Artificial Intelligence:

Image Classification using a Convolutional Neural Network and Support Vector Machine

Nathan Hewett

August 2022

Table of Contents

Font Conventions

This report contains the following typographical conventions.

Italic: Used for filenames, pathnames, pseudo code and emphasis.

`Courier:` Used for program code, methods, commands, and database contents.

Google Collab Link:

https://colab.research.google.com/drive/1njoVp-RxlcpvlyWVjODnNblqSH6oE9Vo?usp=sharing

1 Introduction

Chest X-ray (CXR) imaging is a portable, accessible, and rapid form of diagnosing conditions related to the lungs (Jacobi *et al,* 2020, pp.35-37). To classify medical images such as these, various machine learning methods can be utilised, including; *Convolutional Neural Networks* (CNN) and *Support Vector Machines* (SVM). CNNs are similar to other feed forward neural networks, in that they include input, hidden and output layers, comprised of connected neurons with associated weights and biases. However, CNNs also contain convolutional layers which implement a *Kernel* for pattern identification. This acts like a small matrix which slides over an image's total matrix of pixels, to identify features. These networks also employ a *Pooling Layer* to reduce the number of input parameters, thus reducing complexity and risk of overfitting (Goodfellow *et al,* 2016, pp.326-341). These extracted features are then passed to the hidden and output layers to produce a probable class score, with the use of activation functions. This deep learning technique is implemented in numerous CXR imaging studies, yielding efficient results (Haque *et al,* 2020, pp.125-129). Alternatively, SVM separates its classes with the use hyper-planes. The optimal hyper-plane is one that provides the maximum distance between the supporting vectors. This distance is referred to as the margin and is maximised to reduce loss. When dealing with data that needs further separation to be classified, SVMs employ a Kernel. This Kernel differs from the CNN implementation, although the principle is similar; it provides a method to reduce the dimensionality of the input. SVMs utilise the Kernel to transform data into a higher dimensional space, thus enabling data points to be fitted (Chandra and Bedi, 2018, pp.1867-1869). SVMs were successfully utilised by Kumari (2013, pp.1686-1690) in classifying MRI images with 98% accuracy. Chaganti *et al* (2020, pp.1-5), conclude that image classification using a CNN achieved a 0.57% increase in accuracy over SVM models. However, research by Wang, Fan and Wang (2021, p.65-67), observes SVMs to be more efficient than CNNs on smaller datasets. Moreover, Peng *et al* (2020, pp.1-5) state that a CNN accuracy can be improved by 2.98% when using a hybrid SVM-CNN model. The efficiency of this technique is reaffirmed by Ahlawat and Choudhary (2020, p.2554-2558), whereby they achieved image recognition accuracy of 99.28% using a CNN-SVM model.

This report will critically analyse both machine learning methods and examine their performance when conducting binary image classification on Covid-19 patient CXRs. The implemented models are CNN and CNN-SVM, the latter of which will extract image pixels with the use of convolutional layers and classify using an SVM. In an attempt to provide comparable results, both the CNN and CNN-SVM models are constructed sequentially, with similar training and spatial-feature learning hyperparameters. The hyperparameters selected for the final model are a result of numerous tests, discussed throughout the Model Testing section. A further analysis of the final models is discussed in the subsequent section. All image data has been acquired from the Covid-19 Radiography Dataset (Chowdhury *et al,* 2020), examples of which are viewable in Figures *1* & *2.* Both models are created using *Keras,* due to its stability and efficient interface for accessing machine learning libraries (Perrotta, 2020, p.199).

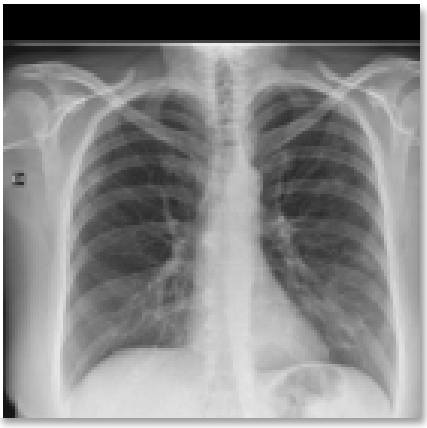


Figure 1: Covid-19 CXR

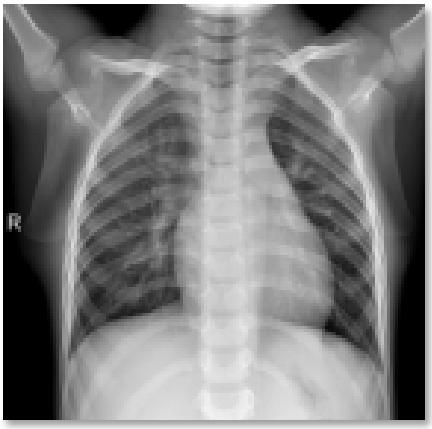


Figure 2: Non Covid-19 CXR

2 Model Testing

2.1 Data Preparation

The training data set for both models has been augmented to enable realistic modifications to the images during training, thus enabling the model to learn from different alterations and identify patterns. Images were transformed horizontally to learn symmetrical variations and zooming-out is included to account for discrepancies in CXR sizing and distancing. To optimise the performance, 20% of the training dataset has been split into a validation subset. This is separate to the test set which calculates the model. Bengio (2012, p.9) states 32 to be a good default batch size. Furthermore, research by Kandel and Castelli (2020, pp.312 - 315) conclude smaller batch sizes with low learning rates produce optimal results. Therefore, both models implement the suggested batch size of 32, with a learning rate of 0.001. This size produced significantly higher accuracy and a lower rate of loss, when compared with larger batch sizes of 64 and 128. These results are viewable in Figure 6.

2.2 Input Layer and Feature Extraction

Simonyan and Zisserman (2015, pp.1 - 14), the creators of the VGG16 CNN, state a small 3x3 Kernel size is optimal for their model. When evaluated alongside other leading CNN models for CXR classification, VGG16 often achieves the highest accuracy (Makris *et al,* 2020, pp.63 - 64). However, research by Tan and Le (2019, pp.1 - 10) observes that iteratively increasing the Kernel size at each convolutional layer enables the retrieval of greater information from neighbouring pixels, thus improving accuracy. When comparing the performance of 3x3 and 5x5 Kernel sizes during testing, the loss remained comparable. However, a Kernel size of 5x5 with a stride of 1x1 achieved the greatest accuracy, with an increase of between 0.0151 and 0.0234. Therefore, these are the selected parameters.

To make up for overlaps from the Kernel size not fitting with the input size, *Same* padding has been applied to append null values to the outer frame and prevent data loss. No testing of the model was performed without padding, therefore future work may benefit by replacing this hyperparameter with *Valid* for comparison. However, Valid produced significantly worse performance when compared with Same padding during research by Wiranata *et al* (2018, pp.211 - 212) and Chen *et al* (2020, pp.20 - 25). In these studies, they conclude that padding prevents information loss, due to its ability to emphasise characteristics of an image for optimal extraction.

The selected activation method used on the convolutional layers, is *Rectified Linear Unit* (ReLU). This provides a maximum value between zero and the activation input, thus ensuring a negative number cannot be returned. This is computationally efficient and provides faster convergence. Furthermore, unlike the *Sigmoid* activation function, ReLU removes the problem of vanishing gradients, thus ensuring weights and biases do not receive small updates. However, it is possible for ReLU to suffer from *Dying ReLU,* in which weights in the network provide negative inputs into ReLU neurons, thus contributing nothing towards backpropagation (Feng and Lu, 2019, pp. 1 - 5). In an attempt to remove this risk and further optimise the model, the activation functions were replaced with *Exponential Linear Unit* (ELU), which can produce negative outputs and remove the Dying ReLU problem. However, as displayed in Figure 3; this produced poorer results during testing and is not used in the final model. The formulae of Sigmoid, ReLu and ELU are demonstrated in Appendix: A.

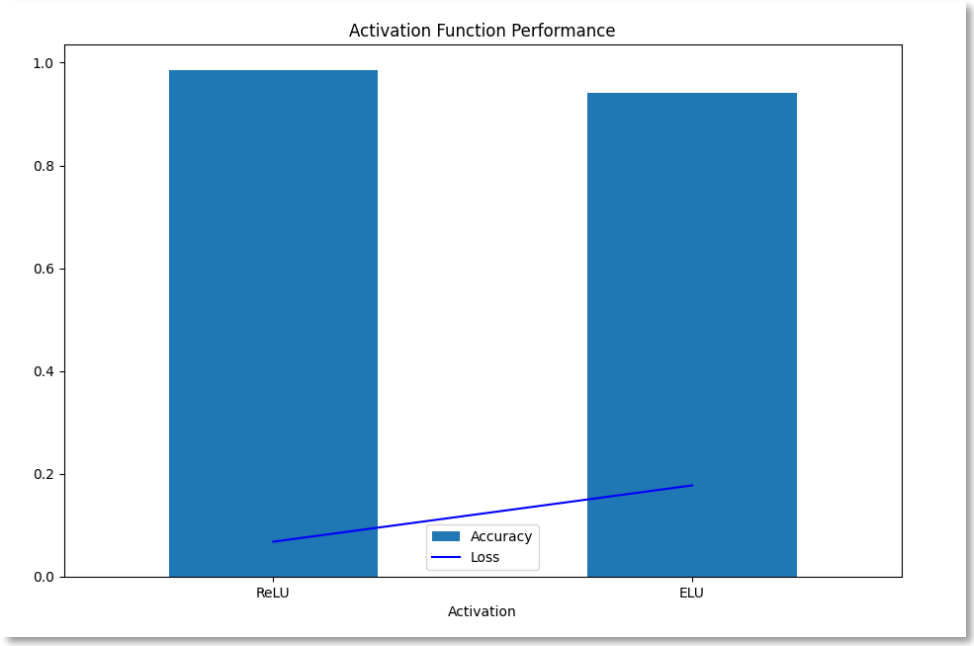


Figure 3: Comparison of ReLu and ELU Performance

The Pooling Layer method used in the model is *Max Pooling.* This reduces the image size and produces a maximum value within the current Kernel. Thus, reducing the number of parameters to avoid overfitting. When combining Max Pooling with the convolutional layer, *translation invariance* is achieved. Therefore, enabling pattern detection throughout the image matrix (Goodfellow *et al,* 2016, pp.326 - 357). *Average Pooling* was not used in this model, as when combined with ReLU it can reduce the weight from strong activations, due to acknowledgement of too many zero activations. Other Pooling methods such as *Hybrid Pooling* aim to provide further optimisations and enable a CNN to increase its generalisation ability (Tong and Tanaka, 2019, p.77). Yin *et al* (2019, pp.1 - 6) propose replacing a convolutional and pooling layer with *UNI-Dimensional, Recurrent Neural Network* (RNN) layers to achieve greater accuracy. This method is complex and includes a heightened risk of overfitting, thus is not implemented in this model. However, it should be considered in future advancements. As visualised in Figure 6, additional convolutional layers were tested for improvements. However, this produced significantly lower performance, presumably due to the model becoming overfitted.

