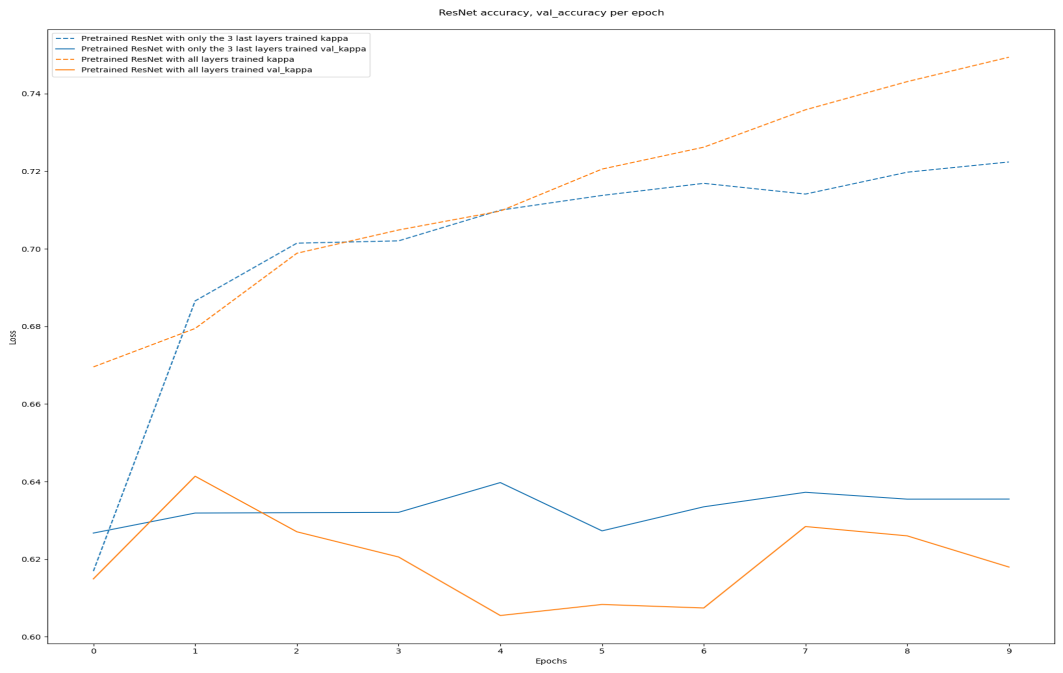


Figure 24: Cohen's Kappa for ResNet with extra dense layers.

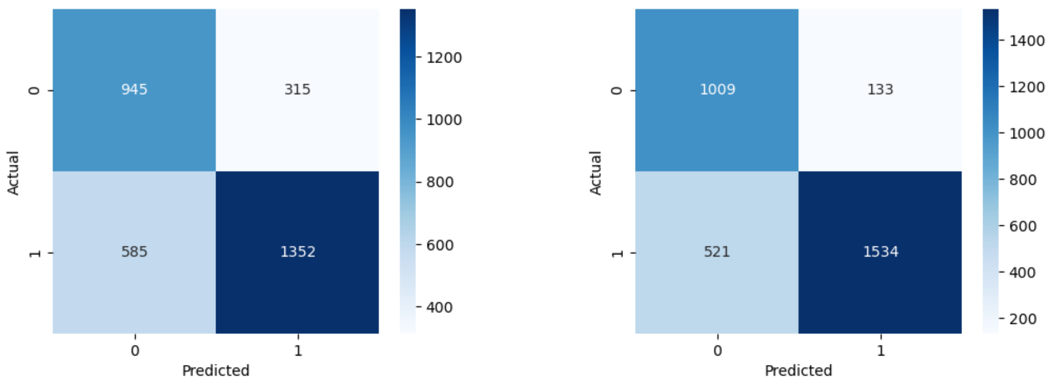


Figure 25: Confusion Matrices for ResNet without extra dense layers on the test dataset (frozen and unfrozen).

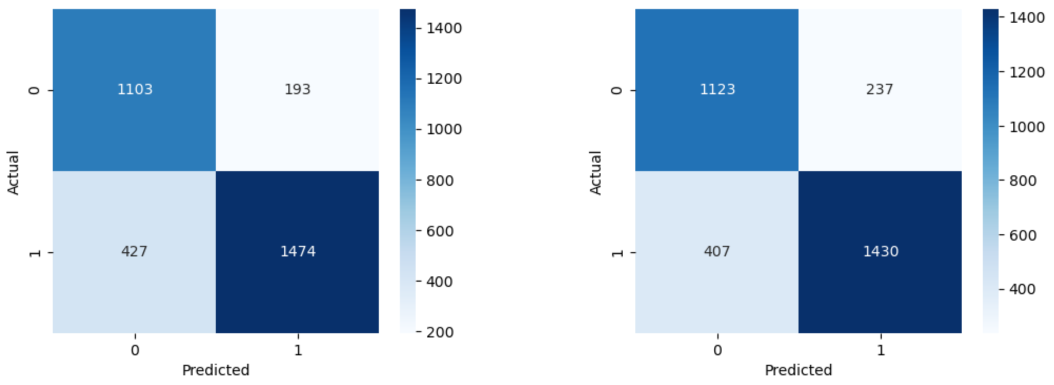


Figure 26: Confusion Matrices for ResNet with extra dense layers on the test dataset (frozen and unfrozen).

1.5.3 Training Analysis

From the results we see that there is no big difference when using extra dense layers on top or not regarding the DenseNet201 models. On the other hand, keeping the base model trainable with a low learning rate delivers better results. This is because the model learns the specifc task more accurately.

For the ResNet50 architecture we have the same observations regarding the extra dense layers, i.e. no significance difference when using them or not. Regarding the base model, when we keep it trainable it seems that we have slighlty better results, although there are signs of overfitting.

