Pneumonia detection using CNNs

36145142, 36302123, 36253953, 36229409

***Abstract*—The number of possibilities of what can be done with computer vision methods is rapidly increasing. A convolutional neural network is a leading method in the sector of computer vision, replacing neural networks that are expensive computationally. This report will investigate the specific application of CNNs in pneumonia detection. The focus was put on exploring all the building blocks that are part of the network, with examples of the most popular architectures: AlexNet and VGG-16. Testing our models on unseen data showed that accuracy was higher for a deeper network, VGG-16. However, both models gave acceptable results above 70%. Transfer learning was used to smooth the training process for computationally expensive VGG-16. The potential improvement could be achieved by using the ResNet model, which is more recent and much deeper network.**

I. Introduction

Computer vision [1] is a sector in computer science that concentrates on making computers see the world through images and videos like humans do, and be able to identify the image and come with insight that humans do or even things that we haven't learned yet. In today's world, computer vision has been rapidly progressing because of the sheer amount of visual data we store, which enables computers to train with large datasets. Moreover, the increase of performance in GPU and video cards to handle these types of calculations also facilitates the application of the computer vision models.

For computers to train as human mind do, the notion of neural network was invented [2]. They usually are composed of fully connected neurons. Each neuron receives inputs, processes it and produces a single output. It is then sent to many other neurons that repeat same process. This was great for simple classification and regression problems but for more complex problems, there was a need to add an activation function which was non-linear such as the famous sigmoid function. The problem with image processing and regular NN was that the input had too many features and was very demanding for the computer. It was also overfitting for the training set. This made scientists realise that there was a need to implement a new step in NN designed for image processing. Yann LeCun came up with CNN (convolution neural network) [3] [4] based on the visual cortex of humans. A basic understanding of the difference between the regular NN and CNN is that CNN creates a map of features using convolutional filters and then performs some sort of reduction of the size of the matrices using pooling layers. At the end of the CNN architecture, usually, at least the last two layers perform a normal NN with an additional dropout method to make the results more generalised. An amazing feature of CNN is that, in theory, if you train a model with many different groups, you will be able to have a head start in the training period by using the optimal parameters that the program found before. This method can be implemented for the classification problem on completely new dataset. CNN can accurately classify a feature independent of its position on the image. This is because performing the extra layers removes much of the noise of the data and groups similar pixels together.

This makes CNN extremely important in numerous sectors, such as business (self-checkout by just looking at the product), government (police could use facial recognition for finding suspects using cameras)or medicine (being able to look at an X-ray and decide if the person is healthy or not). Application of CNN in medicine could help overworked doctors to be confident in sending people which are classified being 80% healthy or whatever criteria they would find to be optimal. In this case, we would most likely prefer to train our model to have a higher recall score which would mean that we prefer our CNN to commit more mistakes in mislabeling healthy people as not healthy than the inverse, which could negatively impact lives.

In this report, we will define more rigorously CNN and its layers. We will then look at two types of architectures of CNN, AlexNet and VGG-16 (Visual Geometry Group), and the usage of them in medicine. Specifically in how to detect pneumonia based on the x-ray of chest.

II. Methods

A. Building blocks of CNN

The first and most fundamental layer of every CNN is the convolutional layer. It takes its name from the convolutional operator, which in mathematics, combines two functions to produce a third function that represents the amount of overlap between the two original functions [5]. In the context of CNN, we define convolution as:

$$S(i,\, j) = \sum_m \sum_n I(i+m,\, j+n) K(m,\, n)$$

where I is two-dimensional input image, K is our filter with dimensions m, n, and S is feature map with dimensions i, j. It is important to note that the mathematical formulation above is used for cross-correlation. However, it does not change anything in a deep learning context, and it is a convention that has been used in many ML libraries [6]. In practice, a feature map is computed by "sliding" filter along the input image and summing element-wise dot product of entries in the same positions. In Figure 1, we visualise a simplistic example with a 3x3 input and a 2x2 kernel.

