Convolutional Neural Networks And Their Application Within Image Categorisation

J. Burns (36017853), W.Stafford (35804459), E.Webb (35955285), W.Wu (36101566)

***Abstract*— In this report we describe the motivation behind Convolutional Neural Networks (CNN) and proceed to explain the Mathematical detail behind the operations used in a CNN. We set out the benefits of using convolution within a Neural Network including sparse interactions, parameter sharing and equivariance. We then apply the CNN framework in `R` using the `keras` package to a data set containing images of clothing items; t-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag and ankle boot. We define a classification problem and achieve 93% accuracy on training data and 92% accuracy on test data. We conclude by giving examples of alternate Networks and other Machine Learning algorithms which can be run on various types of data.**

I. Introduction

Artificial Neural Networks (also referred to as Classic Neural Networks or NNs) are a popular choice for modelling complex, pattern-oriented, non-linear systems in the real world. Since their advent in the mid 1940s they have been used to tackle problems ranging from financial modelling [1] to medical imaging and signal processing [2]. These Classic Neural Networks can be trained on data sets where each observation is represented by an array of numerical and categorical covariates.

Due to their non-parametric nature, Classic Neural Networks are able to construct models without any prior knowledge of the distribution of the data or the interactions between the variables. [3] This makes them an effective alternative to the commonly used parametric statistical methods, particularly at performing tasks such as classification.

Although Classic Neural Networks perform well on many data sets, as the data becomes more complex NNs can be limited in their application. With that in mind, we are able to adapt the Classic Neural Network to tackle advanced data and perform the tasks we desire.

One such case where this is appropriate is when working with data that exhibits a grid-like topology, such as time-series (1 dimensional) or image data (2 dimensional). [4] A Convolutional Neural Network (CNN) builds on the Classic Neural Network by containing at least one *Convolutional Layer* to process the grid-like data.

There are a plethora of building blocks that go into constructing a Convolutional Neural Network, each with their own computational and statistical benefits or weaknesses. In this report, we discuss some of these components individually before combining them to apply to a real world example - image categorisation. Finally we elaborate further on components that could have improved our Neural Network and, subsequently, the trained model.

II. Methods

A. Convolution

We often want to explain our data by common patterns and structures - for grid-like data we do this through convolution. Convolution is a smoothing process that takes two functions, an *Input* and a *Kernel*, and expresses how the shape of one is altered by the other in the form of a third function - the *Feature Map* (Output). This produces a result that amplifies a certain feature within the data.

The general mathematical definition of convolution for continuous data is the integral of the product of two functions, f (the input) and g (the kernel) - where g has been reversed and shifted:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau. \tag{1}$$

We can think of the *Input* f as an impulse function, and the *Kernel* g as a response function, to gain the intuition behind shifting g. When we deliver a unit impulse, we get a response given by g. We could deliver several impulses with varying magnitudes at various different time points. If we then choose some arbitrary point in the future, we would get a combination of these shifted responses, each of which has its own respective weighting. [5]

In Equation 1, $g(t-\tau)$ is the value of g at time t if the impulse was sent at time τ. $f(\tau)$ is the magnitude of the impulse at time τ. Imagining f and g as functions of τ, we see that g has been reflected in the τ-axis - this is a concept known as *Kernel Flipping*. We can then calculate the value of the output function for all values of τ by integrating, leaving us with a function of t. This intuition is only true for time invariant response functions, g, i.e. g gives an identical response independent of when the impulse is sent.

In practicality, we often employ convolution with discretized grid-like data. The input is a multidimensional array of data (y-dimensional grid of data per observation x) and the kernel is a multidimensional array of parameters (a "pattern" we wish to identify). The multidimensional arrays are called *Tensors*.

Convolution is useful for telling us how two functions operate together, but for a CNN we wish to know how similar these two functions are i.e. treating our kernel as a pattern we wish to discover the presence of. As we discussed previously, in Equation 1 we have effectively flipped the kernel as we integrate over τ. If we do not flip the kernel, we can update Equation 1 for discretized data to be:

$$O(i,j) = (I*K)(i,j) = \sum_m \sum_n I(m,n)K(i+m, j+n). \quad (2)$$

where the output O is indexed from 0 (instead of 1), since it is referring to shifts. This operation is formally known as *Cross-Correlation* but is referred to as *Convolution* since its only difference is not flipping the kernel. Equation 2 indicates that when the two functions are identical the output will be a high positive value. Conversely, when the functions are exact opposites, the output will be at its lowest. The output function (also a discrete grid) will now be a summarised version of the input whereby its values tell us how similar the two original functions are. This is a useful result for CNNs.