Each *Dropout* layer in the model is set to 50%, thus setting a random half of neurons to zero. This teaches subsequent layers to not rely on specific characteristics and to utilise all available information, thus further reducing overfitting (Makris *et al,* 2020, p.63). Testing was conducted with decreasing Dropout percentages; however, this provided a reduction in accuracy. A second convolutional layer with identical Max Pooling and Dropout parameters is included in the model, with the Kernel number increased to 64. This is to enable capturing of further patterns as the combinations become more complex. Ioffe and Szegedy (2015, pp.7 - 8) state that the inclusion of *Batch Normalisation* can improve results of image classification and remove the risk of vanishing gradients. Models were evaluated with Batch Normalisation; however, this produced a reduction in accuracy for this model. Klambauer *et al* (2017, pp.1 - 9) conclude that Batch Normalisation could also be attained using the *Scaled Exponential Linear-Unit* (SeLU) activation function, thus creating *Self-Normalizing Neural Networks.*

2.3 Hidden Layers and Classification

The extracted feature maps are flattened into a vector to be inserted into the network for processing. The subsequent fully connected layer receives the preceding output and applies the ReLU activation function. The *Universal Approximation Theorem* establishes; a network containing one hidden layer can perform any continuous function, albeit that layer may become impractical and fail to learn and generalise correctly. Research by Shabbeer *et al* (2020, pp.112 - 118) analyses the effects in performance of varying numbers of hidden layers. The study concludes that optimal performance for shallow CNNs is obtained by including more neurons, and that deeper CNNs should include less neurons. Geron (2019, pp.323 - 325) states that deep networks provide better results for complex challenges. This statement is reiterated by Ba and Caruana (2014, pp.1 - 8), stating; accuracy can be increased by 5% with the inclusion of two additional hidden layers. The same research provides empirical evidence that shallow CNNs can achieve performances comparable to deeper networks through model mimicking. When attempting to optimise the baseline model, additional hidden layers were implemented with increasing nodes to assess any improvements in accuracy. However, as seen in Figure 6; this produced a minor variance in comparison to the original model. As such, the final model contains only one hidden layer. Moving forward, when attempting to optimise future models, more experimentation should be performed in regard to the number of hidden layers and neurons.

As previously stated, these models are created for binary classification; thus, linear functions are selected in the output layers. The CNNs activation function is Sigmoid. This logistic function will always output a value between 0-1, therefore determining the probability. However, when using a SVM for binary classification, *Linear SVM* is used to create a hyperplane between the two categories. As such, the CNN-SVM model implements this with the use of a linear Kernel. Ahlawat and Choudhary (2019, pp.2554 - 2558) argue that the ability of a SVM to maximise the distance of the decision boundary with a hyperplane, makes this method optimal for binary classification and produce better performance than a traditional CNN. A summary of the final model is displayed in Figure: 4.

Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 150, 150, 32)	2432
max_pooling2d_10 (MaxPooling2D)	(None, 75, 75, 32)	0
dropout_15 (Dropout)	(None, 75, 75, 32)	0
conv2d_11 (Conv2D)	(None, 75, 75, 64)	51264
max_pooling2d_11 (MaxPooling2D)	(None, 37, 37, 64)	0
dropout_16 (Dropout)	(None, 37, 37, 64)	0
flatten_5 (Flatten)	(None, 87616)	0
dense_10 (Dense)	(None, 256)	22429952
dropout_17 (Dropout)	(None, 256)	0
dense_11 (Dense)	(None, 1)	257

Total params: 22,483,905
Trainable params: 22,483,905
Non-trainable params: 0

Figure 4: Model Summary

2.4 Backpropagation and Gradient Descent

To enable *Backpropagation*, an effective loss function must be applied. For the CNN model; *Binary cross entropy* is selected, as this *logarithmic loss* function is often used to determine errors for binary classifications. This loss function was employed for CNN image classification by Ruby *et al* (2020, pp.5393 - 5396), when achieving 95.62% accuracy in training. The CNN-SVM model employs the L2-regularised *Hinge* loss function, thus providing a SVM model classification. (Geron, 2019, pp.172 - 173). When selecting a method for gradient descent, *Adaptive Gradient Methods* (AGM) ensure faster convergence, compared with *Stochastic Gradient Descent* (SGD) (Ruder, 2017, p.10). Furthermore, Keskar and Socher (2017, pp.1 - 7) claim AGMs are insensitive to hyperparameters and provide accelerated initial progress. However, they are outperformed in later phases and generalise poorly in comparison to SGD. Therefore, they suggest a hybrid strategy of switching between the two aforementioned optimiser methods at specific epochs to yield better performance. This hybrid strategy was not performed. However, SGD, and the AGM methods; *Adaptive Moment Estimation* (Adam), *Nesterov-accelerated Adaptive Moment Estimation* (Nadam), and Adam variant; AdaMax, were all assessed to determine which is most efficient for this model. Moreover, to ensure the most accurate model is obtained and to prevent overfitting, early stopping has been included during training. This ensures that if the performance begins to decrease over subsequent epochs, the training is stopped, and the optimal model is saved. Demonstrated in Figure: 5 is the performance of each optimiser on this model. Adam produced the greatest accuracy, with an increase of 0.0089, 0.0544 and 0.0771, compared to Nadam, AdaMax and SGD, respectively. Likewise, Adam produced a reduction in loss of 0.0173, 0.1034 and 0.1398.

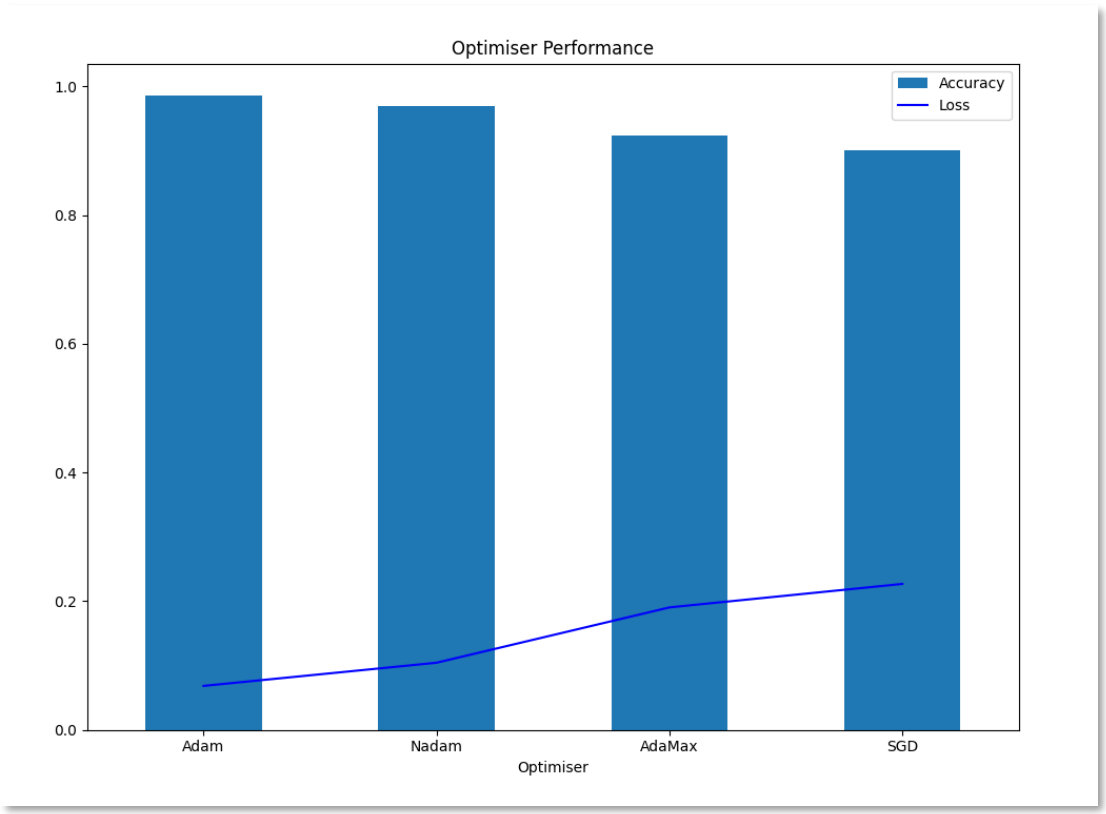


Figure 5: Evaluation of Optimisers

2.5 Test Model Analysis

	Hyperparameter Modification	Accuracy	Loss
Model A	2 conv layers, 3 hidden layers (Adam optimiser)	0.9731	0.1080
Model B	5 conv layers, 1 hidden layer (Adam optimiser)	0.9008	0.3209
Model C	2 conv layers, 1 hidden layer (Adam optimiser)	0.9779	0.0869
Model D	Model C with 3x3 kernel & decrease dropout (0.40)(Adam optimiser)	0.9545	0.1056
Model E	Model C with 3x3 kernel & 1 layer: ELU (Adam optimiser)	0.9628	0.0827
Model F	Model C with batch size: 64 (Adam optimiser)	0.9276	0.2120
Model G	Model C with batch size: 128 & batch normalisation (Adam optimiser)	0.9045	0.2221
Model H	Model C with batch normalisation (Adam optimiser)	0.7210	0.5370
Model I	Model C with all layers replaced with ELU (Adam optimiser)	0.9421	0.1777
Model J	Model C with AdaMax optimiser	0.9235	0.1903
Model K	Model C with Nadam optimiser	0.9690	0.1042
Model L	Model C with SGD optimiser	0.9008	0.2267