In real-life applications, the input image size is much larger in width and height. On top of that, they have a depth of 3 for each RGB channel. Hence, the filter's deepness must match the number of channels to produce two-dimensional mapping. Usually, it is necessary to have multiple kernels; in that case, each two-dimensional mapping is stacked on top of the other.

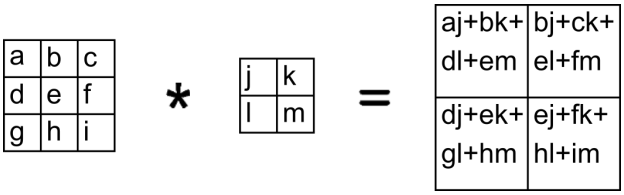

Fig. 1: Example of computing feature map

Thus we can deduce the dimensions of the output layer as $(n - f + 1, n - f + 1, n_f)$ where n is the size of the input, f size of the filter and n_f number of filters. An important modification to convolution operation is padding. It has been introduced to overcome two problems. Firstly, pixels on the edges of the image are misrepresented as the filter does not pass through them as often (for example top left pixel gets taken into account only once).
Moreover, as it has been seen before, each convolution leads to a decrease in the size of the image, which is not always desirable. Padding solves those issues by adding pixels with the value 0 around the edges of the image. Depending on the model's needs, the padding size can be adjusted. In Figure 2, we show padding with size 1.

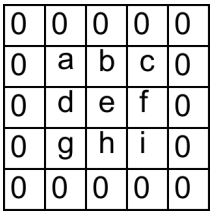

Fig. 2: Example of padding on 3x3 image

Two common choices for padding size, which are also supported by Keras library in Python, are "valid" and "same". Valid padding means that no padding is applied and is a default choice in the Conv2D method in Python. The same padding uses padding size for which output preserves input size. It can be calculated by the formula $p = \frac{f-1}{2}$, where f is the size of the filter. By that, we can deduce that the size of the filters is usually an odd number. Another choice we have to make when doing convolution is stride. It determines the step size at which we "slide" through the input image. By default, it is set to 1, and the larger it becomes, the smaller the feature map becomes. After calculating the feature map, it is necessary to introduce non-linearity to our model. The most common choice in CNN is the ReLU function. The reason for that preference is a simpler definition of the function itself and its derivative. It makes errors less likely to occur when doing backpropagation to re-evaluate the weights of parameters. In the sigmoid function, the gradient is around 0 almost everywhere except in the centre of the function, which leads to the vanishing of the gradient signal at deeper stages of the network [7]. In ReLU case, the situation is simpler as the gradient is either 0 or 1.

Another important layer in CNN's structure is the pooling layer. Pooling takes the output of the layer before and replaces the neighbourhood of outputs with a summary statistic. This summary statistic range from a normal average or weighted average based on distance from the middle pixel. However, the most effective and popular choice was found to be max pooling, which picks the maximum. Parameters to specify for the pooling layer are size and stride. An example of pooling with window of size 2 and stride 1 is shown in Figure 3.

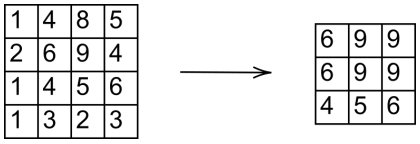

Fig. 3: Example of 2x2 pooling with stride 1

The reasons for pooling are a reduction in the output size, which leads to speeding up the computations but also increase in the robustness of finding the features in the image. More specifically, it makes the model invariant to translation in the input image [6]. It means that even if the feature that filters tried to identify is slightly misplaced, its value will still be preserved in the output. For instance, if the goal is to detect a plane on the image, it is not necessary for the plane to be on the exact pixels as in the training set.