A good recommendation when building a model using transfer learning is to first test optimizers to get a low bias and good results in training set, then look for regularizers if you see overfitting over the validation set. Freezing on the pretrained model reduces computation time, reduces overffiting but lowers accuracy. When the new dataset is very different from the datased used for training it may be necessary to use more layer for adjustment. On the selecting of hyperparameters, it is important for transfer learning to use a low learning rate to take advantage of the weights of the pretrained model.

1.6 CNN Models

We start by building our CNN Models which have two main variations in terms of the convolutional layers used. The first approach is a funnel in architecture where the number of filters for each convolutional layer becomes smaller as we get deeper in the network. The second approach is a funnel out architecture where the number of filters for each convolutional layer becomes greater as we get deeper in the network.

For all our models we use the same configuration regarding the convolutional layers and the max pooling layers as shown in Table (4) and Table (5)

kernel size	strides	padding	dilation rate
(3,3)	(1,1)	same	(1,1)

Table 4: Convolutional layers configuration

pool size	strides	padding
(2,2)	(2,2)	same

Table 5: Max pooling layers configuration

In order to find the best model for each architecture we use the **Bayesian Optimization Tuner**. The tunable parameters are the following:

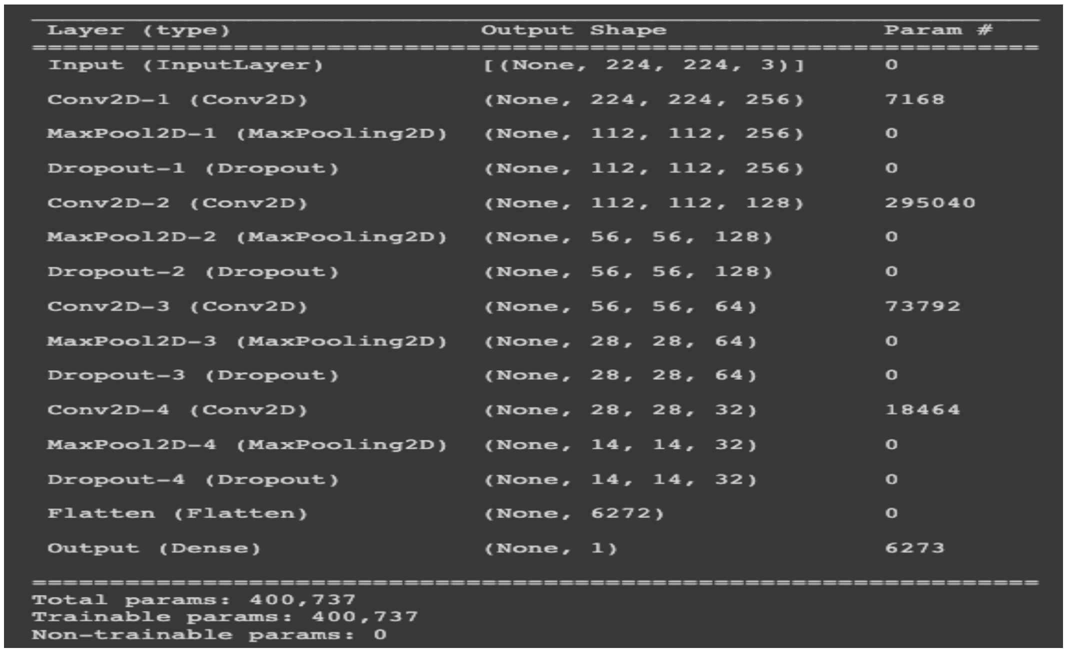
convolutional layers : (4, 5)
dropout rate : (0.3, 0.2)
funnel in : True

1.6.1 Model Training

For the training part we chose to use the Adam optimizer, imported form **keras** library with a learning rate of 0.001. For all training variations we chose to configure 8 epochs and 64 as batch size. We also use early stopping with monitoring of the validation loss and a patience of 5 epochs. Lastly we use **ReduceLROnPlateau** which reduces learning rate when a metric has stopped improving. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

Funnel In

The best model according to the Bayesian optimization tuner is shown bellow in Figure(27) and has 4 convolutional layers and a dropout rate of 0.3:

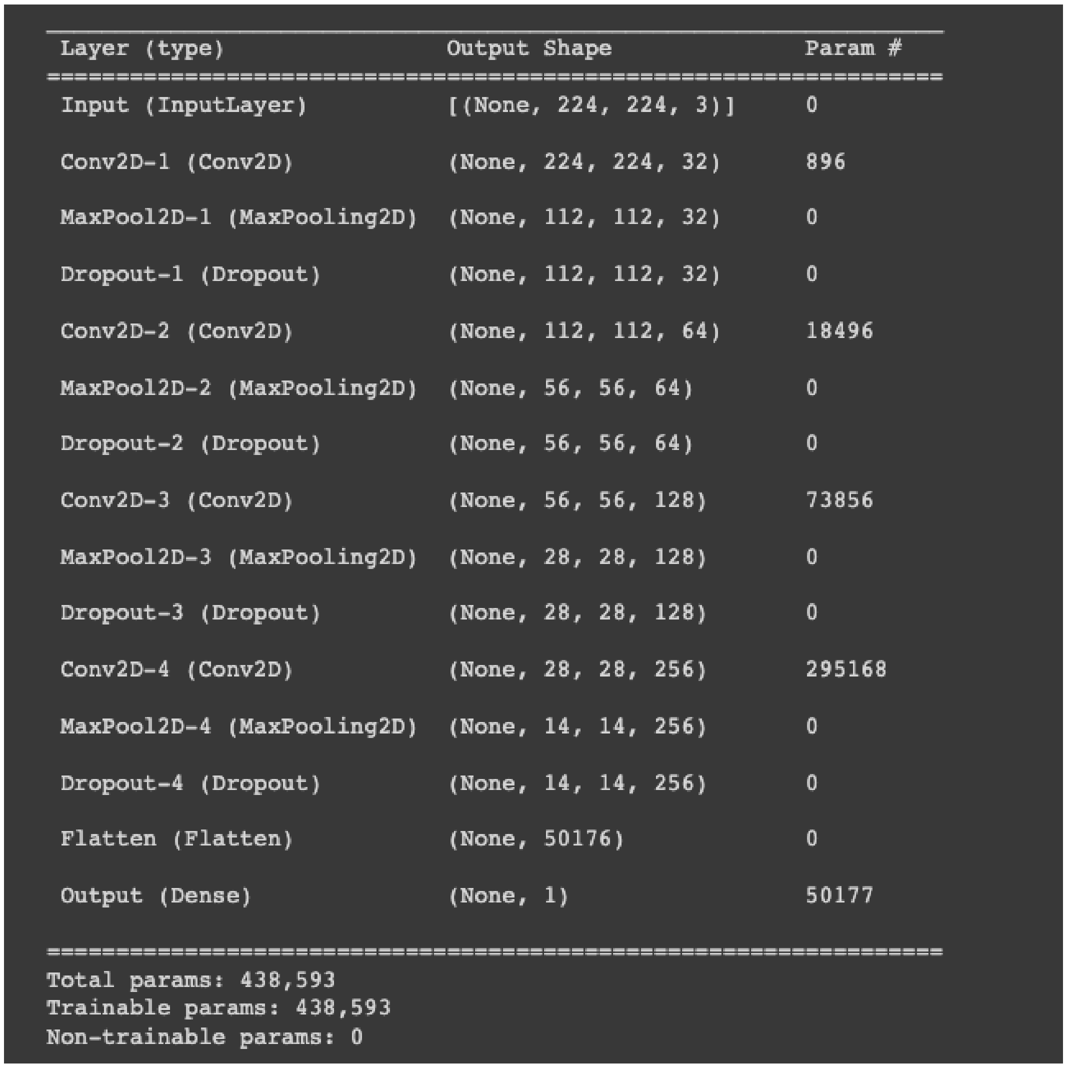


```
 Layer (type)                  Output Shape              Param #
=================================================================
 Input (InputLayer)            [(None, 224, 224, 3)]     0

 Conv2D-1 (Conv2D)             (None, 224, 224, 256)     7168

 MaxPool2D-1 (MaxPooling2D)    (None, 112, 112, 256)     0

 Dropout-1 (Dropout)           (None, 112, 112, 256)     0

 Conv2D-2 (Conv2D)             (None, 112, 112, 128)     295040

 MaxPool2D-2 (MaxPooling2D)    (None, 56, 56, 128)       0

 Dropout-2 (Dropout)           (None, 56, 56, 128)       0

 Conv2D-3 (Conv2D)             (None, 56, 56, 64)        73792

 MaxPool2D-3 (MaxPooling2D)    (None, 28, 28, 64)        0

 Dropout-3 (Dropout)           (None, 28, 28, 64)        0

 Conv2D-4 (Conv2D)             (None, 28, 28, 32)        18464

 MaxPool2D-4 (MaxPooling2D)    (None, 14, 14, 32)        0

 Dropout-4 (Dropout)           (None, 14, 14, 32)        0

 Flatten (Flatten)             (None, 6272)              0

 Output (Dense)                (None, 1)                 6273

=================================================================
Total params: 400,737
Trainable params: 400,737
Non-trainable params: 0
```

Figure 27: Funnel In CNN Model.

Funnel Out

The best model according to the Bayesian optimization tuner is shown bellow in Figure (28) and has 5 convolutional layers and a dropout rate of 0.3:



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

 Conv2D-1 (Conv2D)           (None, 224, 224, 32)      896

 MaxPool2D-1 (MaxPooling2D)  (None, 112, 112, 32)      0

 Dropout-1 (Dropout)         (None, 112, 112, 32)      0

 Conv2D-2 (Conv2D)           (None, 112, 112, 64)      18496

 MaxPool2D-2 (MaxPooling2D)  (None, 56, 56, 64)        0

 Dropout-2 (Dropout)         (None, 56, 56, 64)        0

 Conv2D-3 (Conv2D)           (None, 56, 56, 128)       73856

 MaxPool2D-3 (MaxPooling2D)  (None, 28, 28, 128)       0

 Dropout-3 (Dropout)         (None, 28, 28, 128)       0

 Conv2D-4 (Conv2D)           (None, 28, 28, 256)       295168

 MaxPool2D-4 (MaxPooling2D)  (None, 14, 14, 256)       0

 Dropout-4 (Dropout)         (None, 14, 14, 256)       0

 Flatten (Flatten)           (None, 50176)             0

 Output (Dense)              (None, 1)                 50177

=================================================================
Total params: 438,593
Trainable params: 438,593
Non-trainable params: 0
```

Figure 28: Funnel In CNN Model.

1.6.2 Training Results

Funnel In

In the below Figures we can see how each model performed, using the different variations. Available are graphs for the training loss, and validation loss, the training accuracy and validation accuracy, the training AUC and validation AUC, the training CohenKappa and validation CohenKappa. Also there are confusion matrices which were produced during the evalution of the models on the test dataset.

