AUTOMATED METHODS TO DETECT PNEUMONIA DISEASES FROM MEDICAL IMAGES

By

Charbel A.K.
Luiz Carlos de Jesus Junior
Muhammad Ali

Submitted to

The University of Liverpool

MSc Artificial Intelligence

Deep Learning

Word Count: 2100

06/03/2022

AUTOMATED METHODS TO DETECT PNEUMONIA DISEASES FROM MEDICAL IMAGES

Submitted to

The University of Liverpool

Word Count: 2100

06/03/2022

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

This report details the development of convolutional neural network (CNN) models to detect healthy lungs from pneumonia infected lungs. The primary objective is to construct a model with maximum accuracy and minimum loss by exploring different architectures and parameters. The code will be analysed and explained and overall performance of models discussed in detail. Furthermore, shortcomings of models along with possible improvements will also be discussed to provide a comprehensive critique.

Chapter 1. INTRODUCTION

Pneumonia is an infection caused by bacteria, viruses and fungi such as influenza, respiratory syncytial virus (RSV), and SARS-CoV-2 as seen in figure 1; it inflames lungs' air sacs and may also cause fluid or pus build-up along with additional symptoms such as cough, fever, chills and trouble breathing (American Lung Association , 2021).

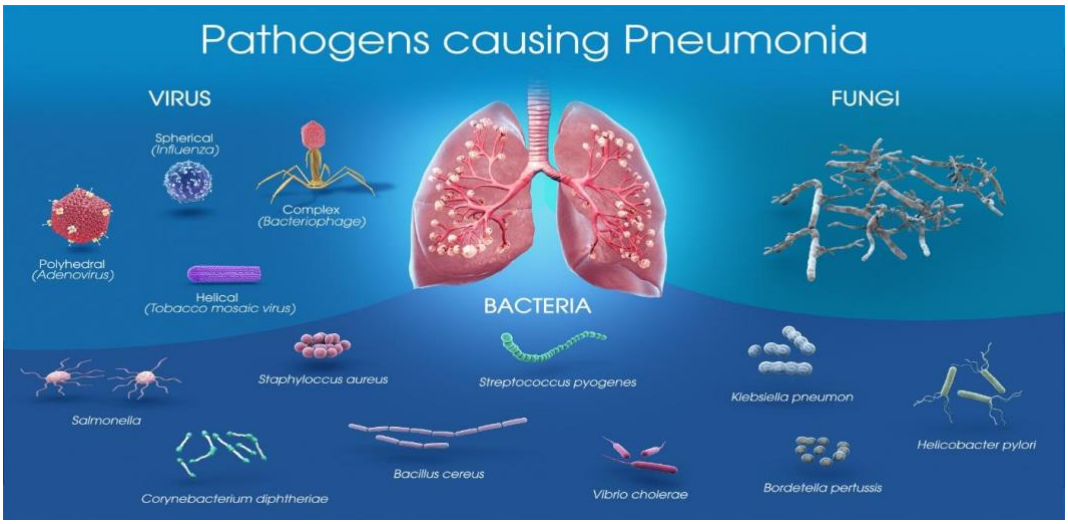

Figure 1: Causes of Pneumonia (KHERA, 2018).

The dataset provided by Kaggle consists of 5856 images (X-ray images carefully labelled by radiologists), including 5216 training images, 16 validation and 624 test images in two classes. Furthermore, the provided dataset is significantly

imbalanced as there are more than 3000 images with pneumonia and 1000 images without pneumonia.

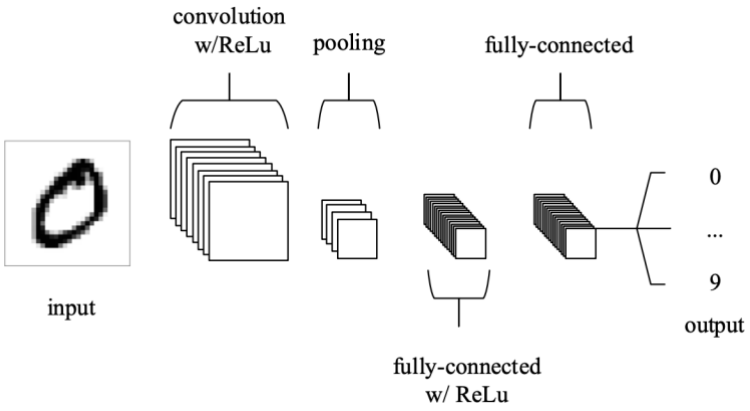

Figure 2: A generic CNN architecture (O'Shea & Nash, 2015).

CNN (figure 2) is a multilayer perceptron (MLP) with a special structure and consists of three basic layers: convolutional layer, pooling layer, and fully-connected layer with a rectified linear activation function (Acharya, et al., 2017). CNN model is extensively employed in a variety of fields related to pattern recognition (Albawi, et al., 2017), leveraging its ability to obtain abstract features as input propagates through the deeper layers (Acharya, et al., 2017). **Convolution-pooling** is the first layer to extract features from an input image and aids in preserving the relationship between pixels by learning image features (Albawi, et al., 2017). **Max-pooling** operation is applied to feature maps with the goal of size reduction (Acharya, et al., 2017). **Stride** (figure 3), one of the options for CNN which aids in parameter reduction, is the number of pixels shifts over the input matrix (Albawi, et al., 2017).

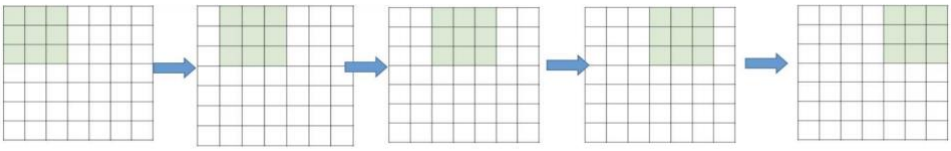

Figure 3: Stride 1 setting where filter window moves one time for each connection (Albawi, et al., 2017)

Padding is added to the frame of the image to provide more space for the kernel to cover the image. Same padding technique (figure 4), employed in the models constructed by authors, involves keeping size of image matrices unchanged as they are fed as input to the convolution process and obtaining matrix of same size which prevents loss of information from image data resulting from convolution (Gulgun & Erol, 2019).