Figure 6: Model Testing Data

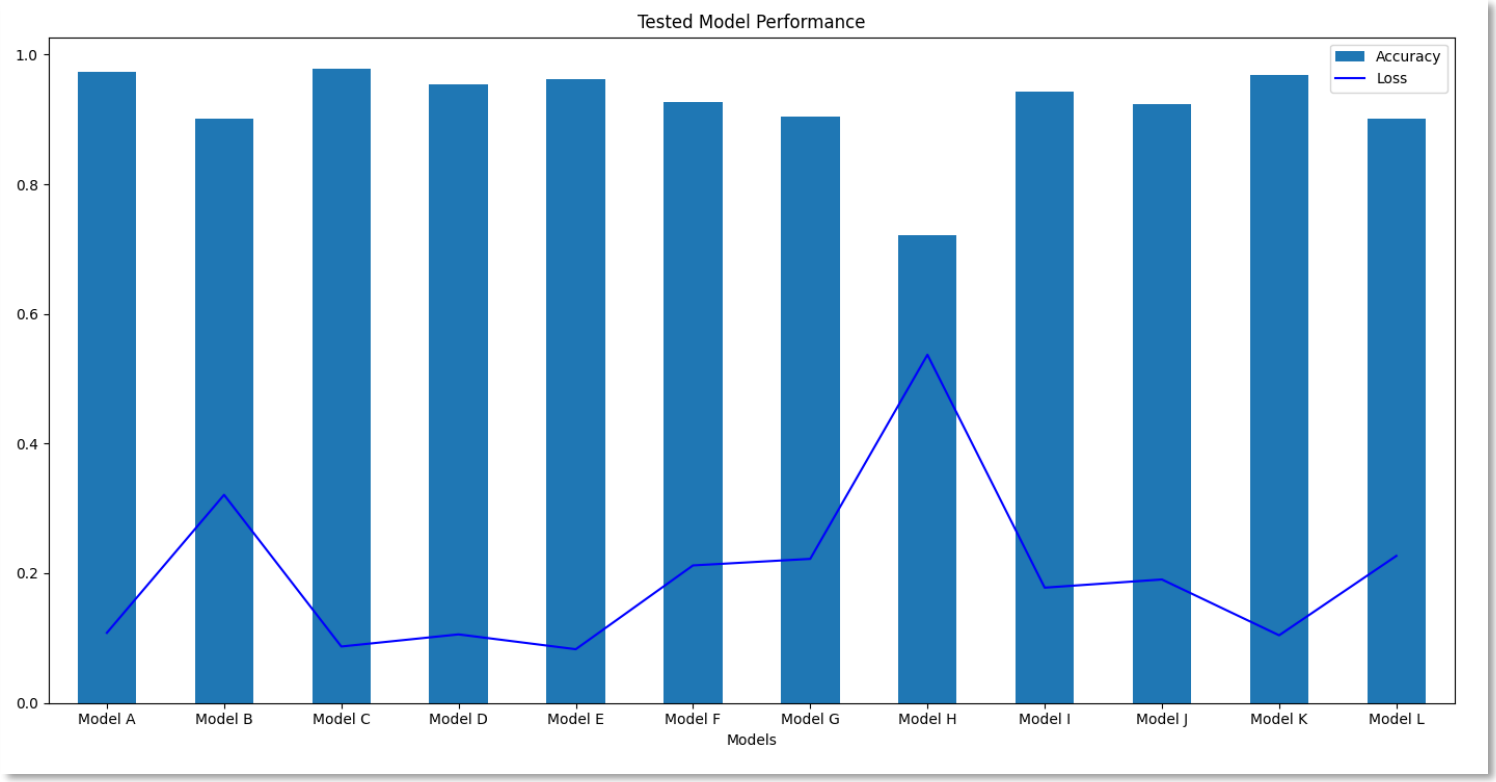


Figure 7: Accuracy and Loss of Each Tested Model

3 CNN & CNN-SVM Analysis

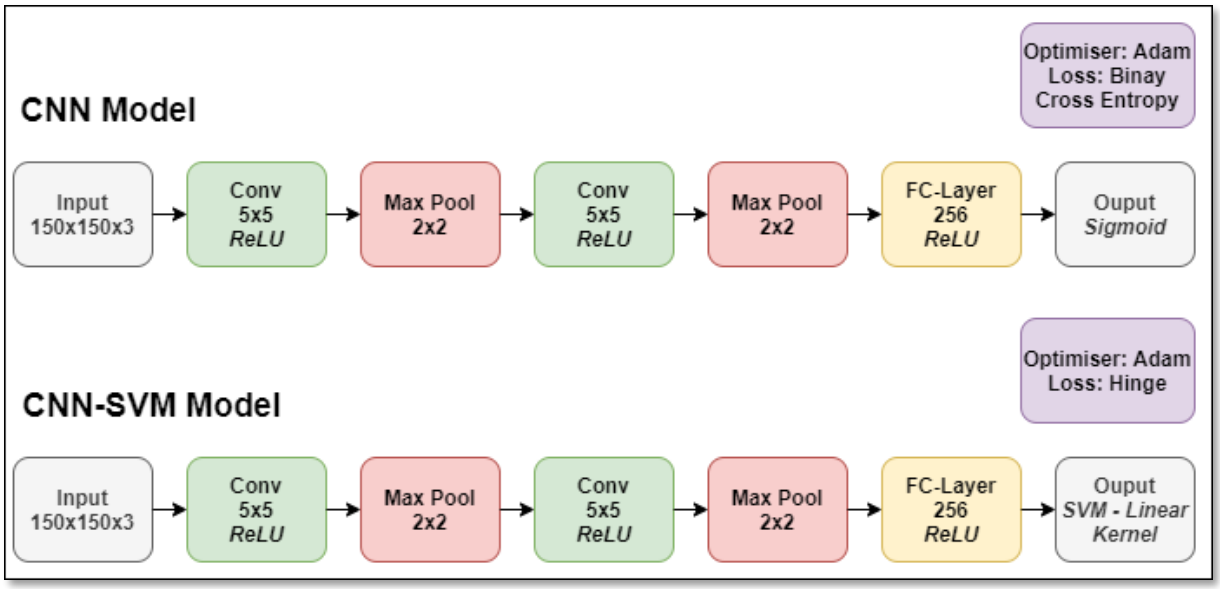


Figure 8: Diagram of CNN & CNN-SVM Models

A diagram displaying the layers of each model can be viewed in Figure: 8. All testing models were trained over 10 epochs. However, to provide further opportunity for improvement, the final CNN and CNN-SVM models were trained at 30 epochs each. Due to the inclusion of early stopping, the CNN model completed at 25 epochs. As displayed in Figures 9 and 10, both models perform comparatively, with loss reducing and accuracy increasing as epochs progress. The CNN model achieves an accuracy rate of 0.9862 and loss of 0.0682. The CNN-SVM achieved a lower accuracy of 0.9710, with and an increased loss of 0.1294. However, when disregarding the results provided by the early stopping intervention and observe the performance of the final epochs, the CNN achieves an increase of 0.0034 and similar loss reduction of 0.0606. Thus, in each instance the CNN model outperforms the CNN-SVM for CXR image classification, albeit with a narrow divergence in accuracy. These findings produce both resemblances and variances to the previously mentioned studies. This is presumably a result of the extensive factors involved, including hyperparameters and layer implementations. Furthermore, the performance of an image classification rests on other factors, such as the quality of the image dataset (Chandra and Bedi, 2021, p.1875). An example of the model being used to correctly identify CXR images is displayed in Figure 9. For further data relating to the tested models, see Appendix B.