Here we have to mention that the models trained with the augmented dataset had the best results.

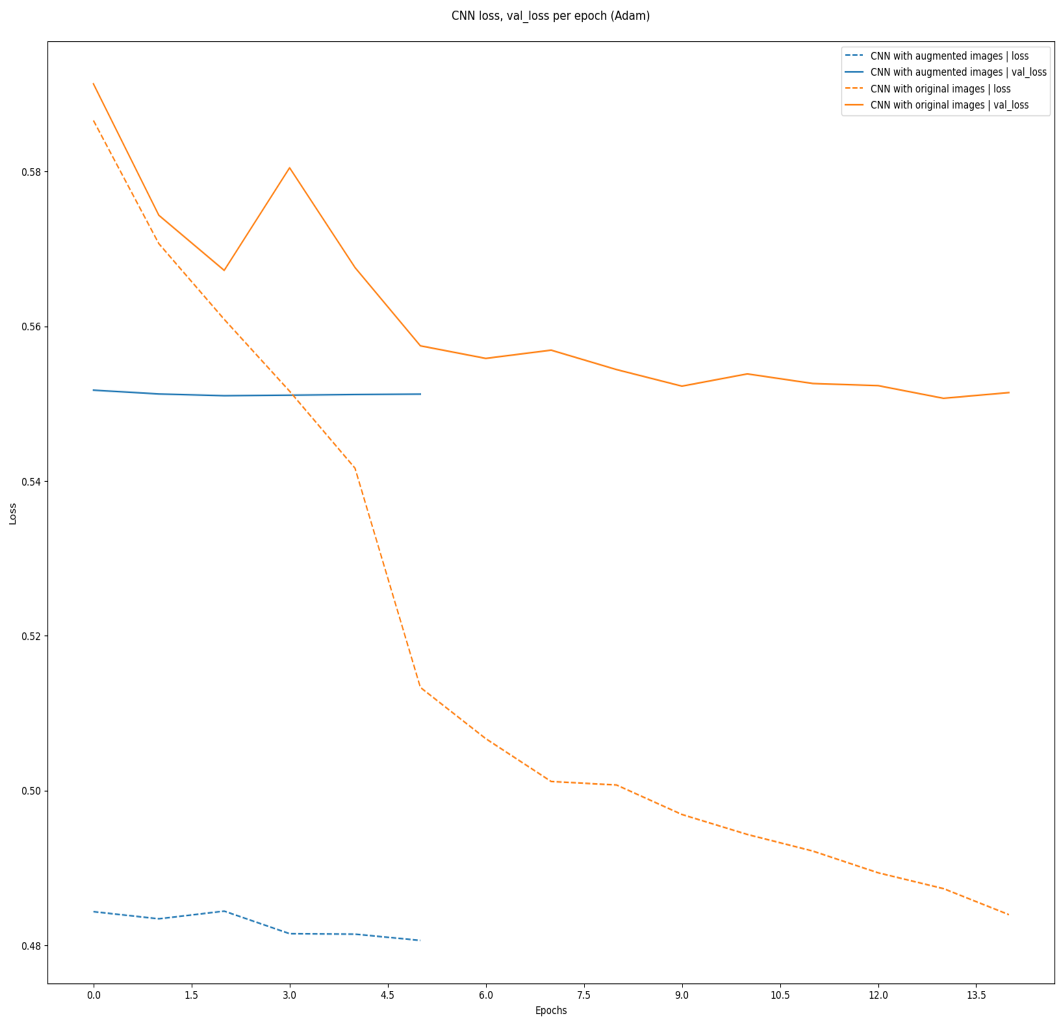


Figure 29: Loss for CNN with funnel-in architecture.

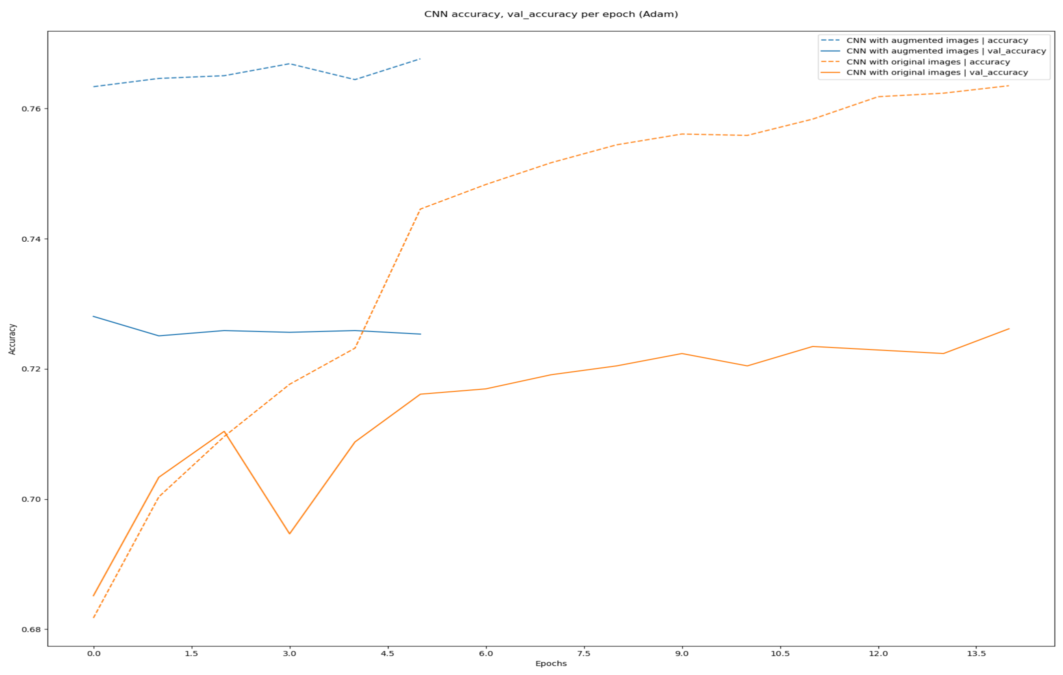


Figure 30: Accuracy for CNN with funnel-in architecture.

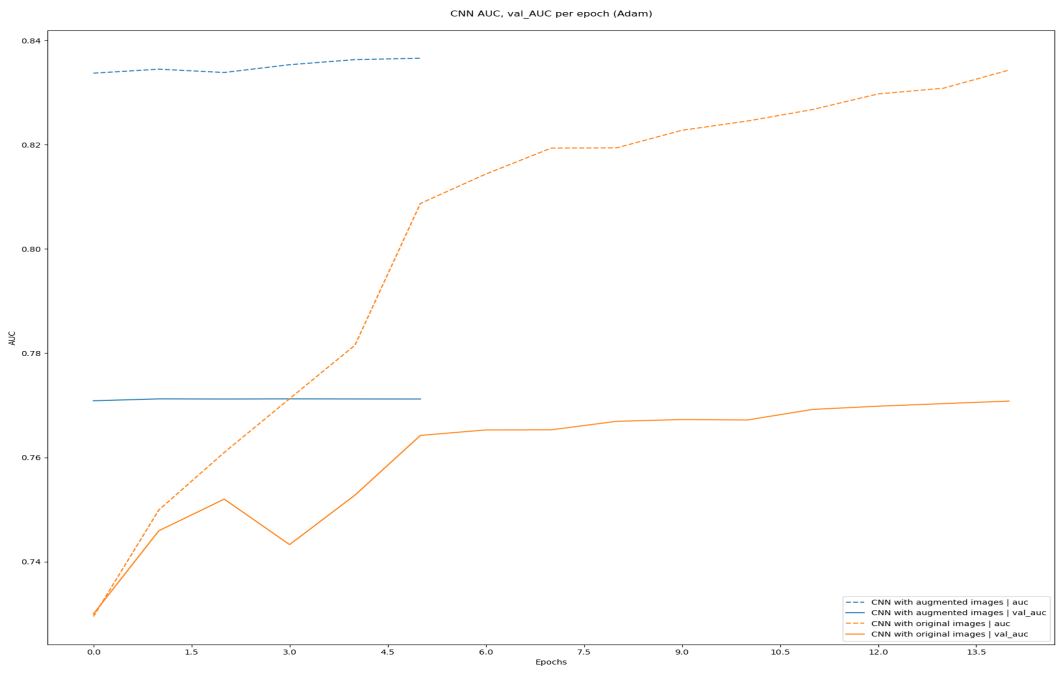


Figure 31: AUC for CNN with funnel-in architecture.

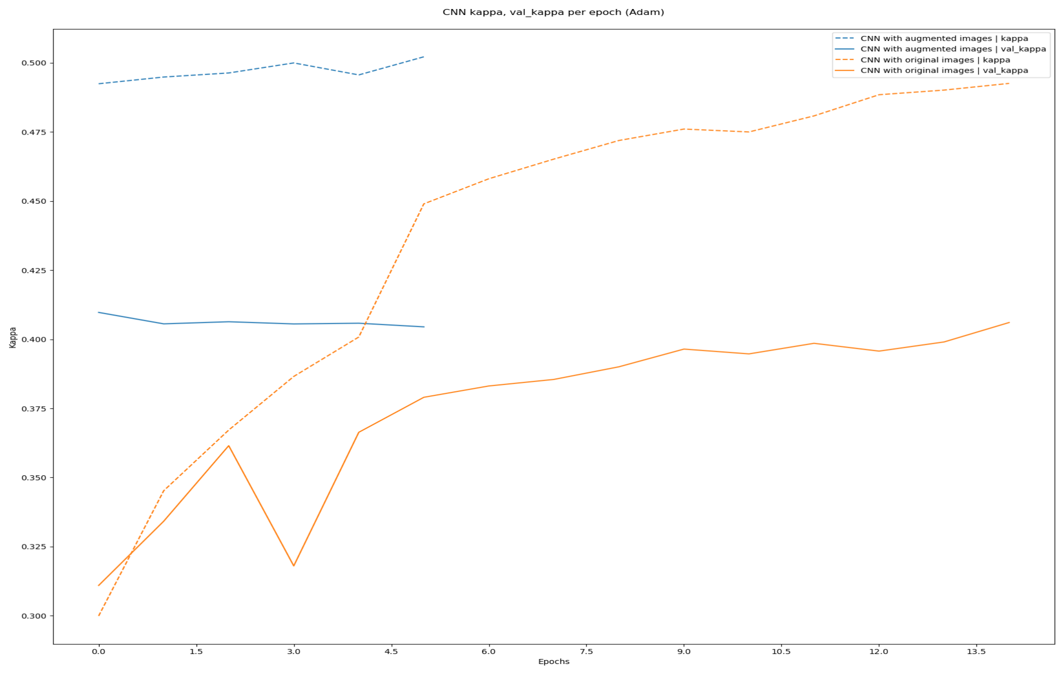


Figure 32: Cohen's Kappa for CNN with funnel-in architecture.

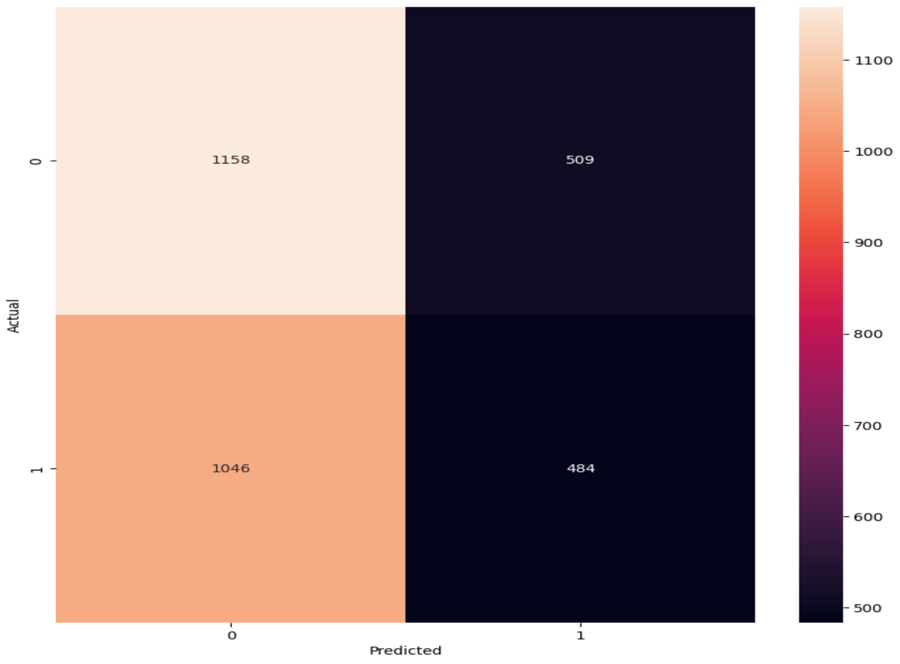


Figure 33: Confusion Matrix for CNN with funnel-in architecture on the test dataset.

Funnel Out

In the below Figures we can see how each model performed, using the different variations. Available are graphs for the training loss, and validation loss, the training accuracy and validation accuracy, the training AUC and validation AUC, the training CohenKappa and validation CohenKappa. Also there are confusion matrices which were produced during the evalution of the models on the test dataset.

Here we have to mention that the models trained with the augmented dataset had the best results.

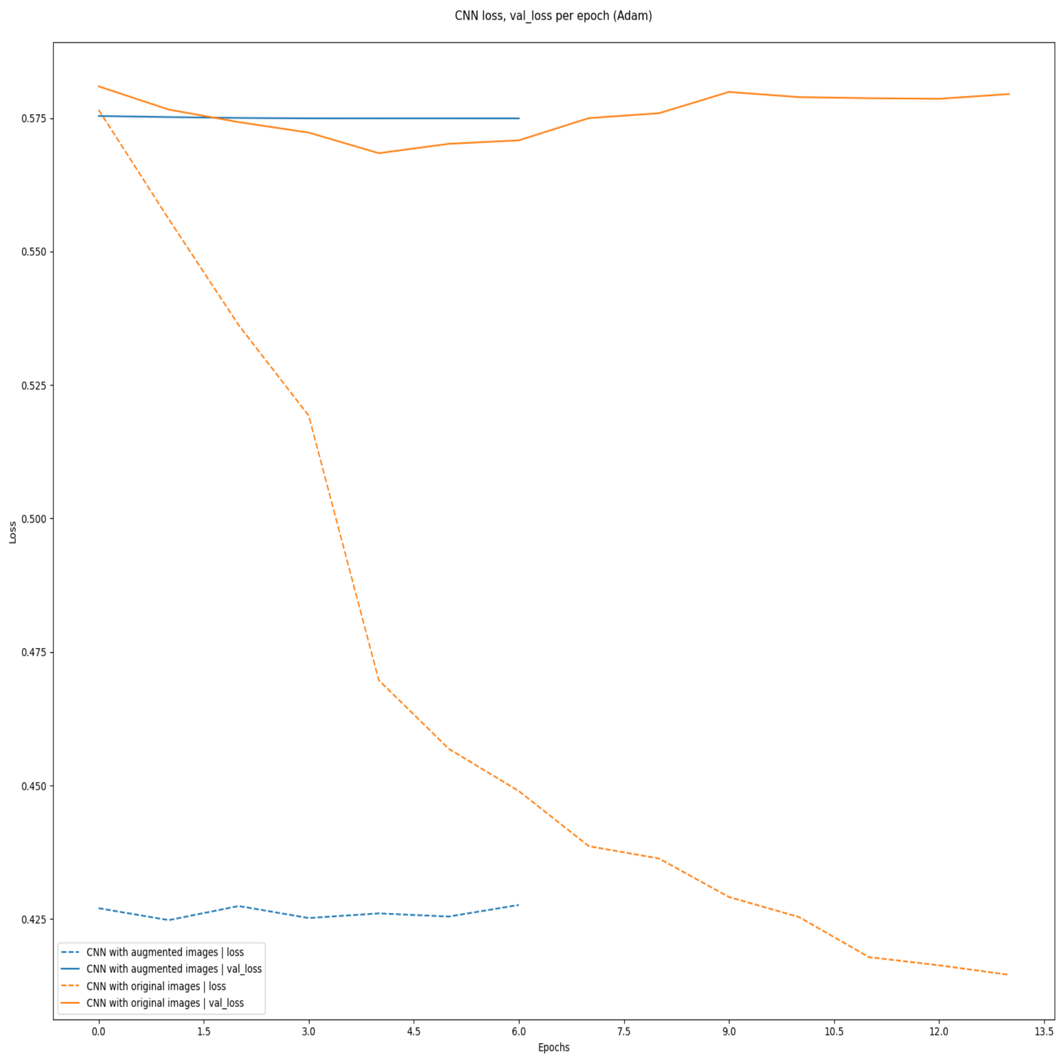


Figure 34: Loss for CNN with funnel-out architecture.

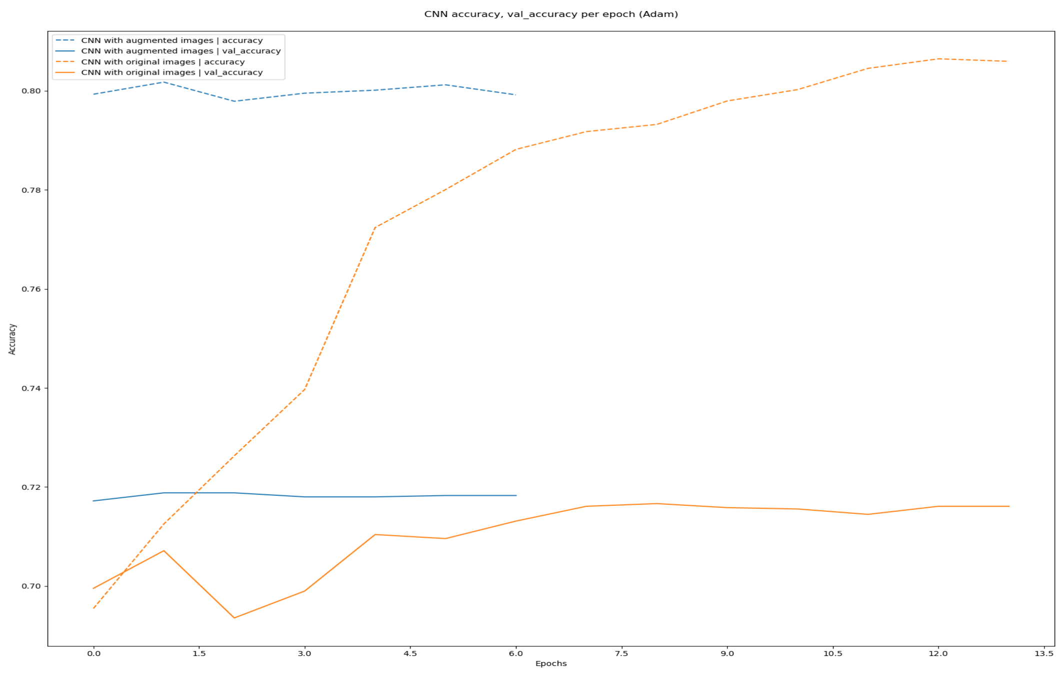


Figure 35: Accuracy for CNN with funnel-out architecture.

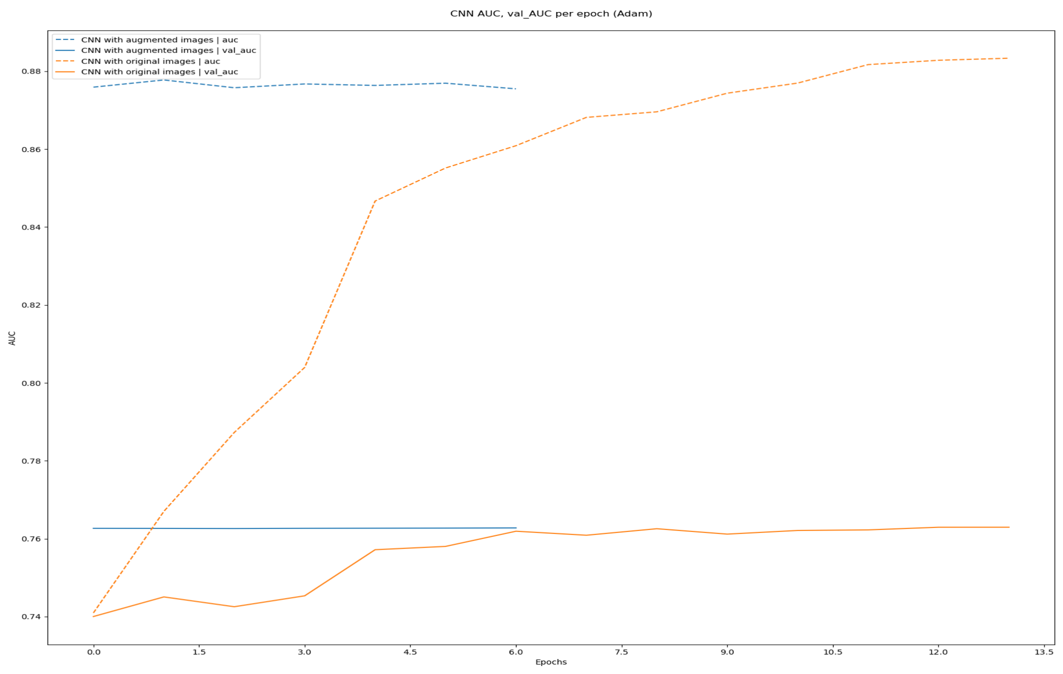


Figure 36: AUC for CNN with funnel-out architecture.

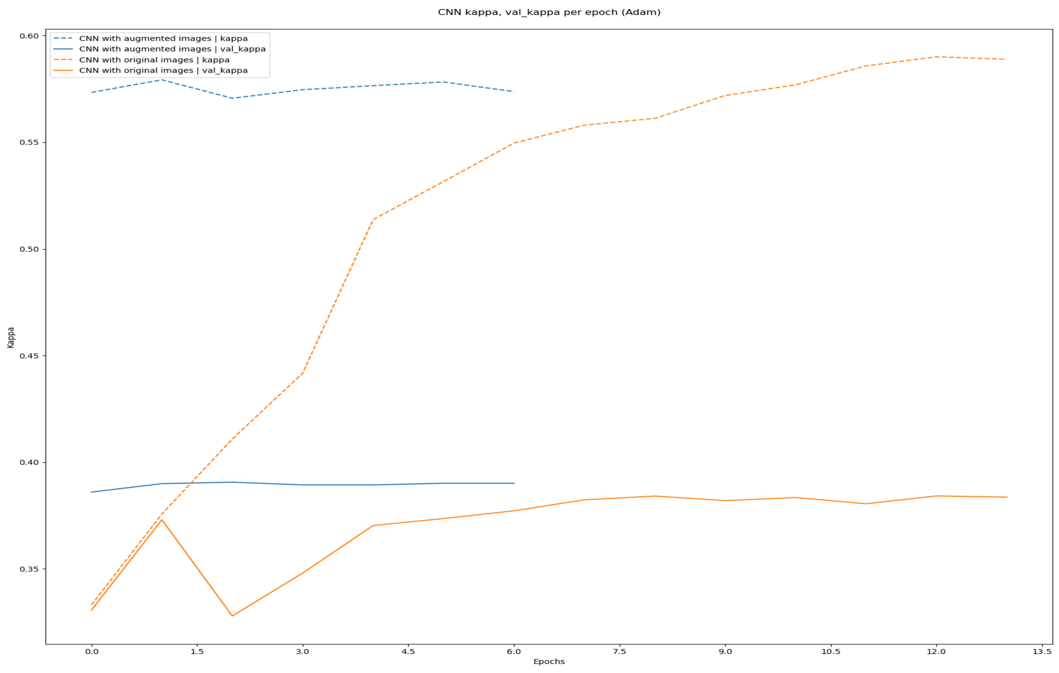