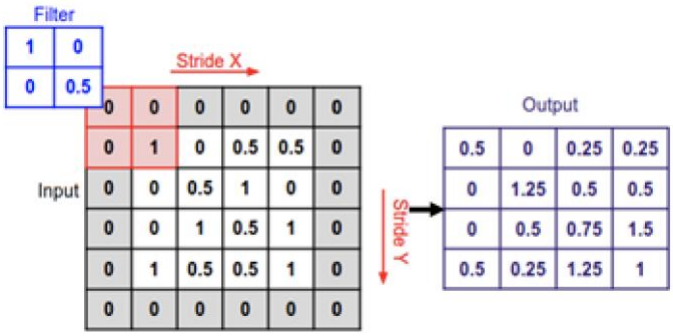

Figure 4: Same padding technique (Gulgun & Erol, 2019).

An **activation function** is the last component of the convolutional layer which is responsible for increasing non-linearity in the output (O'Shea & Nash, 2015).

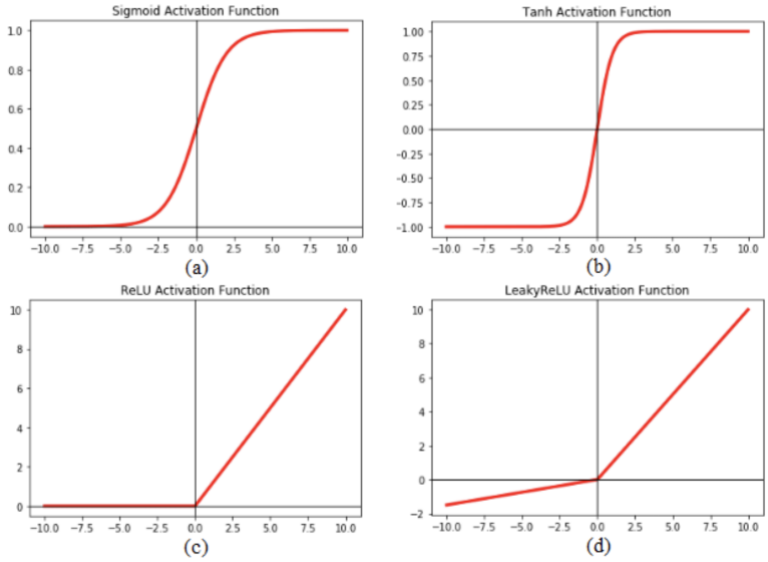

Figure 5: Different types of activation functions (Kandel & Castelli, 2021).

In addition to grasping key concepts of CNN architecture as highlighted above, understanding testing metrics and terminology is crucial to effectively model experiments and draw comparisons. **Confusion matrix** (figure 6) is a performance metric for machine learning classifications problems where output can be two or more classes. It is presented in a tabloid form with four different combinations of predicted and actual values (Acharya, et al., 2017). **Accuracy** is a measure that calculates number of right predictions over all available dataset whereas **recall** (sensitivity) is a ratio between correctly classified positive samples to the total number of positive samples in a given dataset (ALDAHOUL, et al., 2021). Finally, **F1-score** is a measure of model's accuracy on dataset and summarises recall and precision in one term: [(2 x precision x recall) ÷ (precision + recall)] (ALDAHOUL, et al., 2021).

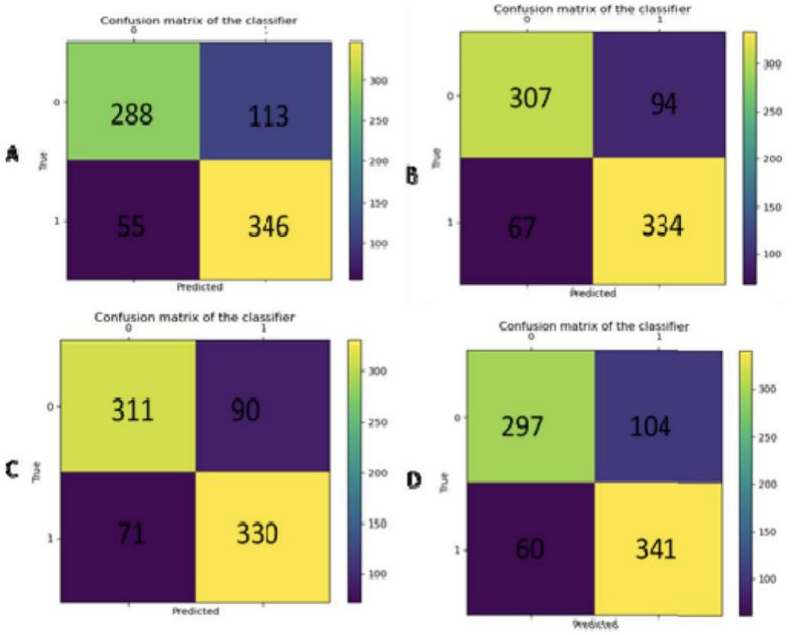

Figure 6: Example of a confusion matrix (ALDAHOUL, et al., 2021).

Chapter 2. SOFTWARE CONFIGURATION & TESTS

This chapter will detail the development of different CNN models, and is divided in 6 sections, with last section dedicated to exploring new techniques and highlight possible extensions, improvements to be carried out in future.

2.1 Data preprocessing

Data preprocessing plays a crucial role in machine learning and deep learning algorithms and is mostly compulsory to achieve better performance (Zheng, et al., 2018). Methods such as resizing, normalisation and cropping are known to increase CNN's performance (Pitaloka, et al., 2017).

The data processing step is the same for all the models discussed below; All the images from the train, test and validation folders were converted to grayscale and resized to 150x150 pixels. Moreover, pixels data were normalised to the range [0,1]. For more details on pre-processing steps and model architectures, check appendix. All the models employ a batch size of 32 and 10 epochs.

2.2 VGG16

To access the full code of VGG16 model, go to this link: **https://colab.research.google.com/drive/1k0pBFhvg8MlOlo8KHsvuwhAe5l3RHY2f#scrollTo=T8HAQB91UkXh**

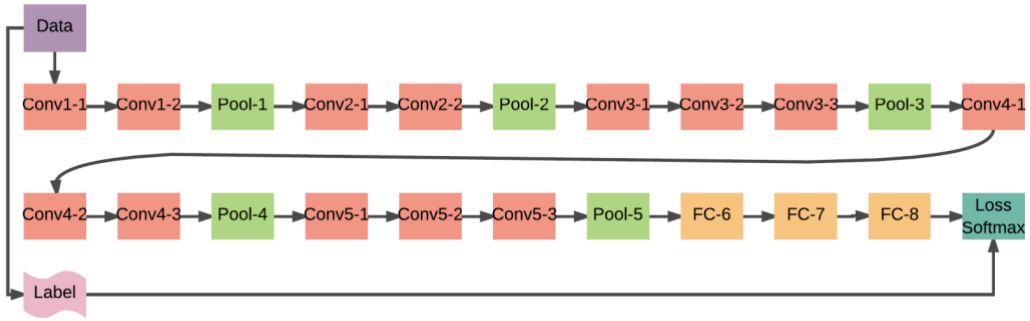

Figure 7: Workflow of VGG16 model (Qassim, et al., 2018).