After all convolution and max-pooling operations, the outputs get flattened into a one-dimensional vector, which is then proceeded by a fully connected layer. From now on, the network will work like a standard neural network. In essence, each node is densely connected with other nodes from other layers. Hence, this part of the network is very expensive computational-wise. Nonetheless, not as expensive as it would be in a traditional neural network approach. Regularization techniques can be applied between fully connected layers to reduce model complexity and avoid overfitting. The most popular method is dropout. In this method, some nodes are being turned off based on some set probability.

Now, as all the building blocks have been introduced, we can look into some of the classical convolutional neural network architectures that were proven to work well with image classification.

B. AlexNet

Alexnet is an architecture designed by Alex Krizhevsky, which competed and won the 2012 ImageNet challenge. From the official paper [8], we can read that it is built with 8 learning layers: 5 convolutional layers and 3 fully connected layers. The exact architecture has been presented in table I.
AlexNet was revolutionary at the time for a few reasons. Firstly, it paved the way for using ReLU as an activation function. According to the paper [8], it reached a training error of 25%, 6 times faster than when using the standard at the time $\tanh$ activation. They performed ReLu at the end of each layer except the output layer. Secondly, training the model on the ImageNet dataset, which contains 1.2 million training images, was too computationally expensive to train on

Layer	Channels	Size	Kernel size	Stride	Padding
Input	3 (RGB)	227x227	-	-	-
Convolution	96	55x55	11x11	4	valid
Max-Pooling	96	27x27	3x3	2	valid
Convolution	256	27x27	5x5	1	same
Max-Pooling	256	13x13	3x3	2	valid
Convolution	384	13x13	3x3	1	same
Convolution	384	13x13	3x3	1	same
Convolution	256	13x13	3x3	1	same
Max-Pooling	256	6x6	3x3	2	valid
Fully Connected	-	4096	-	-	-
Fully Connected	-	4096	-	-	-
Output	-	1000	-	-	-

TABLE I: AlexNet architecture

one GPU. Hence, training has been spread across two GPUs in parallel. AlexNet used dropout in the first two fully connected layers, which helps prevent overfitting. In total, the network has 60 million trainable parameters.

C. VGG-16

VGG-16 model was proposed by the Visual Geometry Group and won the 2014 edition of the ImageNet challenge. The interesting feature of this architecture is that the size of the filter, stride and padding are constants for all convolutional and max-pooling layers. Table II presents the entire architecture.

Layer	Channels	Size	Kernel size	Stride	Padding
Input	3 (RGB)	224x224	-	-	-
2xConvolution	64	224x224	3x3	1	same
Max-Pooling	64	112x112	3x3	2	valid
2xConvolution	128	112x112	3x3	1	same
Max-Pooling	128	56x56	3x3	2	valid
3xConvolution	256	56x56	3x3	1	same
Max-Pooling	256	28x28	3x3	2	valid
3xConvolution	512	28x28	3x3	1	same
Max-Pooling	512	14x14	3x3	2	valid
3xConvolution	512	14x14	3x3	1	same
Max-Pooling	512	7x7	3x3	2	valid
Fully Connected	-	4096	-	-	-
Fully Connected	-	4096	-	-	-
Output	-	1000	-	-	-

TABLE II: VGG-16 architecture

The network comes with downsides and advantages. Firstly, the network is quite deep, which makes it computationally expensive. In fact, the original network has been trained for 2 to 3 weeks [9]. On the other hand, the model is appealing due to its simplicity, as many parameters remain unchanged, and the dimensions of the layers change in a very structured way. Each set of convolutional layers doubles the number of feature maps (channels), with the exception of the last one. In contrast, max-pooling layers shrink the height and width by half. The main idea behind the architecture is that the results achieved by large kernel sizes inside AlexNet can be replicated using many smaller kernel windows. The network consists of roughly 138 million parameters.

