The Generation Game: An image processing system for procedural game generation using a convolutional deep learning architecture

Nathan Butt
16013327

University of the West of England

May 29, 2019

Image recognition is a foundational technology whose application can be witnessed in many games as well as across other fields such as computer graphics. In recent years with the advent of deep learning, convolutional neural networks are being used as a potential solution to many image recognition problems due to the potential accuracy and flexibility. One area where these could be applied is in procedural generation where pictures could be used to create levels or game mechanics. Here we propose a possible involving the use of convolutional neural networks in order to generate elements of a game based on a set of predefined parameters.

1 Introduction

There are numerous examples of image recognition being used in games. One example is EyeToy Play 3 (SIE, 2005) which contained a feature known as "RecogCam" where players could use select images in order to access secret content within the game. Whilst an enhancement to the game, the system was limited in that is required a pre-set range of images, limiting gameplay potential and requiring more space to store the required image data.

Machine learning is a family of algorithms that has seen widespread development and adoption in recent decades, particularly with regards to artificial neural networks, which are being applied in numerous sectors to solve a wide variety of problems including image recognition. The primary strength of these algorithms is the ability to learn tasks from training data as the name implies. This makes them very adaptable to the needs of the individual providing the correct data and architecture is used. In regards to examples such as the EyeToy, this would both solve the data problem as well as open gameplay opportunities as the game could contain procedural elements based on the image presented.

This report seeks to evaluate this possibility via creating a simple game which allows individuals to select classes of random images and generate a game which can then be played. Image processing will be performed via a neural network which will be trained using a selection of images where the results of said network are used to generate the game. This game will be evaluated on both the success of the task and the performance of the image recognition.

Given the need to interoperate images, we can consider this a multi-class image classification problem. Therefore the neural network for this system will utilise a convolutional neural network architecture (CNN).

Figure 1: *An Example of the recogCam cards used in the image recognition system (SIE, 2005)*

2 Related Work

CNNs were first presented by Lecun et al., 1998. and are a deep-learning neural network structure designed for image analysis via the use of convolutional filtering. This architecture introduces a number of new layer constructions and operations to neural networks in order to facilitate this. Additionally, it has resulted in swathes of new method pertaining to loss and activation functions.

2.1 Convolutions and Convolutional Layers

Fundamental to a CNN is the concept of convolutions which are resulting integral functions created as a product of two other functions. Given the resulting function is an integral it defines the area under the resulting curve. This function can also be defined discreetly resulting in equation 1 (Smith, 1997).

$$(f \times g)[n] = \sum_{m=-\infty}^{+\infty} f[m]g[n-m] \qquad (1)$$

This function is optimal for images can be represented in functional definition via the use of matrices which can then be substituted in to allow for the application of various functions ranging from Gaussian curves to edge detection allowing either the image to be edited or features to be extracted from the image for training (Lecun et al., 1998).

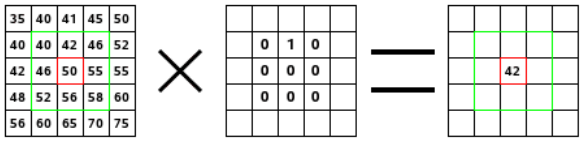


Figure 2: *An example image convolutional operation. (*Convolution Matrix*)*

Application of convolutions to images is performed via a convolutional matrix or image kernel which is an N x N matrix defining a select filtering operation as demonstrated in Figure 2. The image can also be thought of as a width*height matrix but given the difference in magnitude the image kernel is applied across the image with a striding operation performed across the image. The resulting substitution can be expressed as follows.

$$(f \times g)[x, y] = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \omega[s, t] f[x-s, y-t] \qquad (2)$$

Convolutional layers perform this exact operation with the image kernel portion $\omega(s, t)$ representing the weights of the layers. These are adjusted during the networks training process which in turn adjusts the features extracted from the image allowing the model to be trained. Convolutional layers are comprised of a node count that equates to $colourDepth * imageWidth * imageHeight * imageCount$ representing all aspects of the image including the seperate channels.

Additionally, the image weight matrices must be defined are defined via a 2D vector along with a stride value which determines the movement of the image kernel across the collection of image nodes.