For our experiments, VGG16 (Visual Geometry Group) architecture (figure 8) uses 16 weight layers with different convolution networks that employ various convolution filters, doubling from 64 to 512. Each convolution layer uses padding = 'same' setting and ReLU as activation function. Although sigmoid has been extensively used in past decades to train deep neural networks (DNNs), ReLU's ability to solve vanishing gradient problems by employing zero gradient has seen it gradually become more popular in training neural networks for image classification (Mastromichalakis, 2021).

```
model = keras.Sequential(
    [
        Conv2D(input_shape = (img_size,img_size,1), filters=64, kernel_size=(3,3), padding = 'same', activation = 'relu'),
        Conv2D(filters=64, kernel_size=(3,3), padding = 'same', activation = 'relu'),
        MaxPool2D(pool_size=(2,2), strides=(2,2)),
        Conv2D(filters=128, kernel_size=(3,3), padding = 'same', activation = 'relu'),
        Conv2D(filters=128, kernel_size=(3,3), padding = 'same', activation = 'relu'),
        MaxPool2D(pool_size=(2,2), strides=(2,2)),
        Conv2D(filters=256, kernel_size=(3,3), padding = 'same', activation = 'relu'),
        Conv2D(filters=256, kernel_size=(3,3), padding = 'same', activation = 'relu'),
        Conv2D(filters=256, kernel_size=(3,3), padding = 'same', activation = 'relu'),
        MaxPool2D(pool_size=(2,2), strides=(2,2)),
        Conv2D(filters=512, kernel_size=(3,3), padding = 'same', activation = 'relu'),
        Conv2D(filters=512, kernel_size=(3,3), padding = 'same', activation = 'relu'),
        Conv2D(filters=512, kernel_size=(3,3), padding = 'same', activation = 'relu'),
        MaxPool2D(pool_size=(2,2), strides=(2,2)),
        Conv2D(filters=512, kernel_size=(3,3), padding = 'same', activation = 'relu'),
        Conv2D(filters=512, kernel_size=(3,3), padding = 'same', activation = 'relu'),
        Conv2D(filters=512, kernel_size=(3,3), padding = 'same', activation = 'relu'),
        MaxPool2D(pool_size=(2,2), strides=(2,2)),
        Flatten(),
        Dense(units=4096, activation='relu'),
        Dense(units=4096, activation='relu'),
        Dense(units=1, activation='sigmoid')
    ]
)
```

Figure 8: VGG16 architecture.

Furthermore, max pooling with same pooling size and stride is used between groups of convolution layers whereas sigmoid function is used in output layer.

Table 1:Parameters of VGG16 model.

Parameter	Value
Loss	Binary cross-entropy
Optimizer	Adam with learning rate = 0.001
Metric	Accuracy

See table 1 and 2 for model parameters and confusion matrix respectively.

Table 2: VGG16 confusion matrix.

		PREDICTED	
		PNEUMONIA	NORMAL
ACTUAL	PNEUMONIA	390	0
	NORMAL	234	0

This model achieves an accuracy of 62.5% and a loss value of 0.6817. For detailed result metrics, see table 3 below.

Table 3: VGG16 Precision, Recall, f1-score and support metrics.

	Precision	Recall	f1-score	support
PNEUMONIA	0.62	1.0	0.77	390
NORMAL	0.00	0.00	0.00	234
Accuracy	-	-	0.62	624
Macro Average	0.31	0.5	0.38	624
Weighted Average	0.39	0.62	0.48	624

2.3 SqueezeNet

To access the full code of SqueezeNet model, go to this link:
https://colab.research.google.com/drive/12C9a7WyHeGUGGjoobglXUhs5i0OmhVgL#scrollTo=tktWeJ-i9fGv

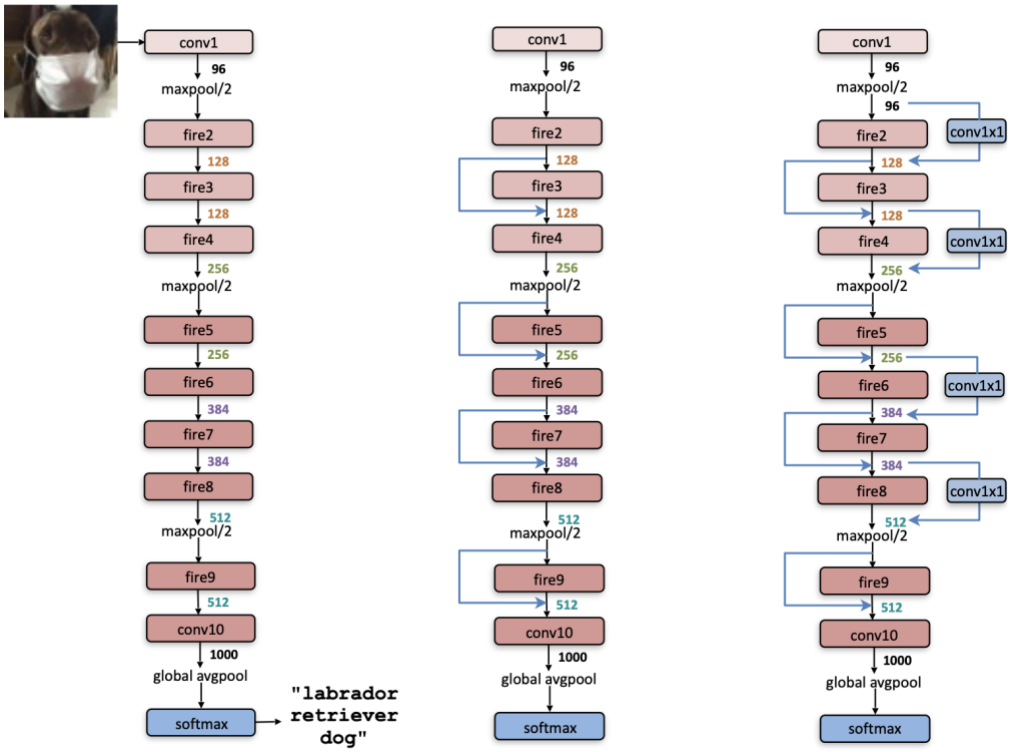

Figure 9: Overview of SqueezeNet architecture (Iandola, et al., 2017).