B. Benefits Of Convolution

There are 3 distinct benefits to using Convolutional Neural Networks compared to a Classic Neural Network.

Firstly, a Classic Neural Network layer includes interactions between every input unit and every output unit. The matrix multiplication within the layer uses a matrix of parameters with a separate parameter describing each unique unit interaction. Therefore each output unit will interact with all of the input units, as illustrated in Figure 1 by the greyed out units in the bottom image. The input units that interact with the output unit are called a *Receptive Field*. If we had m inputs and n outputs then we require $m \times n$ parameters, producing a run-time of $T(m \times n)$, per example. [4] Convolutional Networks improve on this by using a kernel far smaller than the input, which limits the number of interactions each output can have to $k << m$. This is known a *Sparse Interactions* and is shown in the top image of Figure 1. The run-time is greatly reduced since $k << m$ and hence $T(m \times n) > T(k \times n)$. Since fewer parameters need to be stored, sparse interactions improve the statistical efficiency and reduce the memory requirements of the model. [4]

Convolutional layers also leverage *Parameter Sharing*. This means that all kernel values are used at every input position. Rather than learning different parameters for each input position, as is the case of Classic Network Layers, Convolutional Layers need to learn only one set of parameters - further reducing the storage requirements of the model. This concept can be seen in Figure 1 in bold red, where the top image exhibits the parameter sharing of a Convolutional Layer and the weight applied to Input 3 is tied to the weights across the other inputs. This contrasts to the bottom image, where the input weights are not tied (no parameter sharing).

This form of parameter sharing induces *Equivariance To Translation*, meaning that if the input changes, the output will change in an identical way. This means applying a transformation to the input, or the output, generates the same result as each other. With 2D image data, if we were to move a feature within the input, that feature will move by the same amount within the output. We can utilise this within the early layers of a CNN when we search for a function of neighboring pixels that could be identified at multiple locations - such as detecting edges. In this instance it is sensible to share parameters across the entire image.

Ultimately, these characteristics improve on the usability, statistical efficiency and computational memory requirements of Classic Network Layers.

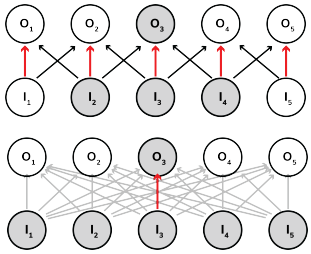

Fig. 1. Focusing on Output 3. (Above) Convolution Layer demonstrating Sparse Connectivity and Parameter Sharing. The output is formed by convolution using a 3×3 kernel, producing a receptive field of size 3 - shown in Grey. Red arrows correspond to central value in the 3×3 kernel, which is used at each location. (Below) Neural Network layer without Sparse Connectivity or Parameter Sharing. Output is produced through dense matrix multiplication, so the receptive field includes all inputs. Red arrow corresponds to central value in the weight matrix, which is used only once. Adapted from [4].

C. Convolutional Neural Network

Within the CNN framework we can define many different kernels and have multiple convolutional layers. The most efficient form of Convolutional Neural Network evolves regularly. Covering the many possible constituents of a CNN is beyond the scope of this report, however we discuss the key concepts and components of the CNN we later use in application.

A Convolution Layer consists of 3 main stages: Convolution Stage, Detector Stage and the Pooling Stage.

1) Convolution Stage: The convolution stage operates as described in Section II-A, with a couple of further nuances to consider. As previously outlined, it is good practice to use a kernel smaller than the input to reduce memory requirements. The drawback of this is that the features at the edge of the tensor have less representation within the model than those at the centre. Additionally, the output shrinks compared to the input, shrinking by a width one unit less than the width of the kernel, meaning there is a finite limit to the number of layers that can be used. This is known as *Valid Convolution*. To counteract this, *Same Convolution* (Figure 2) employs *Zero Padding*, appending extra zeros to the input tensor to ensure the output is of the same dimension and all features are captured. The optimum method lies somewhere between these alternatives and hence we use a blend of the two within our framework.