There are a few similarities between AlexNet and VGG-16. Such as the usage of ReLU as an activation function or the pattern of increase in the number of channels and decrease in the size of input as we go deeper in the network. Despite those analogies, there are several differences. To begin with, the dimensions of the input image are different. Next, VGG-16 is a much deeper network with 16 trainable layers, compared to 8 weight layers in AlexNet. It was possible due to the smaller size of convolutional filters in the former network. Lastly, the number of parameters in VGG-16 is more than double of AlexNet, which makes it more computationally expensive and prone to overfit, but VGG-16 tends to perform better.

III. Experiment/Results

Now, as we have seen the technical specification of two networks, we can apply them to solve real-world problems. We will look into pneumonia detection using a chest x-ray dataset from Kaggle (https://www.kaggle.com/datasets/alifrahman/chestxraydataset). The dataset is already split into train and test sets, containing two directories, "NORMAL" and "PNEUMONIA". We will further split the train set using an 80-20 proportion to get the validation set. We will apply both AlexNet and VGG-16 frameworks, one after another, and then compare their results.

A. AlexNet

Firstly, after loading the data, we resize it to desired scale (227x227 in AlexNet). Then, the image is an array of pixels from 0 (black) to 255 (white), which we then normalise by dividing by 255. Next, we build the network architecture using existing methods from the Keras library for convolutional, max-pooling and dense layers. As an activation function in the output, the sigmoid function has been chosen as our classification problem is binary. We set binary cross-entropy as the loss function that training tries to minimise. Adam's algorithm was picked as our optimiser. According to Keras's documentation, a stochastic gradient descent method uses first- and second-order moments. The model's training plot on the dataset is shown in Figure 4.

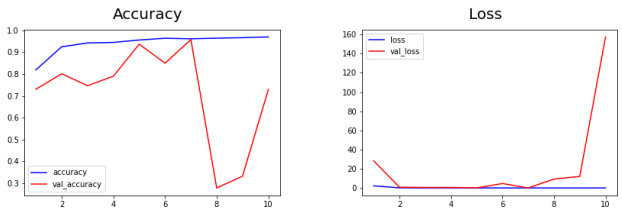

Fig. 4: Error metrics after each training epoch of AlexNet

10 epochs were enough to achieve satisfactory results on the training set. However, we can observe abnormal behaviour after epoch 7. Essentially, our validation accuracy drops to 30%, and validation loss spikes up to 160. We are not entirely sure why this occurred as the dropout layer with probability 40% has been applied to avoid overfitting. The possible explanation could be too big of a learning rate. However, the default one (0.001) was picked. Another reason could be the nature of batch training. We used a model checkpoint at an epoch with the highest validation accuracy to overcome this issue. This resulted in the final model being the one at epoch 7 with training accuracy=96.1%, validation accuracy=95.7%, training loss=0.106 and validation loss=0.106. Now we use

our model to predict the test set. The results are satisfactory, with accuracy being 73.1%, precision 70.3% and recall 98.5% (full results with comparison in Table III).

B. VGG-16

VGG-16 requires an input image of size 224x224. We normalise the data set after we shrank it. However, we will take a different approach to building this model. Instead of replicating the architecture from scratch, we will take advantage of transfer learning. Consequently, we can facilitate our learning process by using weights from the original model trained on the ImageNet database with 1.5 million images. We can copy all the weights up until the output layer, which we train using our dataset. Hence, not only will our training time decrease, but our accuracy will likely increase. The results of training are presented in Figure 5.

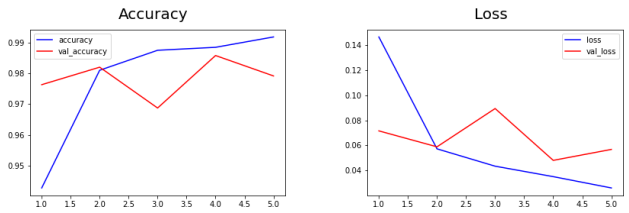

Fig. 5: Error metrics after each training epoch of VGG-16

As we can see, the training showed high accuracy from the first epoch, likely due to transfer learning. We did not experience any unexpected drop in validation accuracy as in AlexNet, and the training went smoother. The results on the test set were also favourable with accuracy=83.3%, precision=79.2% and recall=99.5% (look Table III).