SqueezeNet is a highly efficient CNN architecture which has relatively few parameters and offers competitive accuracy. SqueezeNet allows for more distributed training approach with both less overhead and better embedded deployment (Iandola, et al., 2017).

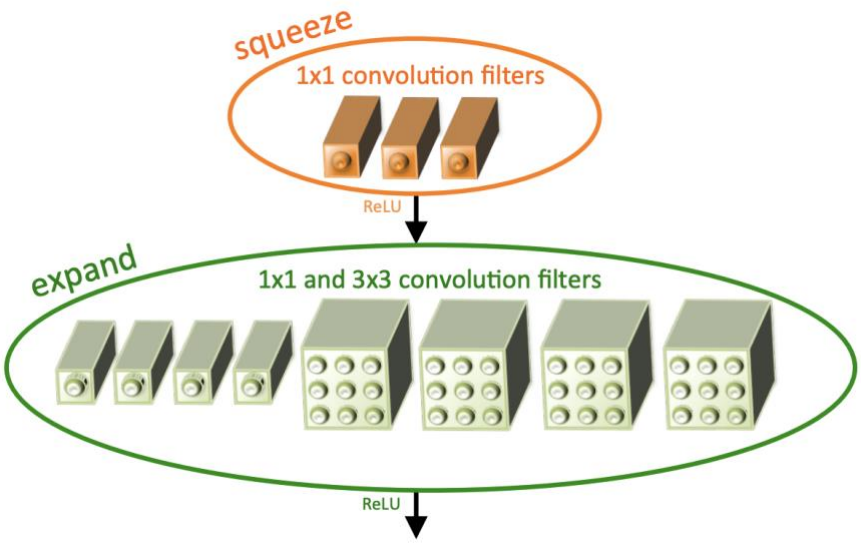

Figure 10: Overview of fire module in a SqueezeNet model (Iandola, et al., 2017).

Fire module (figure 10), crucial building block of a SqueezeNet model, consists of a squeeze convolution layer comprising of 1x1 filters, which feeds in to an expand layer that comprises of a mix of 1x1 and 3x3 convolution filters. Figure 11 shows more details of SqueezeNet model constructed for pneumonia detection by authors.

```
def FireModule(s_1x1, e_1x1, e_3x3, name):

    def layer(x):
        squeeze = Conv2D(s_1x1, 1, activation='relu', kernel_initializer='he_normal', name=name+'_squeeze')(x)
        squeeze = BatchNormalization(name=name+'_squeeze_bn')(squeeze)

        expand_1x1 = Conv2D(e_1x1, 1, padding='same', activation='relu', kernel_initializer='he_normal', name=name+'_expand_1x1')(squeeze)

        expand_3x3 = Conv2D(e_3x3, 3, padding='same', activation='relu', kernel_initializer='he_normal', name=name+'_expand_3x3')(squeeze)

        expand_merge = Concatenate(axis=3, name=name+'_expand_merge')([expand_1x1, expand_3x3])
        return expand_merge

    return layer
```

```
def SqueezeNet(nb_classes=1, input_shape=(img_size,img_size,1)):

    input_image = Input(shape=input_shape)
    conv1 = Conv2D(96, 7, 2, activation='relu', kernel_initializer='he_normal', name='conv1')(input_image)
    maxpool1 = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), name='maxpool1')(conv1)
    fire2 = FireModule(s_1x1=16, e_1x1=64, e_3x3=64, name='fire2')(maxpool1)
    fire3 = FireModule(s_1x1=16, e_1x1=64, e_3x3=64, name='fire3')(fire2)
    fire4 = FireModule(s_1x1=32, e_1x1=128, e_3x3=128, name='fire4')(fire3)
    maxpool4 = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), name='maxpool4')(fire4)
    fire5 = FireModule(s_1x1=32, e_1x1=128, e_3x3=128, name='fire5')(maxpool4)
    fire6 = FireModule(s_1x1=48, e_1x1=192, e_3x3=192, name='fire6')(fire5)
    fire7 = FireModule(s_1x1=48, e_1x1=192, e_3x3=192, name='fire7')(fire6)
    fire8 = FireModule(s_1x1=64, e_1x1=256, e_3x3=256, name='fire8')(fire7)
    maxpool8 = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), name='maxpool8')(fire7)
    fire9 = FireModule(s_1x1=64, e_1x1=256, e_3x3=256, name='fire9')(maxpool8)
    fire9_dropout = Dropout(rate=0.5, name='fire9_dropout')(fire9)
    conv10 = Conv2D(nb_classes, 1, activation='relu', kernel_initializer='he_normal', name='conv10')(fire9_dropout)
    conv10 = BatchNormalization(name='conv10_bn')(conv10)
    avgpool10 = GlobalAveragePooling2D(name='avgpool10')(conv10)
    sigmoid = Activation('sigmoid', name='sigmoid')(avgpool10)
    model = Model(inputs=input_image, outputs=[sigmoid])
    return model
```

Figure 11: SqueezeNet architecture.

Table 4: Parameters of SqueezeNet.

Parameter	Value
Loss	Binary cross-entropy
Optimizer	Adam with learning rate = 0.04
Metric	Accuracy

ReLU, applied through squeeze and expand layers, along with sigmoid in outoutlayer and a dropout with 50% ratio employed after the fire9 module yields comparatively better results. Dropout implementation also effectively addresses the issue of overfitting during training stage (Hwa Kieu, et al., 2020).

Table 5: SqueezeNet performance metrics.

	Precision	Recall	f1-score	support
PNEUMONIA	0.81	0.93	0.86	390
NORMAL	0.84	0.63	0.72	234
Accuracy	-	-	0.82	624
Macro Average	0.82	0.78	0.79	624
Weighted Average	0.82	0.82	0.81	624

The model, with the parameters shown in table 4, achieved a loss value of 0.5478 and an accuracy of 81.57%. For more detailed performance metrics and confusion matrix, see table 5 and 6 respectively.

Table 6: SqueezeNet confusion matrix.

		PREDICTED	
		PNEUMONIA	NORMAL
ACTUAL	PNEUMONIA	362	28
	NORMAL	87	147

2.4 DenseNet

To access the full code of DenseNet model, go to this link:
https://colab.research.google.com/drive/1qTedMCbP8gRGnULAXRBBsdiswYj11K7o#scrollTo=VPUot1JNV2nr

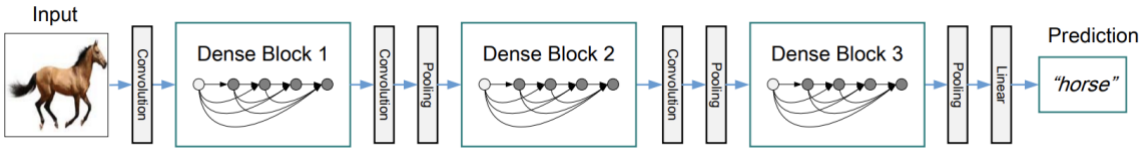

Figure 12: Overview of DenseNet architecture (Huang, et al., 2018).