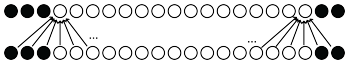

Fig. 2. Illustration of Same Convolution from one layer to the next, utilising Zero Padding. Adapted from [4].

The *Stride Length* of convolution can also be changed at the expense of feature distinction. The Stride Length refers to the

shift from one receptive field to the next during Convolution. This reduces the dimensions of the output tensor, but also reduces the computational cost. [6] An example of Stride with length 2 is shown in Figure 3, this halves the number of output units compared to inputs. To ensure computational and statistical efficiency, we use Strides of length 1 and 2.

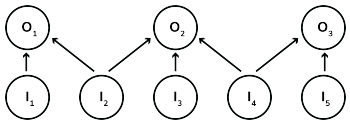

Fig. 3. Stride of length 2; Each output unit has a receptive field of 3 input units. Adapted from [4].

2) Detector Stage: As with all NNs, a Detector Stage using a Non-Linearity Activation Function (such as a *Rectified Linear Function*, ReLU) is used after each Convolution to prevent the model collapsing into a simple linear model. [7] ReLU takes the form:

$$R(x) = (x)_+ = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

This enables us to use stochastic gradient descent with back-propagation of errors to train our network.

3) Pooling Stage: Typically the final stage in a convolution layer, pooling functions are used to summarise certain regions of the feature map. Here we do not reduce the size of the feature map; we replace its values by evaluating local outputs. Popular functions include max pooling (Figure 4), average pooling, L^2 norm and weighted average pooling. This helps the network become less sensitive to slight adjustments to the input; we are more concerned with finding the presence of certain patterns rather than where they are located.

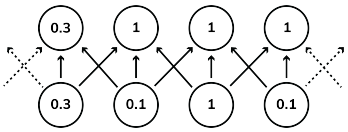

Fig. 4. Example of Max Pooling applied to the output of a Convolution layer. Adapted from [4].

III. Application and Results

A. Application Of CNNs To A Real Image Data Set

Now we use the methods previously explained in a practical example to show how CNNs can be used in image classification. The data set we are using is the Fashion MNIST data set, contained within the `keras` package in R. [8] It contains $70,000$, 28×28 pixel grayscale images, $60,000$ of which form our training set for the model and $10,000$ form the test set. The data are assigned to 10 categories, labelled $0-9$ within the training labels, with the following key:

$0 \rightarrow$ T-shirt/top $\quad 1 \rightarrow$ Trouser $\quad 2 \rightarrow$ Pullover

$3 \rightarrow$ Dress $\quad 4 \rightarrow$ Coat $\quad 5 \rightarrow$ Sandal

$6 \rightarrow$ Shirt $\quad 7 \rightarrow$ Sneaker $\quad 8 \rightarrow$ Bag $\quad 9 \rightarrow$ Ankle boot

We train a CNN with the intention to accurately predict the category of future images into one of the 10 categories defined above. We will compare the performance of this CNN with a NN also trained on this data. [9] Before we begin training the model we must firstly process the data. Each pixel takes a grayscale value between 0 and 255, as can be seen in Figure 5.

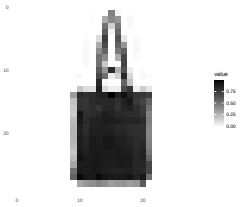

Fig. 5. Representation of final image in Fashion MNIST test dataset (label = 8).

We scale each pixel value down from $0-255$ to the range $0-1$, before training our CNN and then we train our model. The make up of the model is as follows, inspired from [10]:

Layer 1

- *Stage 1. Convolutional:* Input Size= 28×28, Padding=Same, Kernel size = 3×3, stride $= 1$, Output Size= 28×28.
- *Stage 2. Convolutional:* Input Size= 28×28, Padding=Valid, Kernel size = 3×3, stride $= 1$, Output Size= 26×26.
- *Stage 3. Max Pooling:*Stride = Pool Size = 2, Output $= 13 \times 13$.
- *Stage 4. Dropout:* This stage takes the last output and randomly changes 25% (chosen value) of pixel values to be changed to 0 and scales all other entries up by $\frac{1}{1-0.25}$ $= \frac{4}{3}$ in order to maintain the sum over all inputs. This is for the purpose of avoiding **overfitting** for our model.