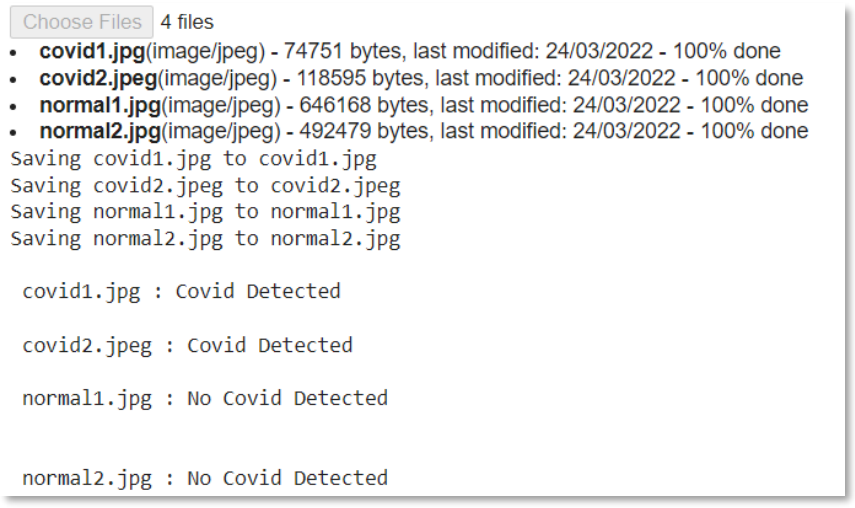


Figure 9: Test Case Example

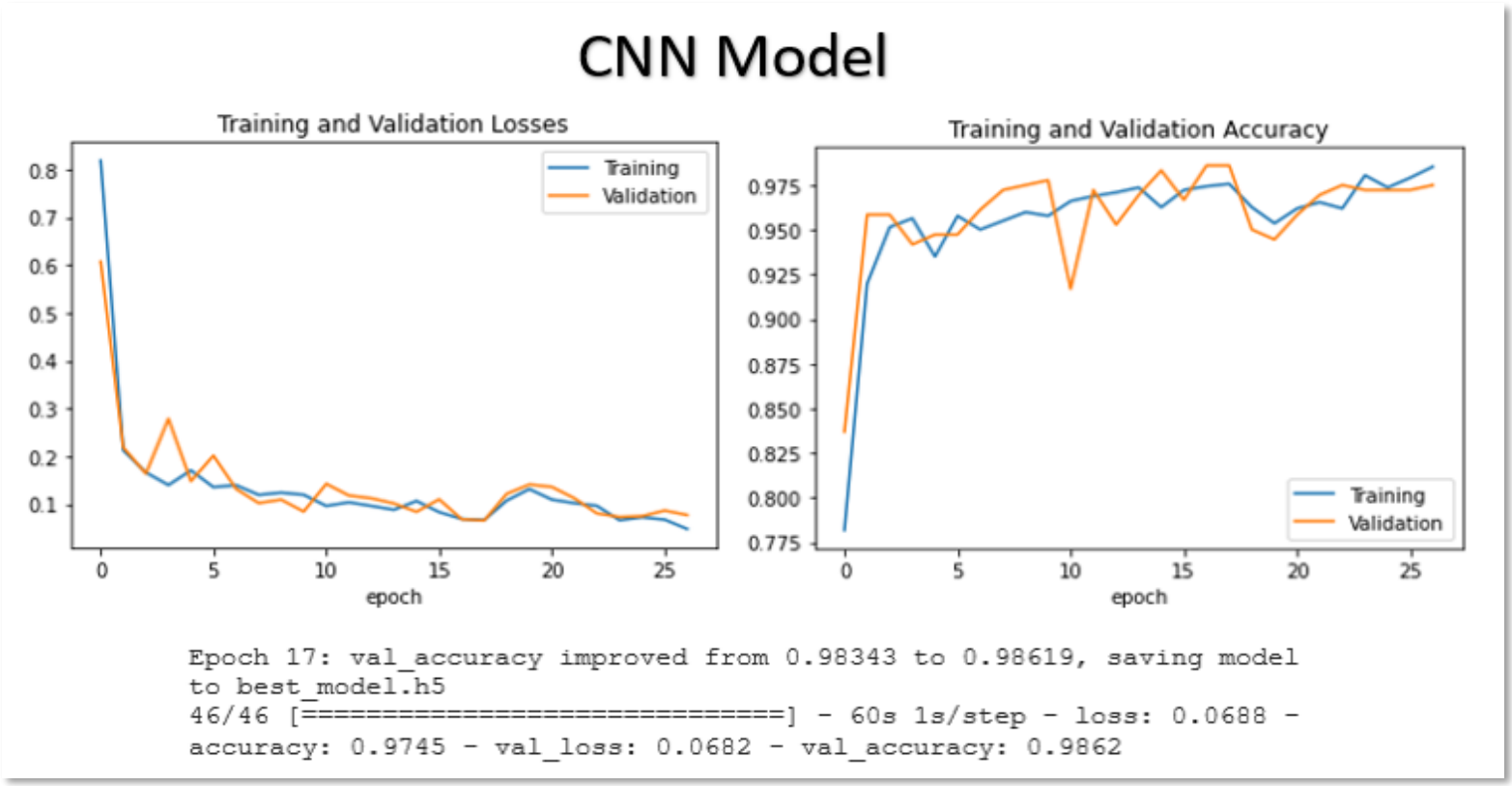


Figure 10: CNN Performance

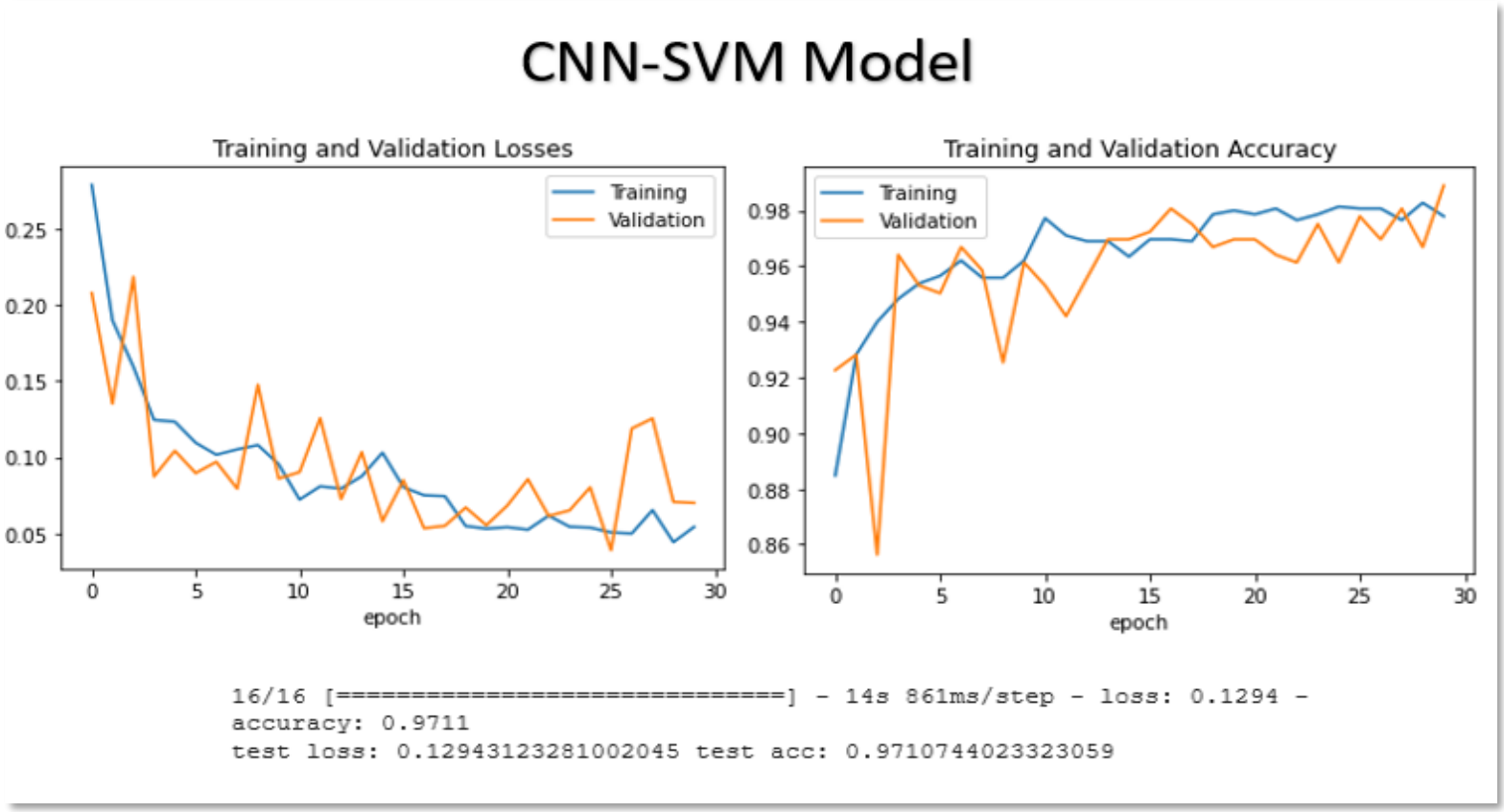


Figure 11: CNN-SVM Performance

4 Conclusion

Referenced throughout this report are examples and opportunities to further optimise the performance of the models, such as examining; hybrid pooling, SeLU activation functions, hybrid Adam-SGD optimisation, and hidden layer expansion. Furthermore, notable CNN models including VGG16 and ResNet50 should also be explored. As observed in the analysis, the CNN model achieves a reduced loss and a minor increase in accuracy, in comparison to the CNN-SVM. However, both models provide effective binary classification for Covid-19 CXR images. Moving forward, further improvement to the scope of the model should be considered, including the provision of multi-class classification for other chest conditions, including Pneumonia. Moreover, greater advancements may be achieved through gathering and exploring CXR images in varying phases of a condition, for example; contraction to recovery.

5 References

Ahlawat, S., and Choudhary, A. (2020) 'Hybrid CNN-SVM classifier for handwritten digit recognition'. *International Conference on Computational Intelligence and Data Science,* 167, pp. 2554 – 2560, doi:10.1016/j.procs.2020.03.309

Ba, J., and Caruana, R. (2014) 'Do deep nets really need to be deep'. *ArXiv Machine Learning,* pp. 1 -10, doi:10.48550/arXiv.1312.6184

Bengio, Y. (2012) 'Practical recommendations for gradient-based training of deep architectures'. *ArXiv Machine Learning*, pp. 1 - 33, doi:10.48550/arXiv.1206.5533

Chaganti, S., *et al.* (2020) 'Image classification using SVM and CNN' . *International Conference on Computer Science, Engineering and Applications,* pp. 1 – 5, doi:10.1109/ICCSEA49143.2020.9132851

Chandra, M., and Bedi, S. (2018) 'Survey on SVM and their application in image classification'. *International Journal of Information Technology,* 13, pp. 1867 – 1877, doi:10.1007/s41870-017-0080-1

Chen, Y., *et al.* (2020) 'Spatial predictions of debris flow susceptibility mapping using convolutional neural networks in Jilin Province, China'. *MDPI Water,* 12(8), doi:10.3390/w12082079

Chowdhury, M. E. H., *et al.* (2020) 'Can AI help in screening viral and covid-19 pneumonia' *Institute of Electrical and Electronics Engineers,* 8, pp. 132665 – 132676, Available at: COVID-19 Radiography Database

Feng, J., and Lu, S. (2019) 'Performance analysis of various activation functions in artificial neural networks'. *Journal of Physics: Conference Series,* 1237(2), doi:10.1088/1742-6596/1237/2/022030

Geron, A. (2019) *Hands-on machine learning with Scikit-Learn, Keras, and Tensorflow.* 2nd edn. Sebastopol: O'Reilly Media Inc, pp. 172 - 325

Goodfellow, I., *et al.* (2016) *Deep learning.* Cambridge: MIT Press, pp. 326 – 366.

Ioffe, S., and Szegedy, C. (2015) 'Batch normalization: Accelerating deep network training by reducing internal covariate shift'. *ArXiv Computer Science,* pp. 1 – 11. Available at: 1502.03167.pdf (arxiv.org) (Accessed: 30 March 2022)

Jacobi, A., *et al.* (2020) 'Portable chest X-ray in coronavirus disease-19 (Covid-19): A pictorial review'. *Clinical Imaging,* 64, pp. 35 – 42, doi:10.1016/j.clinimag.2020.04.001

Kandel, I., and Castelli, M. (2020) 'The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset'. *ICT Express,* 6(4), pp. 312 – 315, doi:10.1016/j.icte.2020.04.010

Keskar, N., and Socher, R. (2017) 'Improving generalization performance by switching from Adam to SGD'. *ArXiv Computer Science,* pp. 1 – 10, Available at: Improving Generalization Performance by Switching from Adam to SGD (arxiv.org) (Accessed: 30 March 2022)

Klambauer, G., *et al.* (2017) 'Self-Normalizing Neural Networks'. *ArXiv Machine Learning,* pp. 1 – 102, doi:10.48550/arXiv.1706.02515

Kumari, R. (2013) 'SVM classification: An approach on detecting abnormality in brain MRI images' *International Journal of Engineering Research and Applications,* 3(4), pp. 168 – 1690, ISSN: 2248-9622. Available at: JI3416861690-with-cover-page-v2.pdf (Accessed: 29 March 2020)

Makris, A., *et al.* (2020) 'Covid-19 detection from chest X-ray images using deep learning and convolutional neural networks'. *medRxiv Computer Science,* pp. 60 – 66, doi:10.1101/2020.05.22.20110817

Peng, Y., *et al.* (2020) 'CNN-SVM: A classification method for fruit fly image with complex background'. *IET Cyber-Physical Systems: Theory & Applications,* 5(4), pp. 1 -6, doi:10.1049/iet-cps.2019.0069