Figure 37: Cohen's Kappa for CNN with funnel-out architecture.

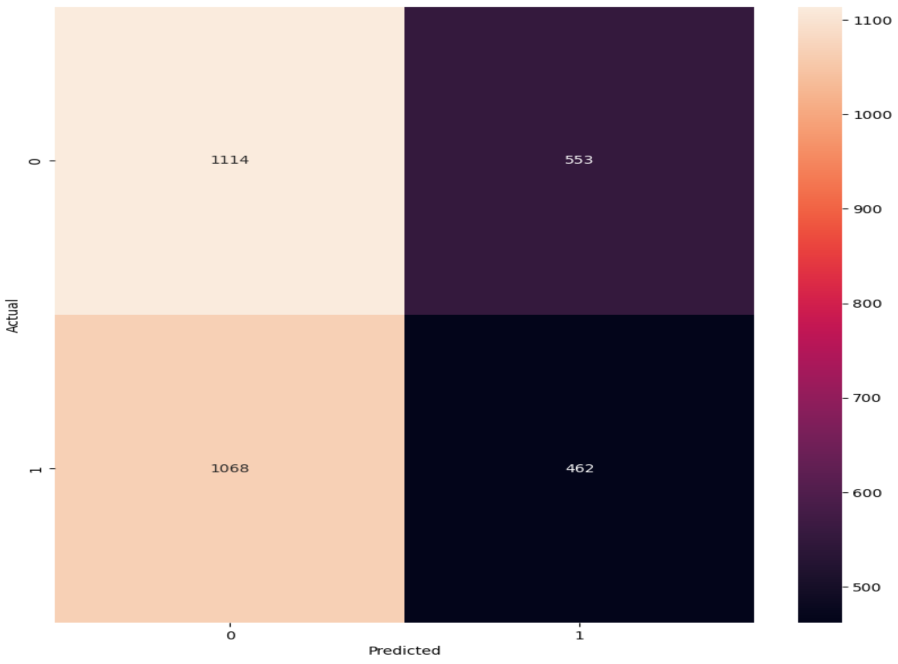


Figure 38: Confusion Matrix for CNN with funnel-out architecture on the test dataset.

1.6.3 Training Analysis

Using the evaluate function on the test dataset for the two models we have the following results. Again it should be mentioned that the used models were trained on the augmented dataset.

The Funnel In architecture delivers the best results between the two CNN architectures we chose, as we can see in the below tables. In general we should mention that the transfer learning process with the pre-trained models delivered much better results in comparison with the custom CNN models. This is because transfer learning allows to circumvent the need for lots of new data. A model that has already been trained on a task for which labeled training data is plentiful will be able to handle a new but similar task with far less data.

Funnel In

Loss	AUC	Accuracy	Cohen's Kappa
0.6259	0.7327	0.6756	0.3406

Table 6: Results of the Funnel-In Model on the test dataset.

Funnel Out

Loss	AUC	Accuracy	Cohen's Kappa
0.6869	0.7119	0.6556	0.3003

Table 7: Results of the Funnel-Out Model on the test dataset.

1.7 Future Work

In this deep learning assignment, we have successfully trained and tested several pre-trained models on our dataset, achieving reasonable performance. However, there is always room for improvement and further exploration.