Layer 2

- Takes the output from Layer 1 and runs the same layer outlay again, except for the dimensions will now be different as a result of Layer 1 and in Stage 1, we now have a stride of 2.

Layer 3

- *Stage 1. Flatten:* This stage takes the input from Layer 2 and changes it from a 2-dimensional input to a 1-dimensional output.
- *Stage 2. Output:* This final stage performs one final dropout stage before assigning each image a probability of being in each of 10 groups, the 10 assigned to the data.

We train our model on the training data, and set a value of 30 epochs, which means we run over our data 30 times, updating the model after every epoch, and see how our model performs.

B. Results

As can be seen in Figure 6, the accuracy increases and loss decreases as we continue to run the model over our data, which

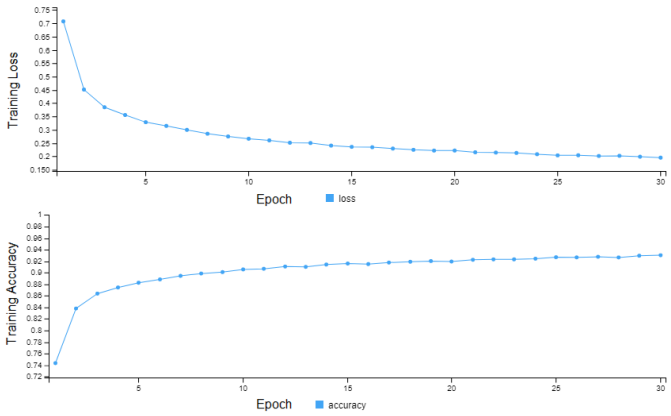

Fig. 6. Loss and accuracy of model during training period

we expect to see. The maximum accuracy attained is 0.9303 (4 s.f) and the minimal loss achieved is 0.1956 (4 s.f), which are both attained after the final epoch. We check how the model performs on our test data, and find the loss to be 0.2208 (4 s.f) and the accuracy to be 0.9205 (4 s.f). There is a slight drop in performance on both metrics from the training stage, which implies our model is slightly overfitting our training data. This is despite the fact we include dropout stages to attempt to avoid this issue. However the loss is low enough, and accuracy is high enough, on the test data to imply our model is still performing well. Furthermore, this means our CNN performs better than the NN trained on this data [9], implying the CNN was a better approach to this problem.

We can use the model to also make predictions for individual images. Let us take the same image as in Figure 5 and see what the predicted values of the image being in each of the 10 categories are. Clearly we expect to see the highest probability for label 8 (bag).

The output vector of probabilities is $(3.627 \times 10^{-4}, 6.828 \times 10^{-9}, 4.561 \times 10^{-7}, 2.677 \times 10^{-6}, 7.816 \times 10^{-8}, 1.187 \times 10^{-11}, 6.792 \times 10^{-7}, 2.009 \times 10^{-3}, 9.996 \times 10^{-1}, 4.368 \times 10^{-8})$(4 s.f).

As stated earlier, all these values sum to 1. The largest value is the 9th value, at 0.9996 (4 s.f), and as our labels were 0-indexed, this corresponds to label 8 (Bag) which is the intended outcome. This application of CNNs is useful in settings such as categorizing online items on sales websites, for example helping eBay suggest categories items should be listed under based on the product picture. This could also be used in simple image recognition for search engines such as Google.

IV. Discussion And Conclusion

Our CNN contains 2 convolutional layers, whereas ideally for a problem like this with 10 possible outcomes, we would want to have many more. Due to time constraints, we think that using 2 layers for a return of 92% test accuracy is sufficient to negate the need for further layers. Although this would most likely result in a higher accuracy, the extra time needed to run a more complex model is not justified in the context of this report. The total time taken to run the model training was 2 hours and 9 minutes. Each epoch on average took 258 seconds, so a further recommendation would be to not run as many epochs to train the model. Once again the extra time required for each epoch towards the later epochs is not resulting in a significant increase in performance. For example the 16th epoch results in an accuracy of 0.9201 (4 s.f) and a loss of 0.2198 (4 s.f) so, for an extra hour of training after this point, the model only increases by 0.0102 in accuracy and decreases by 0.0242 in loss. Therefore, depending on time available and cost, it may be worth the slight decrease in performance to run less epochs.