Perrotta, P. (2020) *Programming machine learning*: *From coding to deep learning.* The Pragmatic Programmers, LLC, pp.100 - 200

Ruby, A., *et al.* (2020) 'Binary cross entropy with deep learning technique for image classification'. *International Journal of Advanced Trends in Computer Science and Engineering,* 9(4), pp. 5393 – 5397, doi:10.30534/ijatcse/2020/175942020

Ruder, S. (2017) 'An overview of gradient descent optimization algorithms'. *ArXiv Machine Learning,* pp. 1 – 14, doi:10.48550/arXiv.1609.04747

Simonyan, K., and Zisserman, A. (2015) 'Very deep convolutional networks for large-scale image recognition'. *ArXiv Computer Vision and Pattern Recognition,* pp. 1 – 14, doi:10.48550/arXiv.1409.1556

Shabbeer B., *et al.* (2020) 'Impact of fully connected layers on performance of convolutional neural networks for image classification'. *Neurocomputing,* 378, pp. 112 – 119, doi:10.1016/j.neucom.2019.10.008

Tan, M., and Le, Q. (2019) 'MixConv: Mixed depthwise convolutional kernels'. *arXiv Computer Science,* pp. 1 – 13. Available at: 1907.09595.pdf (arxiv.org) (Accessed: 29 March 2022)

Tong, Z., and Tanaka, G. (2019) 'Hybrid pooling for enhancement of generalisation ability in deep convolutional networks'. *Neurocomputing,* 333, pp. 76 – 85, doi:10.1016/j.neucom.2018.12.036

Wang, P., Fan,. E., and Wang, P. (2021) 'Comparative analysis of image classification algorithms based on machine learning'. *Pattern Recognition Letters,* 41, pp. 61 – 67, doi:10.1016/j.patrec.2020.07.042

Wiranata, A., *et al.* (2018) 'Investigation of padding schemes for faster R-CNN on vehicle detection' *International Conference on Control, Electronics, Renewable Energy and Communications,* pp. 208 – 212, doi:10.1109/ICCEREC.2018.8712086

Yin *et al* (2019) 'CNN and RNN mixed model for image classification' *MATEC Web of Conferences,* 277(4), pp. 1 – 7, doi:10.1051/matecconf/201927702001

6 Appendix A: Activation Function Formulae

Sigmoid:

$$f(x_i)=\frac{e^{x_i}}{1+e^{x_i}}=\frac{1}{1+e^{-x_i}}, \qquad f'(x_i)=\frac{e^{-x_i}}{\left(1+e^{x_i}\right)^2}$$

ReLU:

$$f(x_i)=\max(0,x_i)=\begin{cases}x_i, x_i>0\\0, x_i<0\end{cases}, \qquad f'(x_i)=\begin{cases}1, x_i>0\\0, x_i<0\end{cases}$$

ELU:

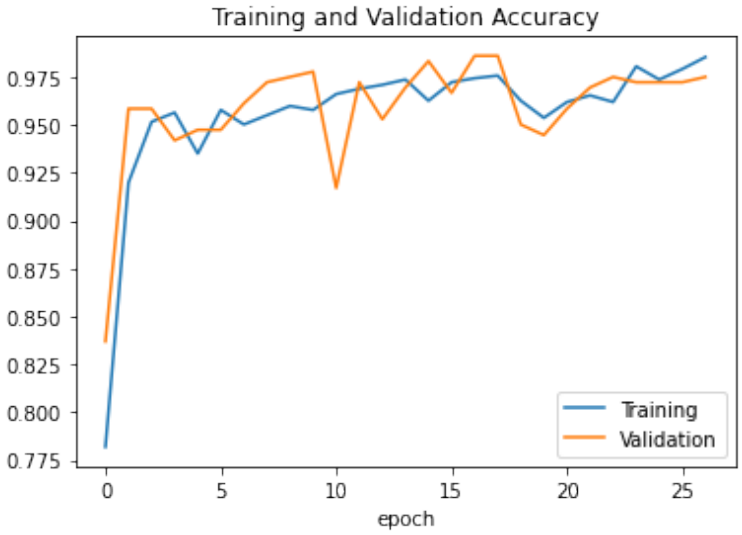
$$f(x_i)=\begin{cases}x_i, & x_i>0\\ \alpha_i\left(e^{x_i}-1\right), & x_i\le 0\end{cases}, \qquad f'(x_i)=\begin{cases}1, & x_i>0\\ f(x_i)+\alpha_i, & x_i\le 0\end{cases}$$

7 Appendix B: Tested Models

7.1 CNN Model

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 150, 150, 32)      2432

 max_pooling2d_10 (MaxPoolin  (None, 75, 75, 32)       0
 g2D)

 dropout_15 (Dropout)        (None, 75, 75, 32)        0

 conv2d_11 (Conv2D)          (None, 75, 75, 64)        51264

 max_pooling2d_11 (MaxPoolin  (None, 37, 37, 64)       0
 g2D)

 dropout_16 (Dropout)        (None, 37, 37, 64)        0

 flatten_5 (Flatten)         (None, 87616)             0

 dense_10 (Dense)            (None, 256)               22429952

 dropout_17 (Dropout)        (None, 256)               0

 dense_11 (Dense)            (None, 1)                 257

=================================================================
Total params: 22,483,905
Trainable params: 22,483,905
Non-trainable params: 0
```

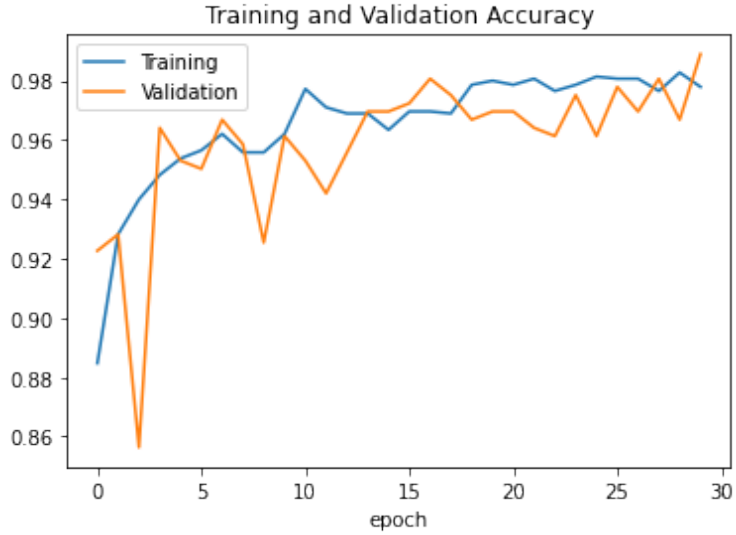
```
Epoch 17: val_accuracy improved from 0.98343 to 0.98619, saving model
to best_model.h5
46/46 [==============================] - 60s 1s/step - loss: 0.0688 -
accuracy: 0.9745 - val_loss: 0.0682 - val_accuracy: 0.9862
```



7.2 CNN-SVM Model

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 150, 150, 32)      2432

 max_pooling2d_10 (MaxPoolin  (None, 75, 75, 32)       0
 g2D)

 dropout_15 (Dropout)        (None, 75, 75, 32)        0

 conv2d_11 (Conv2D)          (None, 75, 75, 64)        51264

 max_pooling2d_11 (MaxPoolin  (None, 37, 37, 64)       0
 g2D)

 dropout_16 (Dropout)        (None, 37, 37, 64)        0

 flatten_5 (Flatten)         (None, 87616)             0

 dense_10 (Dense)            (None, 256)               22429952

 dropout_17 (Dropout)        (None, 256)               0

 dense_11 (Dense)            (None, 1)                 257

=================================================================
Total params: 22,483,905
Trainable params: 22,483,905
Non-trainable params: 0
```

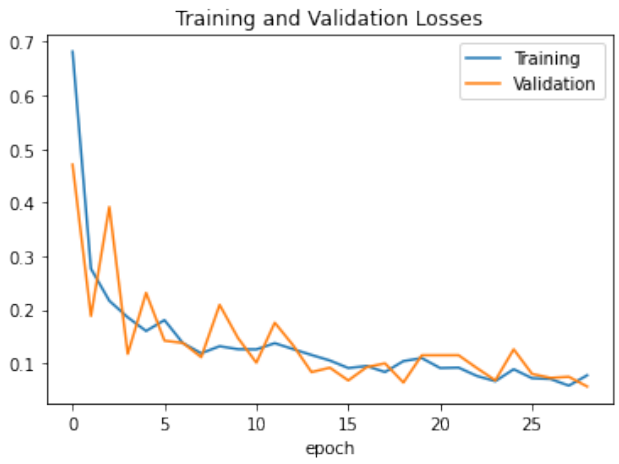
```
16/16 [==============================] - 14s 861ms/step - loss: 0.1294 -
accuracy: 0.9711
test loss: 0.12943123281002045 test acc: 0.9710744023323059
```

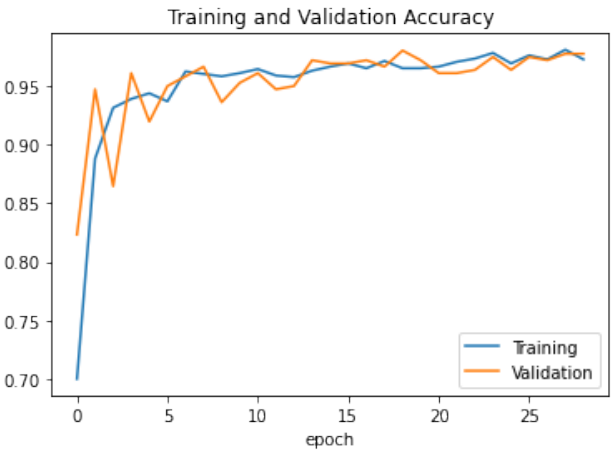


7.3 Test Model A: Two Convolutional Layers and Three Hidden Layers

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 150, 150, 32)      2432

 max_pooling2d (MaxPooling2D  (None, 75, 75, 32)       0
 )

 dropout (Dropout)           (None, 75, 75, 32)        0

 conv2d_1 (Conv2D)           (None, 75, 75, 64)        51264

 max_pooling2d_1 (MaxPooling  (None, 37, 37, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 37, 37, 64)        0

 flatten (Flatten)           (None, 87616)             0

 dense (Dense)               (None, 256)               22429952

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 512)               131584

 dropout_3 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 1024)              525312

 dropout_4 (Dropout)         (None, 1024)              0

 dense_3 (Dense)             (None, 1)                 1025

=================================================================
Total params: 23,141,569
Trainable params: 23,141,569
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 29: val_accuracy did not improve from 0.98066
46/46 [==============================] - 57s 1s/step - loss: 0.0778 -
accuracy: 0.9731 - val_loss: 0.0569 - val_accuracy: 0.9779
Epoch 29: early stopping
```

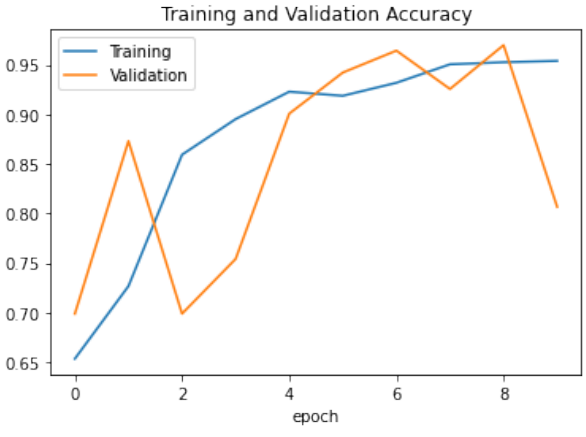




7.4 Test Model B: Five Convolutional Layers