Apart from having better parameter efficiency, DenseNet configuration (figure 12) has improved flow of information and gradients within the network which aids in training significantly. Moreover, training of deeper network architectures is achieved by direct access between each layer and the gradients from the loss function and the original input function (Huang, et al., 2018). Another salient feature and reason for exploring this confguration was dense connections's regularisation effect (Huang, et al., 2018) which aids in overfitting reduction, especially in smaller training set sizes such as the one employed in this report.

```python
def H(inputs, num_filters, dropout_rate):
    x = tf.keras.layers.BatchNormalization(epsilon=eps)(inputs)
    x = tf.keras.layers.Activation('relu')(x)
    x = tf.keras.layers.ZeroPadding2D((1, 1))(x)
    x = tf.keras.layers.Conv2D(num_filters, kernel_size=(3, 3), use_bias=False, kernel_initializer='he_normal')(x)
    x = tf.keras.layers.Dropout(rate=dropout_rate )(x)
    return x

def transition(inputs, num_filters, compression_factor, dropout_rate):
    x = tf.keras.layers.BatchNormalization(epsilon=eps)(inputs)
    x = tf.keras.layers.Activation('relu')(x)
    num_feature_maps = inputs.shape[1]

    x = tf.keras.layers.Conv2D(np.floor(compression_factor * num_feature_maps).astype(np.int),
                               kernel_size=(1, 1), use_bias=False, padding='same', kernel_initializer='he_normal', kernel_regularizer=tf.keras.regularizers.l2( 1e-4 ))(x)
    x = tf.keras.layers.Dropout(rate=dropout_rate)(x)
    x = tf.keras.layers.AveragePooling2D(pool_size=(2, 2))(x)
    return x

def dense_block(inputs, num_layers, num_filters, growth_rate , dropout_rate):
    for i in range(num_layers):
        conv_outputs = H(inputs, num_filters, dropout_rate)
        inputs = tf.keras.layers.Concatenate()([conv_outputs, inputs])
        num_filters += growth_rate
    return inputs, num_filters
```

```python
input_shape = (img_size, img_size, 1)
num_blocks = 3
num_layers_per_block = 4
growth_rate = 16
dropout_rate = 0.4
compress_factor = 0.5
eps = 1.1e-5

num_filters = 16

inputs = tf.keras.layers.Input(shape=input_shape)
x = tf.keras.layers.Conv2D(num_filters, kernel_size=(3, 3), use_bias=False, kernel_initializer='he_normal', kernel_regularizer=tf.keras.regularizers.l2(1e-4))(inputs)

for i in range(num_blocks):
    x, num_filters = dense_block(x, num_layers_per_block, num_filters, growth_rate, dropout_rate)
    x = transition(x, num_filters, compress_factor, dropout_rate )

x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(1)(x)
outputs = tf.keras.layers.Activation('sigmoid')(x)
```

Figure 13: Detailed configuration of DenseNet parameters and settings.

Max pooling and average pooling are used between the dense blocks in what's known as transition layers. Note that the output layer was adapted to our context of binary classification as opposed to multiclass classification, hence sigmoid function was selected over softmax due to it's better performance (Mastromichalakis, 2021).

Table 7: DenseNet Parameters.

Parameter	Value
Loss	Binary cross-entropy
Optimizer	Adam with learning rate = 0.0001
Metric	Accuracy

The model, with the parameters shown in table 7, achieved a loss value of 1.42 and an accuracy of 66.50%. For more detailed performance metrics and confusion matrix, see table 8 and 9 respectively.

Table 8: DenseNet performance metrics.

	Precision	Recall	f1-score	support
PNEUMONIA	0.65	1.00	0.79	390
NORMAL	1.00	0.11	0.19	234
Accuracy	-	-	0.67	624
Macro Average	0.83	0.55	0.49	624
Weighted Average	0.78	0.67	0.57	624

Table 9: DenseNet confusion matrix.

		PREDICTED	
		PNEUMONIA	NORMAL
ACTUAL	PNEUMONIA	390	0
	NORMAL	209	25

2.5 AutoKeras

To access the full code of AutoKeras model, go to this link:
https://colab.research.google.com/drive/1ScldPtmMYOn9zFHOZkwYKdtlj8qYE9u1#scrollTo=8MszmCAzleLI

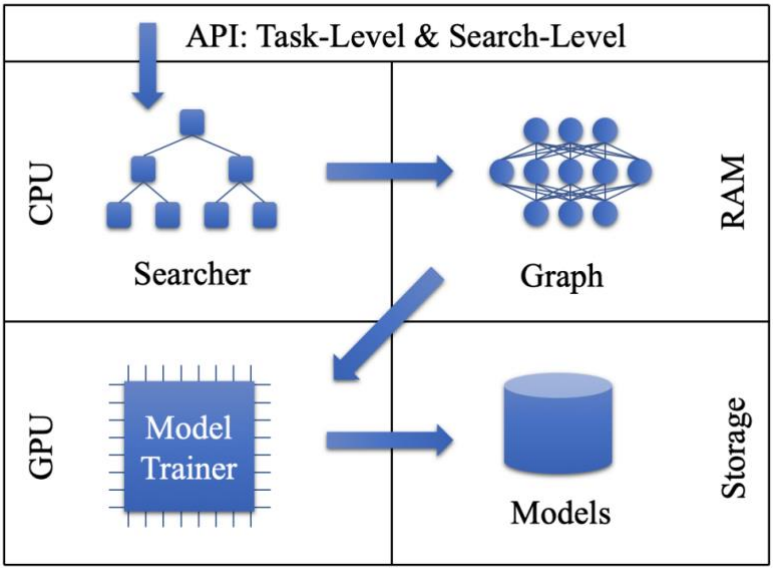

Figure 14: AutoKeras model overview (Jin, et al., 2019).

Automated Machine Learning (AutoML) has become an increasingly important research topic, with wide range of applications within the ML and deep learning (DL) domain. AutoKeras is an AutoML system based in Keras that allows domain experts who aren't familiar with machine learning (ML) methods to employ ML techniques both with ease and effectiveness (Jin, et al., 2019). The idea is that the system will automatically identify the optimum parameters/model/architecture for a certain problem.

Table 10: AutoKeras confusion matrix.