In our application we use single channeled data in the form of grayscale images, rather than coloured images with multiple channels (such as Red, Green, Blue). This means we only require one kernel per feature, compared to three when using RGB (Tiled Convolution). Whilst grayscale is more computationally efficient, a Tiled Convolution approach would have been better at identifying features and could have provided a better result. Additionally, we use convolution in series although it is often good practice to perform convolution in parallel. Convolution in series involves using only one kernel at a time and hence can only extract one feature across the input tensor. Convolution in parallel is able to extract many features in unison which would improve the accuracy of our model but would come at a computational cost. Overall, our model is suitable to our time and cost requirements.

Whilst convolution utilises the equivariance property, this does not extend to transformations such as rotation of the image or change of scale. [4] An alternative that addresses this are Graph Neural Networks (GNNs). Because GNNs represent the structure of an image with edges and vertices, the model training process is invariant to rotation. [11] In our application, we don't use rotated images. If we were to use rotated images in a further study, applying these two methods in parallel (Convolutional Graph Neural Networks, ConvGNNs) would compensate for this issue. [12] The GNN component is used to model the relationships of the objects identified by the Convolution component. [13] However, a limitation of Neural Networks, including CNNs and GNNS, is the so-called "black box" issue. With the significant investment into Neural Networks over recent years, their designers are understanding less and less about how their model is actually reaching the outcome it provides. [14]

CNNs are useful tools, however they do have drawbacks, such as the inability to deal with rotations effectively. Despite this, they do offer increased accuracy and performance, as shown using the MNIST data set to train a model using a CNN compared to a NN. We have demonstrated how CNNs can be effective when classifying images, achieving a model accuracy of 93.03%. For other applications, different components can be used to alter the statistical efficiency of image classification models at the expense of computational requirements and time. Overall, CNNs are flexible tools that can be adapted to model different types of data, with a grid-like topology, based on the constraints of the model designers.

References

[1] A. Yona, T. Senjyu, A. Y. Saber, T. Funabashi, H. Sekine, and C.-H. Kim, "Application of neural network to 24-hour-ahead generating power forecasting for pv system," in *2008 IEEE Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century*. IEEE, 2008, pp. 1–6.

[2] A. Miller, B. Blott *et al.*, "Review of neural network applications in medical imaging and signal processing," *Medical and Biological Engineering and Computing*, vol. 30, no. 5, pp. 449–464, 1992.

[3] S. Walczak, "Artificial neural networks," in *Encyclopedia of Information Science and Technology, Fourth Edition*. IGI Global, 2018, pp. 120–131.

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

[5] S. W. Smith, *Digital signal processing : a practical guide for engineers and scientists*, 1st ed., ser. Demystifying technology series. Amsterdam ; Boston: Newnes, 2003.

[6] A. Géron. (2017, Apr.) Hands-on machine learning with Scikit-Learn and TensorFlow concepts, tools, and techniques to build intelligentsystems. Paperback. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1491962291

[7] G. J. W. T. H. Tibshirani, *An Introduction To Statistical Learning: With Applications In R*. Springer, 2021.

[8] "Keras Documentation." [Online]. Available: https://cran.r-project.org/web/packages/keras/keras.pdf

[9] "Classic Neural Network Used On MNIST Data." [Online]. Available: https://tensorflow.rstudio.com/tutorials/beginners/basic-ml/tutorial_basic_classification/

[10] L. Francisco, "Convolutional Networks in R." [Online]. Available: https://www.r-bloggers.com/2018/07/convolutional-neural-networks-in-r/

[11] D. Parikh, "Alternatives to CNN (Convolutional Neural Network)." [Online]. Available: https://iq.opengenus.org/alternatives-to-cnn/

[12] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[13] S. Hong, "Understand What GNN Is and What GNN Can Do," 2020. [Online]. Available: https://towardsdatascience.com/an-introduction-to-graph-neural-network-gnn-for-analysing-structured-data-afce79f4cfdc

[14] S. Snapp, "The Problematic Black Box Nature of Neural Networks and Deep Learning," 2020. [Online]. Available: https://www.brightworkresearch.com/the-problematic-black-box-nature-of-neural-networks-and-deep-learning/#:~:text=Neural%20networks%20and%20deep%20learning%20are%20normally%20black,solution%20is%20unknown%20to%20the%20users%20or%20observers.