One potential avenue for future work is to increase the number of epochs during training. By increasing the number of epochs, we can allow the model to better learn the underlying patterns in the data, potentially leading to improved accuracy. We will monitor the training progress and the validation performance to ensure that the model is not overfitting to the training data. Increasing the number of epochs during training is a potentially fruitful approach for improving model accuracy. With additional training time, the model can more effectively learn the underlying patterns in the data, potentially leading to improved performance on the validation set. However, it is important to carefully monitor the training process to prevent overfitting. Due to resource limitations, we were only able to train each model for 10 epochs. It should be noted that all models, with the exception of one, completed training for the full 10 epochs without early stopping. Moving forward, it may be valuable to investigate the impact of longer training times on model performance, while still ensuring that overfitting does not occur.

Another area for future exploration is the use of confusion plots per class. These plots provide insight into which classes the model is struggling to distinguish, allowing us to identify areas for further improvement. We can use these plots to guide our efforts to adjust the model or augment the data.

In addition, we plan to use ensemble models in future work. By combining the predictions of multiple models, we can potentially achieve higher accuracy than any individual model could achieve on its own. Given that many of the top-ranking projects in the competition related to the data utilized ensemble models, we believe this approach could yield substantial improvements in accuracy. So, we will experiment with using different types of models, such as ResNet 50 and DenseNet, to see which combinations perform best.

Furthermore, we can use also **multi-task learning**. Multi-task learning is a technique that involves training a single model on multiple related tasks. In the context of classifying bone X-rays in the MURA dataset, the related tasks could include identifying the presence of a fracture, determining the type of fracture, and assessing the severity of the fracture. By jointly training the model on these related tasks, the model can leverage the shared information between the tasks and learn better representations of the images. This can lead to improved accuracy and generalization of the model on new and unseen images.

Lastly, data augmentation is another technique that we plan to explore. By applying various transformations to the images, such as rotations or flips, we can artificially increase the size of our dataset and potentially improve the model's ability to generalize to new data.

Overall, there are several exciting directions for future work in this deep learning assignment. By continuing to experiment and iterate, we hope to achieve even better performance on this challenging task.