		PREDICTED	
		PNEUMONIA	NORMAL
ACTUAL	PNEUMONIA	0	390
	NORMAL	0	234

AutoKeras model achieved a loss value of 4.1537 and an accuracy of 37.5%. The authors contend that AutoML, in this case, failed to find optimum architecture at default settings so future experiments could explore better tuning of AutoKeras parameters.See table 10 and 11 for confusion matrix and performance metrics respectively.

Table 11: AutoKeras Performance metrics.

	Precision	Recall	f1-score	support
PNEUMONIA	0.00	0.00	0.00	390
NORMAL	0.38	1.00	0.55	234
Accuracy	-	-	0.38	624
Macro Average	0.19	0.50	0.27	624
Weighted Average	0.14	0.38	0.20	624

2.6 Kaggle

To access the full code of Kaggle model, go to this link:
https://colab.research.google.com/drive/1CyFJEztMrxvfuSr2xna9IxwQlY7icdLP#scrollTo=3zzuuWyeWFLw

Kaggle is a platform where companies and institutions can create data challenges for data scientists and machine learning enthusiasts, creating a challenging environment for both new learners and seasoned coders.

In order to have an effective comparative analysis, authors of this report decided to include the model of one of the top competitors in the pneumonia challenge, an

entry that received over 200 upvotes and more than 1000 copies from his codebase were made. The author, (Mathur, 2020), opts for a mix of convolution layers, with strides equal to 1 and 2, padding = 'same', ReLU as an activation function, batch normalisation, max-pooling and dropout (rate = 0.2) as settings of CNN architecture.

Table 12: Kaggle parameters.

Parameter	Value
Loss	Binary cross-entropy
Optimizer	RMSprop
Metric	Accuracy

The model, with the parameters shown in table 12, achieved a loss value of 2.4284 and an accuracy of 78.68%. See table 13 and 14 for confusion matrix and performance metrics respectively.

Table 13: Kaggle confusion matrix.

		PREDICTED	
		PNEUMONIA	NORMAL
ACTUAL	PNEUMONIA	387	3
	NORMAL	130	104

Table 14: Kaggle performance metrics.

	Precision	Recall	f1-score	support
PNEUMONIA	0.75	0.99	0.85	390
NORMAL	0.97	0.44	0.61	234
Accuracy	-	-	0.79	624
Macro Average	0.86	0.72	0.73	624
Weighted Average	0.83	0.79	0.76	624

2.7 Future Work

In this work, authors presented automated methods to detect Pneumonia diseases from medical images using Deep Learning architectures (VGG16, SqueezeNet, Densenet, Autokeras, etc.). The models presented can be improved by adding new layers, however, this will introduce more hyperparameters that should be adjusted. The main goal of this work will be to identify if there are any Deep Learning techniques which distinctly outperforms other techniques?

Future work aims to develop an end-to-end system for pneumonia identification via deep learning detection, segmentation, and classification. A large-sized dataset of different lung diseases will be collected to train the model to improve the accuracy of the disease detection. The performance can be improved by using more complex feature extraction based on Deep Learning techniques such as colour, texture, and shape, and using more datasets.

Furthermore, a model can be developed to predict the possibility of having Pneumonia disease in humans based on certain indicators, datasets collected from humans using sensors, blood tests, and other medical tests that can help doctors identify or predict if the patient has a genetic predisposition to develop Pneumonia disease.

Chapter 3. CONCLUSION

In this report, authors built different models to find the most optimum solution for pneumonia detection. Table 15 shows performance metrics of all models created.

Table 15: Weighted average of different performance metrics.

Model	Precision	Recall	f1-score	support
VGG16	0.39	0.62	0.48	624
SqueezeNet	0.82	0.82	0.81	624
DenseNet	0.78	0.67	0.57	624
AutoKeras	0.14	0.38	0.20	624
Kaggle	0.83	0.79	0.76	624

The weighted average represents the 'averaging of the support-weighted mean per label (in this case 'PNEUMONIA' and 'NORMAL'). Moreover, higher values mean a more acceptable value for both categories. The f1-score measures the accuracy of the dataset, combining both precision and recall metrics. In terms of *precision* which answers the crucial question 'What proportion of positive identifications were actually made' and recall which answers 'What proportion of actual positives were identified correctly', a holistic understanding of model's accuracy can be attained by these two answers. Therefore, the model with highest f1-score is considered to be the best, particularly in our case where provided dataset is imbalanced and in case of high positive vs negative cases, the formula will readjust value based on value of precision or recall values. SqueezeNet, with an f1-score of 0.81, is the highest performing model for this comparative analysis study.

REFERENCES

American Lung Association , 2021. Pneumonia Symptoms and Diagnosis. *American Lung Association Scientific and Medical Editorial*, 30 July.

KHERA, G., 2018. *Pneumonia.* [Online]
Available at: https://www.scientificanimations.com/pneumonia/
[Accessed 02 March 2022].

Hwa Kieu, S. T., Bade, A., Hijazi, M. H. A. & Kolivand, h., 2020. A Survey of Deep Learning for Lung Disease Detection on Medical Images: State-of-the-Art, Taxonomy, Issues and Future Directions. *Journal of Imaging.*

Albawi, S., Mohammed, T. A. & Zawi, S. A., 2017. *Understanding of a convolutional neural network.* Antalya, 2017 International Conference on Engineering and Technology (ICET).

Acharya, U. R. et al., 2017. A deep convolutional neural network model to classify heartbeats. *Computers in Biology and Medicine,* Volume 89, pp. 389-396.

O'Shea, K. & Nash, R., 2015. An Introduction to Convolutional Neural Networks. *arXiv.*

Kandel, I. & Castelli, M., 2021. Transfer Learning with Convolutional Neural Networks for Diabetic Retinopathy Image Classification. A Review. *Applied Sciences,* 10(6).

Gulgun, O. D. & Erol, H., 2019. CLASSIFICATION PERFORMANCE COMPARISONS OF DEEP LEARNING MODELS IN PNEUMONIA DIAGNOSIS USING CHEST X-RAY IMAGES. *Turkish Journal of Engineering.*

ALDAHOUL, N., Karim, H. A. & ABDULLAH, M. H. L., 2021. An Evaluation of Traditional and CNN-Based Feature Descriptors for Cartoon Pornography Detection. *IEEEAccess.*

Pitaloka, D. A., Wulandari, A., Basaruddin, T. & Liliana, D. Y., 2017. *Enhancing CNN with Preprocessing Stage in Automatic Emotion Recognition.* Bali, 2nd International Conference on Computer Science and Computational Intelligence 2017, ICCSCI.