	AlexNet	VGG-16
Train Accuracy	0.961	0.988
Validation Accuracy	0.957	0.986
Train Loss	0.106	0.035
Validation Loss	0.106	0.048
Test Accuracy	0.731	0.833
Precision	0.703	0.792
Recall	0.985	0.995
F1 Score	0.82	0.882
Time per epoch	2 min	15 min

TABLE III: Comparison of AlexNet and VGG-16 models

Clearly, VGG-16 performed better not only during validation but also on the test set. However, AlexNet had one advantage, which was faster training. Overall, the comparison result is as expected since VGG-16 is a more recent and much deeper network. Another thing in favour of VGG-16 was the usage of transfer learning. Nevertheless, both networks have test scores above 70% and show the power and potential of CNN in computer vision tasks.

IV. Discussion

From the above discussion, we can conclude that we studied the building blocks of a CNN and two of its architectures. CNN includes three layers: convolutional layers, pooling layers, and fully connected layers. We also analysed two architectures: AlexNet and VGG-16 model, which are useful for image classification. We applied these to the real-world dataset of chest X-rays for Pneumonia detection, either viral or bacterial, with no difference in their class. CNN performed image extraction, reducing the input's dimensionality for the fully connected layers. If we applied a normal neural network, the accuracy of the validation set would suffer because the algorithm would overfit the train data.

Both AlexNet and VGG-16 architectures have their pros and cons when applied to the data set. As the number of parameters in VGG-16 is much greater than AlexNet, VGG-16 is computationally expensive, making the training duration much longer. But when we compare the error metrics and accuracy of both the models, the VGG-16 model performed better in all the metrics other than duration. It also needed fewer epochs to reach a high accuracy score, but it is due to the transfer learning that we implemented. This wasn't possible for AlexNet since it was missing in Keras Library. However, both models showed validation accuracy above 70%, showing their excellent performance. We can potentially improve the accuracy by applying more recent architectures, such as ResNet. We could have also applied some data augmentation to increase the accuracy or even used an improved version of the ReLu function such as Leaky ReLU function.

V. Acknowledgements

For coding, we used a youtube tutorial (https://www.youtube.com/watch?v=jztwpsIzEGc) to learn how to properly load the data using keras utilities and plot the results. For the implementation of AlexNet, we took inspiration from an article (https://thecleverprogrammer.com/2021/12/13/alexnet-architecture-using-python) in how specific methods should be used. For transfer learning code, we used https://www.youtube.com/watch?v=lHM458ZsfkM to learn how to implement it. We have also used the following Python libraries:

- Pandas
- Numpy
- Tensorflow
- Keras
- Visualkeras

References

[1] R. Szeliski, *Computer vision: algorithms and applications*. Springer Nature, 2022.

[2] P. Picton, "What is a neural network?" in *Introduction to Neural Networks*. Springer, 1994, pp. 1–12.

[3] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*. Ieee, 2017, pp. 1–6.

[4] P. Sermanet, S. Chintala, and Y. LeCun, "Convolutional neural networks applied to house numbers digit classification," in *Proceedings of the 21st international conference*

on pattern recognition (ICPR2012). IEEE, 2012, pp. 3288–3291.

[5] E. W. Weisstein, "Convolution." *From MathWorld–A Wolfram Web Resource.* [Online]. Available: https://mathworld.wolfram.com/Convolution.html

[6] I. Goodfellow, "Convolutional Networks," in *Deep Learning*. [Online]. Available: https://www.deeplearningbook.org/contents/convnets.html

[7] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*. Ieee, 2017, pp. 1–6.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3065386

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. [Online]. Available: https://arxiv.org/pdf/1409.1556.pdf