```
conv2d_4 (Conv2D)            (None, 150, 150, 16)      1216

max_pooling2d_4 (MaxPooling  (None, 75, 75, 16)        0
2D)

dropout_6 (Dropout)          (None, 75, 75, 16)        0

conv2d_5 (Conv2D)            (None, 75, 75, 32)        12832

max_pooling2d_5 (MaxPooling  (None, 37, 37, 32)        0
2D)

dropout_7 (Dropout)          (None, 37, 37, 32)        0

conv2d_6 (Conv2D)            (None, 37, 37, 64)        51264

max_pooling2d_6 (MaxPooling  (None, 18, 18, 64)        0
2D)

dropout_8 (Dropout)          (None, 18, 18, 64)        0

conv2d_7 (Conv2D)            (None, 18, 18, 128)       204928

max_pooling2d_7 (MaxPooling  (None, 9, 9, 128)         0
2D)

dropout_9 (Dropout)          (None, 9, 9, 128)         0

conv2d_8 (Conv2D)            (None, 9, 9, 256)         819456

max_pooling2d_8 (MaxPooling  (None, 4, 4, 256)         0
2D)

dropout_10 (Dropout)         (None, 4, 4, 256)         0

flatten_2 (Flatten)          (None, 4096)              0

dense_4 (Dense)              (None, 256)               1048832

dropout_11 (Dropout)         (None, 256)               0

dense_5 (Dense)              (None, 1)                 257

=================================================================
Total params: 2,138,785
Trainable params: 2,138,785
Non-trainable params: 0
```

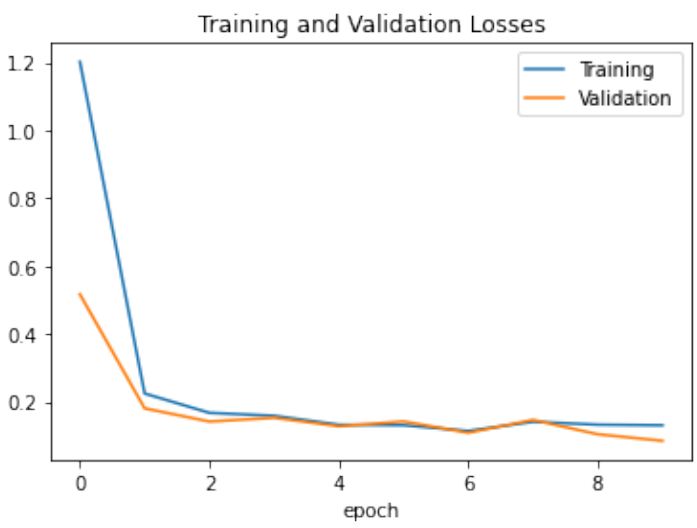
```
16/16 [==============================] - 20s 1s/step - loss: 0.3210 -
accuracy: 0.9008
test loss: 0.32096055150032043 test acc: 0.9008264541625977
```

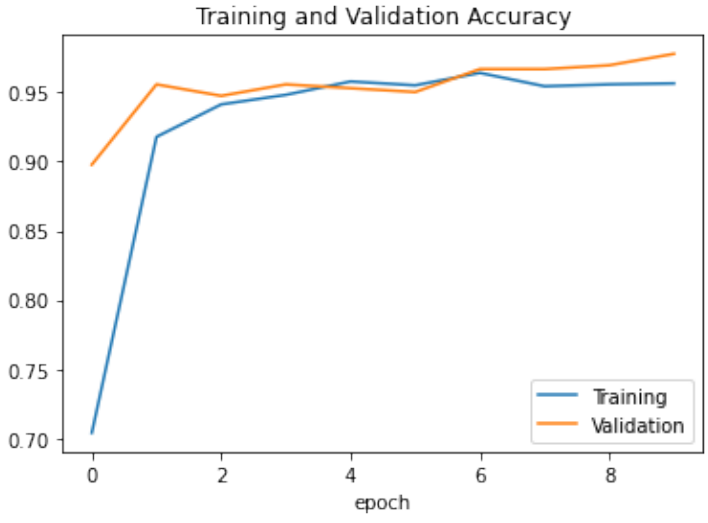


7.5 Test Model C : Two Convolutional Layers, One Hidden Layer

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 150, 150, 32)      2432

 max_pooling2d (MaxPooling2D  (None, 75, 75, 32)       0
 )

 dropout (Dropout)           (None, 75, 75, 32)        0

 conv2d_1 (Conv2D)           (None, 75, 75, 64)        51264

 max_pooling2d_1 (MaxPooling  (None, 37, 37, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 37, 37, 64)        0

 flatten (Flatten)           (None, 87616)             0

 dense (Dense)               (None, 256)               22429952

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 22,483,905
Trainable params: 22,483,905
Non-trainable params: 0
```

```
16/16 [==============================] - 25s 2s/step - loss: 0.1081 -
accuracy: 0.9731
test loss: 0.10807916522026062 test acc: 0.9731404781341553
```

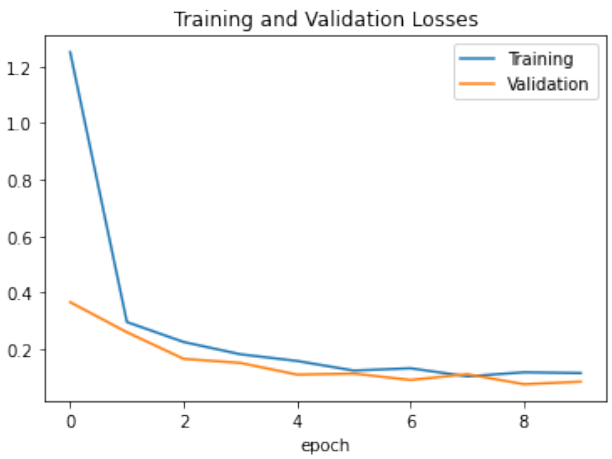


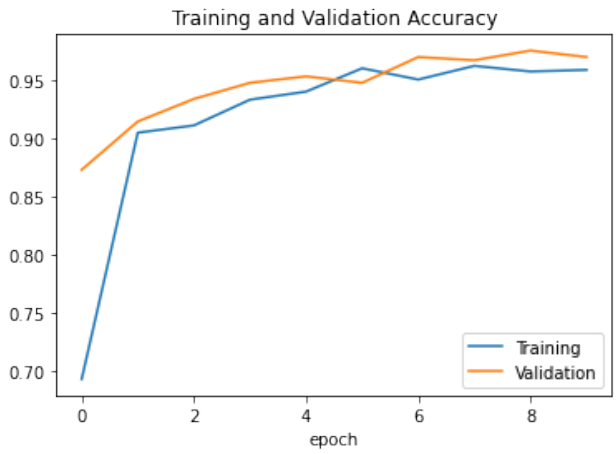


7.6 Test Model D: Model C Variation (3x3 Kernel, Dropout Decrease)

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_26 (Conv2D)          (None, 150, 150, 32)      896

 max_pooling2d_26 (MaxPoolin  (None, 75, 75, 32)       0
 g2D)

 dropout_39 (Dropout)        (None, 75, 75, 32)        0

 conv2d_27 (Conv2D)          (None, 75, 75, 64)        51264

 max_pooling2d_27 (MaxPoolin  (None, 37, 37, 64)       0
 g2D)

 dropout_40 (Dropout)        (None, 37, 37, 64)        0

 flatten_12 (Flatten)        (None, 87616)             0

 dense_25 (Dense)            (None, 256)               22429952

 dropout_41 (Dropout)        (None, 256)               0

 dense_26 (Dense)            (None, 1)                 257

=================================================================
Total params: 22,482,369
Trainable params: 22,482,369
Non-trainable params: 0
_________________________________________________________________
```