Zheng, X., Wang, M. & Ordieres-Mere, J., 2018. Comparison of Data Preprocessing Approaches for Applying Deep Learning to Human Activity Recognition in the Context of Industry 4.0. *Sensors.*

Qassim, H., Verma, A. & Feinzimer, D., 2018. *Compressed residual-VGG16 CNN model for big data places image recognition.* Las Vegas, 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC).

Mastromichalakis, S., 2021. SigmoReLU: An improvement activation function by combining Sigmoid and ReLU. *Preprints.*
Iandola, F. N. et al., 2017. *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size.* Toulon, 5th International Conference on Learning Representations.

Huang, G., Liu, Z., Maaten, L. v. d. & Weinberger, K. Q., 2018. *Densely Connected Convolutional Networks.* [Online]
Available at: https://arxiv.org/pdf/1608.06993.pdf
[Accessed 05 March 2022].

Jin, H., Song, Q. & Hu, X., 2019. *Auto-Keras: An Efficient Neural Architecture Search System.* Anchorage, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.

Mathur, M., 2020. *Pneumonia Detection using CNN(92.6% Accuracy).* [Online]
Available at: https://www.kaggle.com/madz2000/pneumonia-detection-using-cnn-92-6-accuracy
[Accessed 05 March 2022].

APPENDICES

Appendix A. DATA PREPROCESSING

```
data_dir = '/content/drive/MyDrive/MSc_Liverpool_University_AI/CSCK506/week08/group_activity/dataset/chest_xray/'
train_data_dir = os.path.join(data_dir, 'train')
test_data_dir = os.path.join(data_dir, 'test')
val_data_dir = os.path.join(data_dir, 'val')

labels = ['PNEUMONIA', 'NORMAL']
img_size = 150

def get_data(data_dir, labels, img_size):
    data = []
    for label in labels:
        path = os.path.join(data_dir, label)
        class_num = labels.index(label)
        for img in os.listdir(path):
            try:
                img_arr = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
                resized_arr = cv2.resize(img_arr, (img_size, img_size)) # Reshaping images to preferred size
                data.append([resized_arr, class_num])
            except Exception as e:
                print(e)
    return np.array(data)
```

Figure 15: Data preprocessing.

```
train_set = get_data(train_data_dir, labels, img_size)
test_set = get_data(test_data_dir, labels, img_size)
val_set = get_data(val_data_dir, labels, img_size)
```

Figure 16: Data loading.

```
x_train = []
y_train = []

x_val = []
y_val = []

x_test = []
y_test = []

for feature, label in train_set:
    x_train.append(feature)
    y_train.append(label)

for feature, label in test_set:
    x_test.append(feature)
    y_test.append(label)

for feature, label in val_set:
    x_val.append(feature)
    y_val.append(label)
```

Figure 17: Data split.

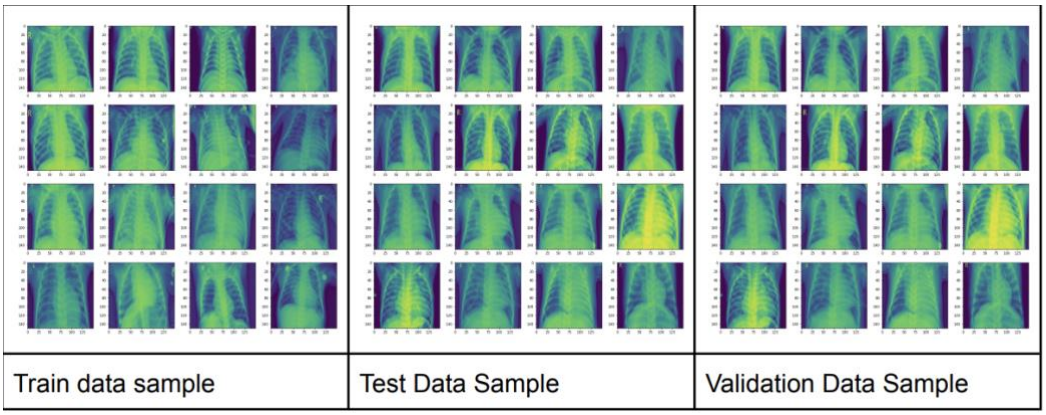

Figure 18: Data Visualisation.

```
x_train = np.array(x_train) / 255
x_val = np.array(x_val) / 255
x_test = np.array(x_test) / 255
```

Figure 19: Data normalisation.

```
x_train = x_train.reshape(-1, img_size, img_size, 1)
y_train = np.array(y_train)

x_val = x_val.reshape(-1, img_size, img_size, 1)
y_val = np.array(y_val)

x_test = x_test.reshape(-1, img_size, img_size, 1)
y_test = np.array(y_test)
```

Figure 20: Data resizing.

Appendix B. MODEL ARCHITECTURES

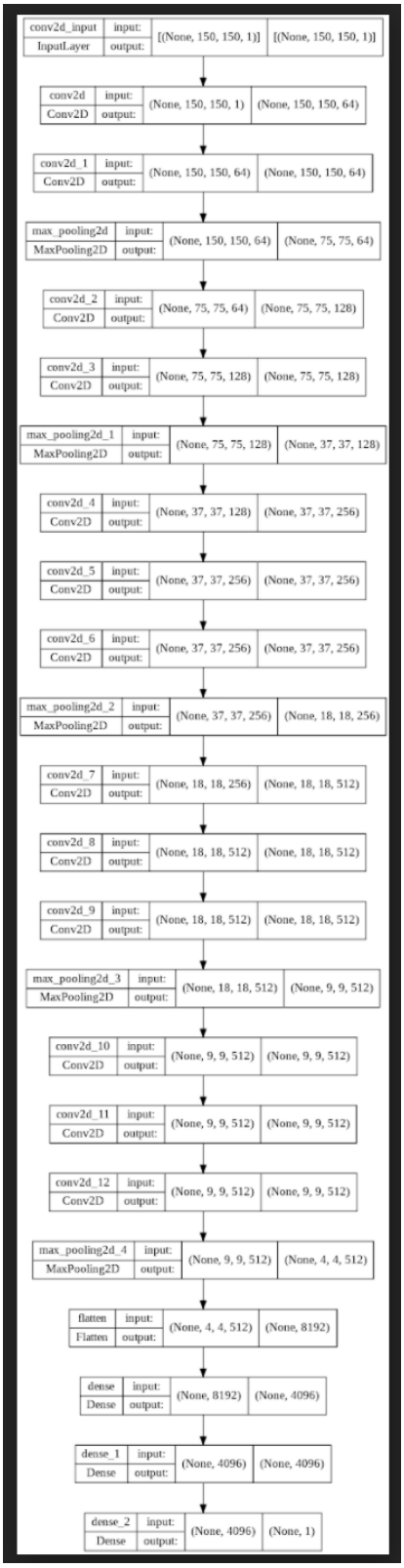

Figure 21: VGG16 network architecture.