```
16/16 [==============================] - 21s 1s/step - loss: 0.1056 -
accuracy: 0.9545
test loss: 0.10560980439186096 test acc: 0.9545454382896423
```

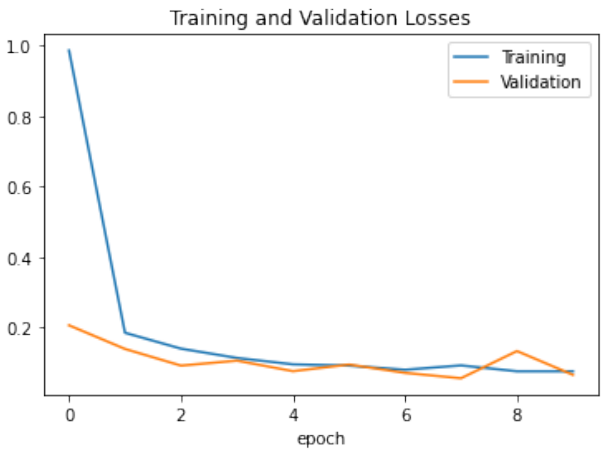


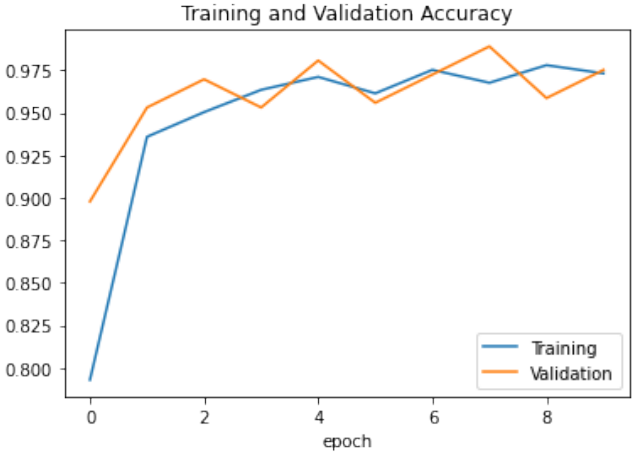


7.7 Test Model E: Model C Variation (3x3 Kernel & ELU)

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_24 (Conv2D)          (None, 150, 150, 32)      896

 max_pooling2d_24 (MaxPoolin  (None, 75, 75, 32)       0
 g2D)

 dropout_36 (Dropout)        (None, 75, 75, 32)        0

 conv2d_25 (Conv2D)          (None, 75, 75, 64)        51264

 max_pooling2d_25 (MaxPoolin  (None, 37, 37, 64)       0
 g2D)

 dropout_37 (Dropout)        (None, 37, 37, 64)        0

 flatten_11 (Flatten)        (None, 87616)             0

 dense_23 (Dense)            (None, 256)               22429952

 dropout_38 (Dropout)        (None, 256)               0

 dense_24 (Dense)            (None, 1)                 257

=================================================================
Total params: 22,482,369
Trainable params: 22,482,369
Non-trainable params: 0
```

```
16/16 [==============================] - 21s 1s/step - loss: 0.0828 -
accuracy: 0.9628
test loss: 0.08275507390499115 test acc: 0.9628099203109741
```

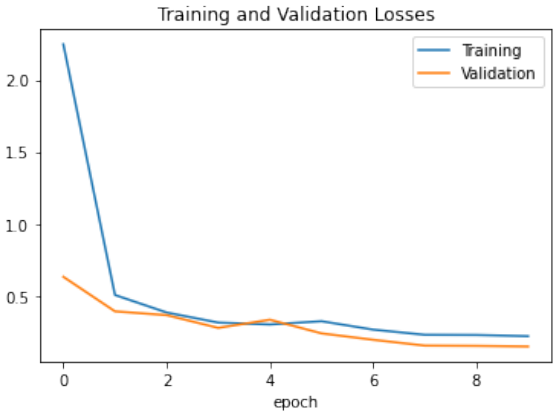


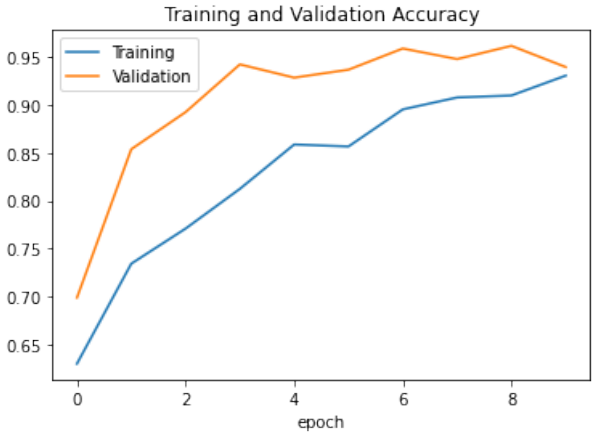


7.8 Test Model F: Model C Variation (Batch Size 64)

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_28 (Conv2D)          (None, 150, 150, 32)      2432

 max_pooling2d_28 (MaxPoolin  (None, 75, 75, 32)       0
 g2D)

 dropout_42 (Dropout)        (None, 75, 75, 32)        0

 conv2d_29 (Conv2D)          (None, 75, 75, 64)        18496

 max_pooling2d_29 (MaxPoolin  (None, 37, 37, 64)       0
 g2D)

 dropout_43 (Dropout)        (None, 37, 37, 64)        0

 flatten_13 (Flatten)        (None, 87616)             0

 dense_27 (Dense)            (None, 256)               22429952

 dropout_44 (Dropout)        (None, 256)               0

 dense_28 (Dense)            (None, 1)                 257

=================================================================
Total params: 22,451,137
Trainable params: 22,451,137
Non-trainable params: 0
```

```
8/8
[==============================] - 19s 2s/step - loss: 0.2121 - accuracy:
0.9277
test loss: 0.21205829083919525 test acc: 0.9276859760284424
```

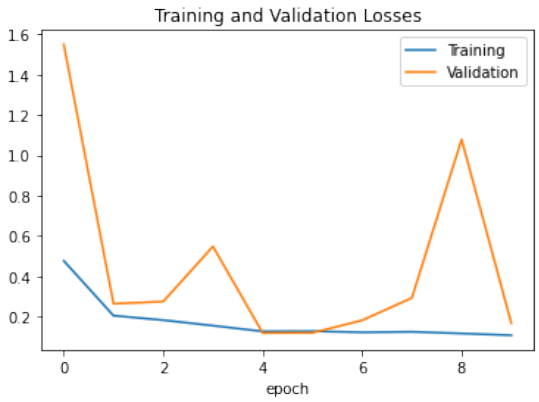


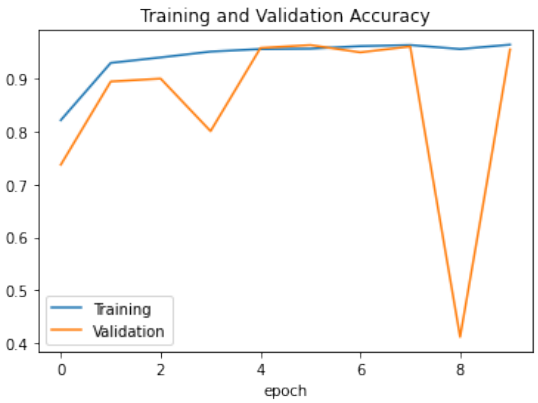


7.9 Test Model G: Model C Variation (Batch Size 128 & Batch Normalisation)

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_34 (Conv2D)          (None, 150, 150, 32)      2432

 max_pooling2d_34 (MaxPoolin  (None, 75, 75, 32)       0
 g2D)

 dropout_49 (Dropout)        (None, 75, 75, 32)        0

 conv2d_35 (Conv2D)          (None, 75, 75, 64)        18496

 max_pooling2d_35 (MaxPoolin  (None, 37, 37, 64)       0
 g2D)

 dropout_50 (Dropout)        (None, 37, 37, 64)        0

 flatten_16 (Flatten)        (None, 87616)             0

 dense_31 (Dense)            (None, 256)               22429952

 batch_normalization (BatchN  (None, 256)              1024
 ormalization)

 dropout_51 (Dropout)        (None, 256)               0

 dense_32 (Dense)            (None, 1)                 257

=================================================================
Total params: 22,452,161
Trainable params: 22,451,649
Non-trainable params: 512
```

```
4/4 [==============================] - 18s 4s/step - loss: 0.2221 -
accuracy: 0.9050
test loss: 0.2221333533525467 test acc: 0.9049586653709412
```

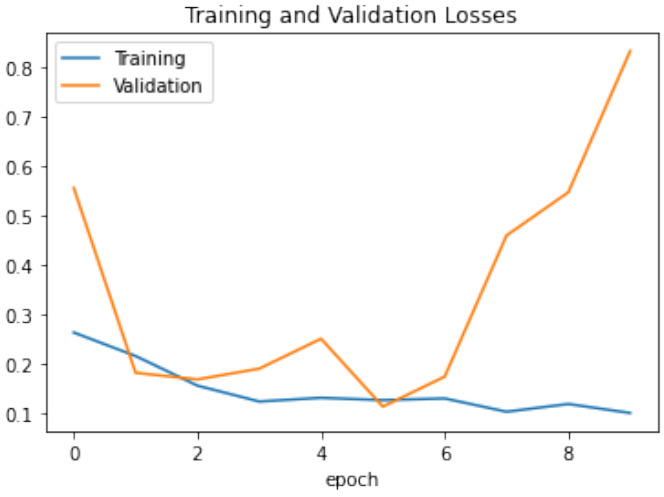


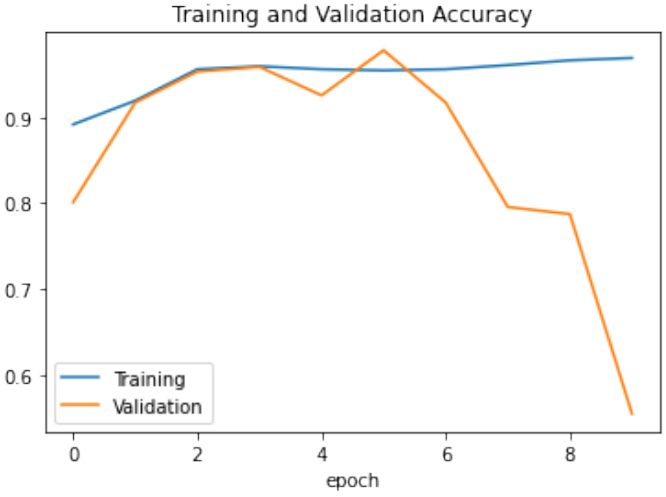


7.10 Test Model H: Model C Variation (Batch Normalisation)

```
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_36 (Conv2D)          (None, 150, 150, 32)      2432

 max_pooling2d_36 (MaxPoolin  (None, 75, 75, 32)       0
 g2D)

 dropout_52 (Dropout)        (None, 75, 75, 32)        0

 conv2d_37 (Conv2D)          (None, 75, 75, 64)        18496

 max_pooling2d_37 (MaxPoolin  (None, 37, 37, 64)       0
 g2D)

 dropout_53 (Dropout)        (None, 37, 37, 64)        0

 flatten_17 (Flatten)        (None, 87616)             0

 dense_33 (Dense)            (None, 256)               22429952

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 dropout_54 (Dropout)        (None, 256)               0

 dense_34 (Dense)            (None, 1)                 257

=================================================================
Total params: 22,452,161
Trainable params: 22,451,649
Non-trainable params: 512
```

```
16/16 [==============================] - 19s 1s/step - loss: 0.5371 -
accuracy: 0.7211
test loss: 0.5370672941207886 test acc: 0.7210744023323059
```