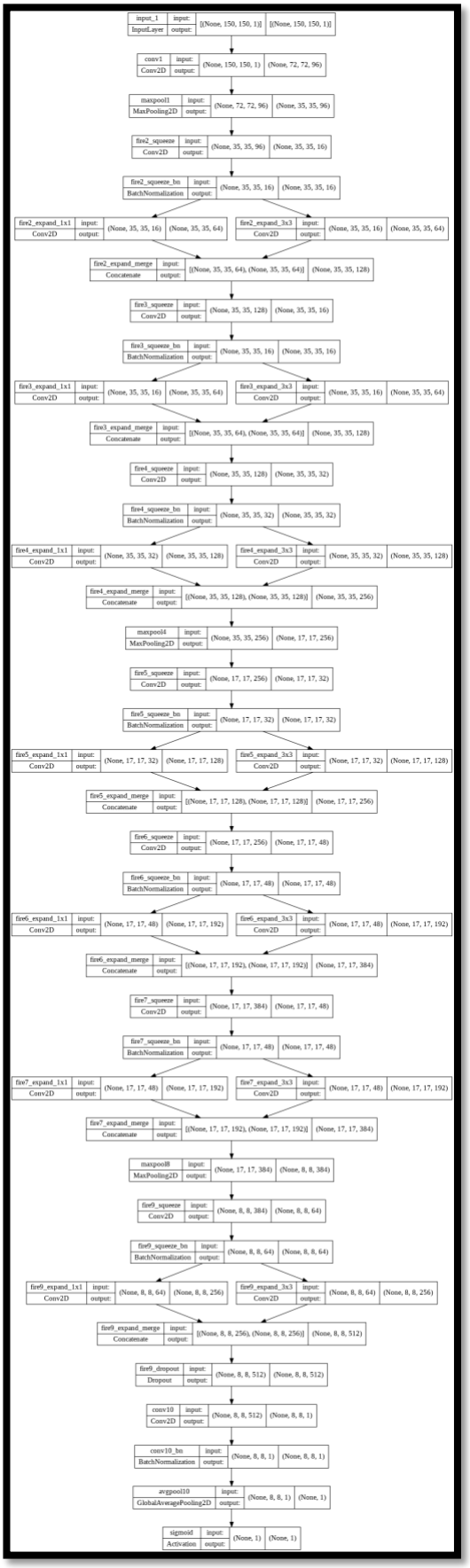

Figure 22: SqueezeNet architecture.

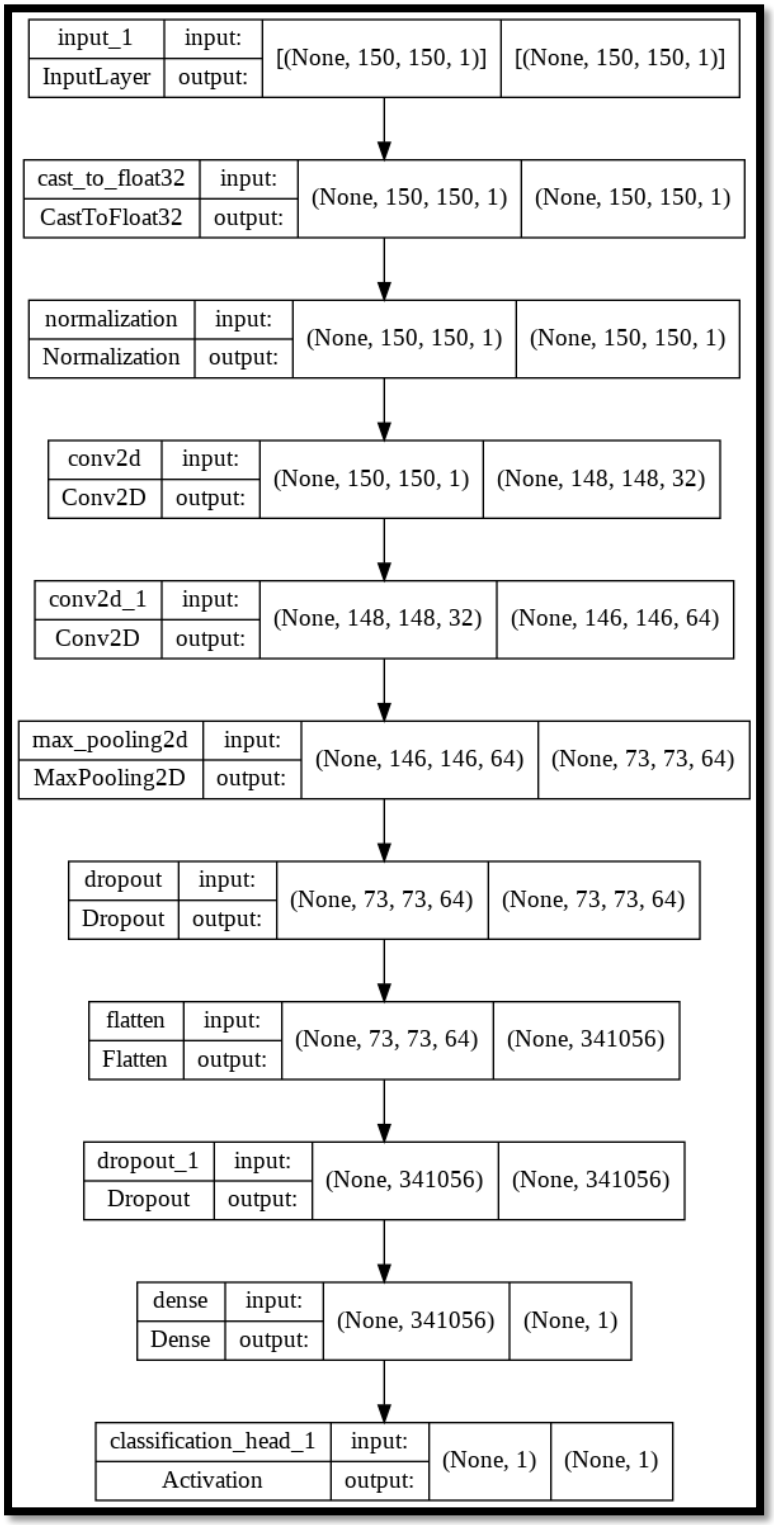

Figure 23: AutoKeras architecture.