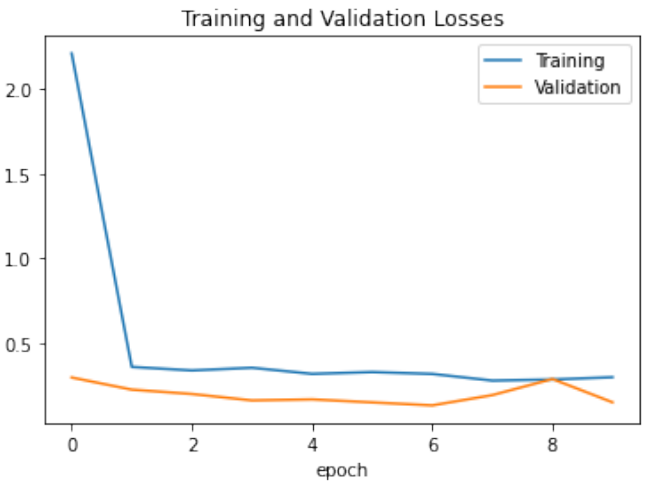


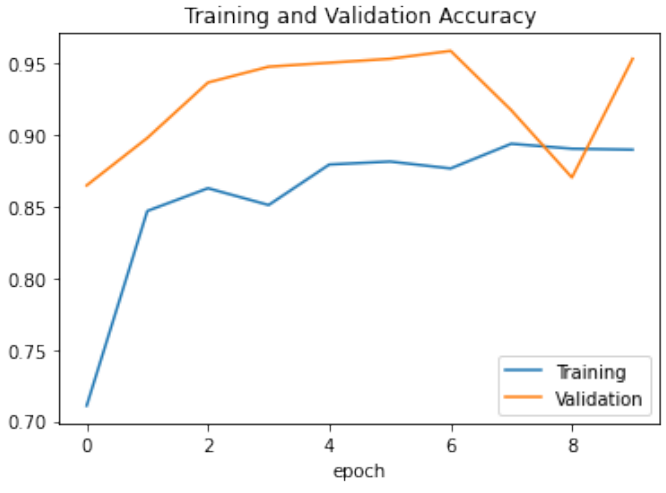


7.11 Test Model I: Model C Variation (Replaced ReLu with ELU)

```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_38 (Conv2D)          (None, 150, 150, 32)      2432

 max_pooling2d_38 (MaxPoolin  (None, 75, 75, 32)       0
 g2D)

 dropout_55 (Dropout)        (None, 75, 75, 32)        0

 conv2d_39 (Conv2D)          (None, 75, 75, 64)        18496

 max_pooling2d_39 (MaxPoolin  (None, 37, 37, 64)       0
 g2D)

 dropout_56 (Dropout)        (None, 37, 37, 64)        0

 flatten_18 (Flatten)        (None, 87616)             0

 dense_35 (Dense)            (None, 256)               22429952

 dropout_57 (Dropout)        (None, 256)               0

 dense_36 (Dense)            (None, 1)                 257

=================================================================
Total params: 22,451,137
Trainable params: 22,451,137
Non-trainable params: 0
```

```
16/16 [==============================] - 20s 1s/step - loss: 0.1778 -
accuracy: 0.9421
test loss: 0.17775020003318787 test acc: 0.942148745059967
```

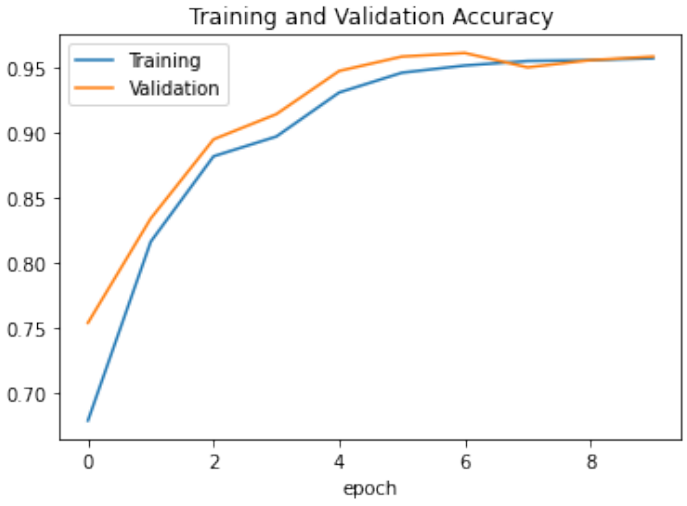


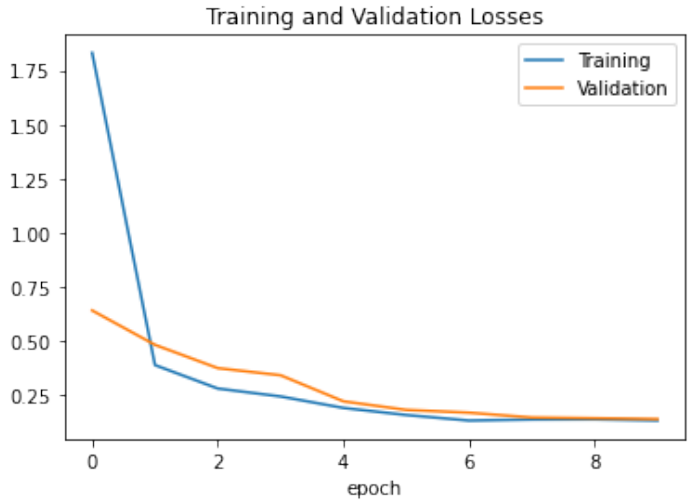


7.12 Test Model J: Model C Variation (AdaMax Optimiser)

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_14 (Conv2D)          (None, 150, 150, 32)      896

 max_pooling2d_10 (MaxPoolin  (None, 75, 75, 32)       0
 g2D)

 dropout_9 (Dropout)         (None, 75, 75, 32)        0

 conv2d_15 (Conv2D)          (None, 75, 75, 64)        18496

 max_pooling2d_11 (MaxPoolin  (None, 37, 37, 64)       0
 g2D)

 dropout_10 (Dropout)        (None, 37, 37, 64)        0

 flatten_7 (Flatten)         (None, 87616)             0

 dense_14 (Dense)            (None, 256)               22429952

 dropout_11 (Dropout)        (None, 256)               0

 dense_15 (Dense)            (None, 1)                 257

=================================================================
Total params: 22,449,601
Trainable params: 22,449,601
Non-trainable params: 0
_________________________________________________________________
```

```
16/16 [==============================] - 20s 1s/step - loss: 0.1903 -
accuracy: 0.9236
test loss: 0.19031934440135956 test acc: 0.9235537052154541
```

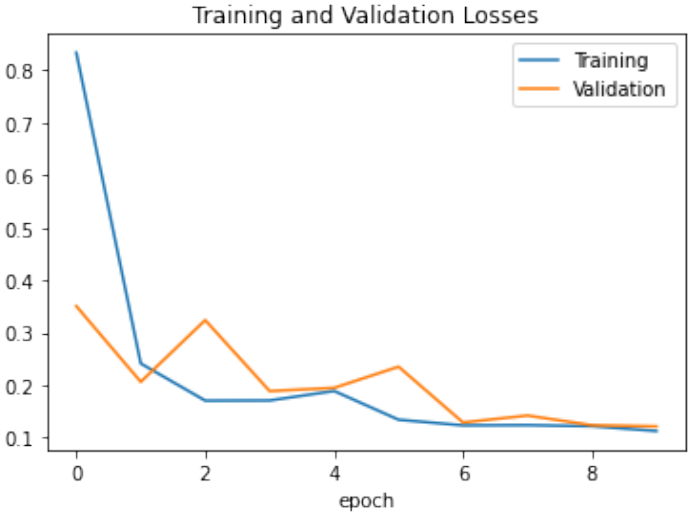


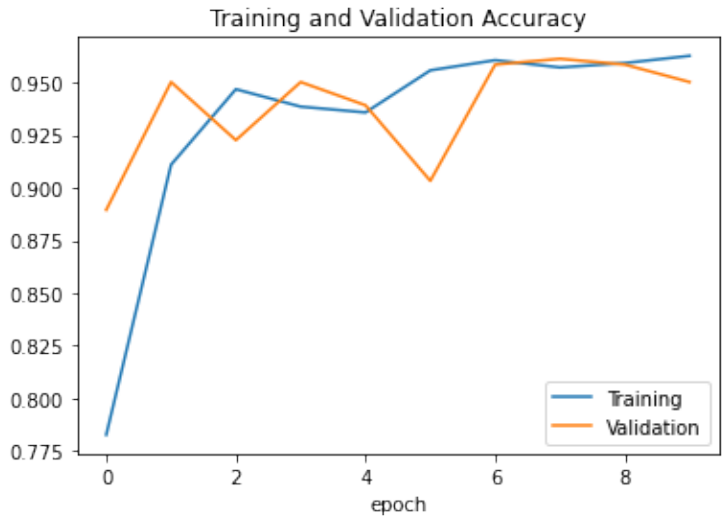


7.13 Test Model K: Model C Variation (Nadam Optimiser)

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_14 (Conv2D)          (None, 150, 150, 32)      896

 max_pooling2d_10 (MaxPoolin  (None, 75, 75, 32)       0
 g2D)

 dropout_9 (Dropout)         (None, 75, 75, 32)        0

 conv2d_15 (Conv2D)          (None, 75, 75, 64)        18496

 max_pooling2d_11 (MaxPoolin  (None, 37, 37, 64)       0
 g2D)

 dropout_10 (Dropout)        (None, 37, 37, 64)        0

 flatten_7 (Flatten)         (None, 87616)             0

 dense_14 (Dense)            (None, 256)               22429952

 dropout_11 (Dropout)        (None, 256)               0

 dense_15 (Dense)            (None, 1)                 257

=================================================================
Total params: 22,449,601
Trainable params: 22,449,601
Non-trainable params: 0
_________________________________________________________________
```

```
16/16 [==============================] - 20s 1s/step - loss: 0.1042 -
accuracy: 0.9690
test loss: 0.10422857850790024 test acc: 0.9690082669258118
```





7.14 Test Model L: Model C Variation (SGD Optimiser)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 150, 150, 32)      896

 max_pooling2d (MaxPooling2D  (None, 75, 75, 32)       0
 )

 dropout (Dropout)           (None, 75, 75, 32)        0

 conv2d_1 (Conv2D)           (None, 75, 75, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 37, 37, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 37, 37, 64)        0

 flatten (Flatten)           (None, 87616)             0

 dense (Dense)               (None, 256)               22429952

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 22,449,601
Trainable params: 22,449,601
Non-trainable params: 0
_________________________________________________________________
```

```
16/16 [==============================] - 19s 1s/step - loss: 0.2267 -
accuracy: 0.9008
test loss: 0.22673912346363068 test acc: 0.9008264541625977
```