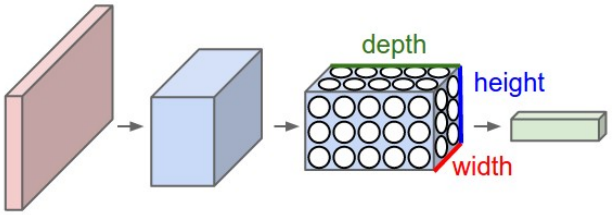


Figure 3: *A visual summery of a nodes composition (*Covolutional Neural Networks*)*

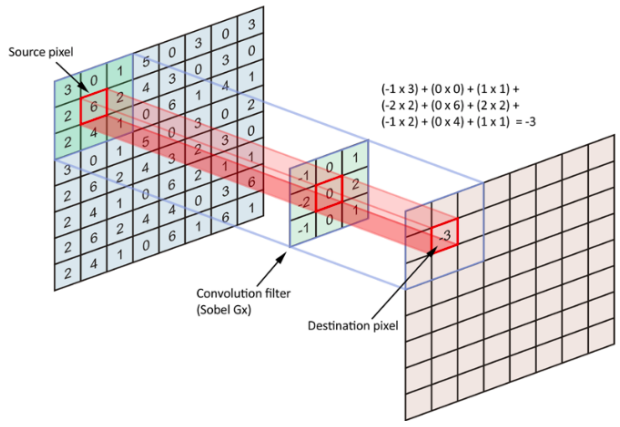


Figure 4: *A visualisation of a convolutional layer operation (Cornelisse and Cornelisse, 2018)*

2.2 Additional Layers

Layers are collections of nodes designed to abstract aspects of the neural networks problems. In CNNs there exists a number of unique layer constructs pertaining to image processing given the nature of the data. Additionally, there are several other general layer constructs that can be used to improve accuracy.

The nature of convolutions results inevitably in multi-dimensional data which cannot be directly mapped to a list of categories which is a single dimensional dataset. This is addressed via a flatten layer which is a fully connected layer where all nodes in the previous layer are connected directly to a set of nodes, resulting in a collection of 1D values. This a CNN classifier this is used to retrieve output from the network or to allow for use of fully connected layers.

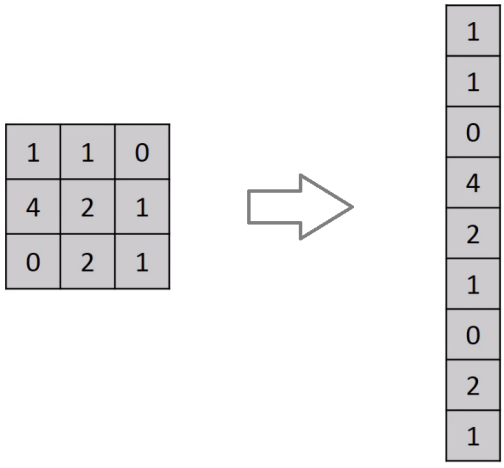


Figure 5: *A basic visulisation of a flatten layer. (Rubikscode, 2018)*

One issue with networks is overfitting which can result in poor performance given generic data, one cause is due to what is known as sampling noise resulting from limited datasets. A layer construct known as dropout addresses this by deactivating a specified proportion of nodes as described in 3 (Srivastava et al., 2014). This is a highly effective strategy resulting in major decreases in error.

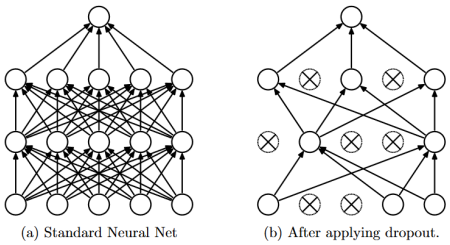


Figure 6: *A dropout layer compared with a standard hidden layer. (Srivastava et al., 2014)*

$$
\begin{aligned}
r^l &= Bernoulli(p) \\
y_i^{l+1} &= r^l * f(x) \\
x &= inputFromConnectedLayers
\end{aligned}
\tag{3}
$$

Pooling layers analyse various elements of a convolutional layers via a defined grid filter which fetches values from the convolutional nodes and retrieves a value according to a select “Pooling operation”(Gu et al., 2018). The two primary types of pooling are max pooling where the highest value within the grid filter is retrieved and subsample pooling. These function similar to dropout layers in that they are designed to reduce error from over-fitting from convolutional layers.

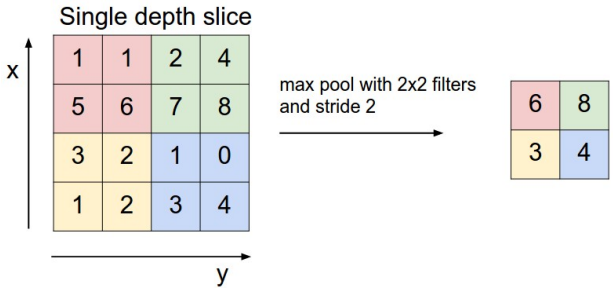


Figure 7: *A illustration of a pooling layer utalising max-pool.* (Covolutional Neural Networks)

CNNs tend to utilise the max pooling operation as it has been demonstrated to result in lower error rates (Scherer, Müller, and Behnke, 2010).

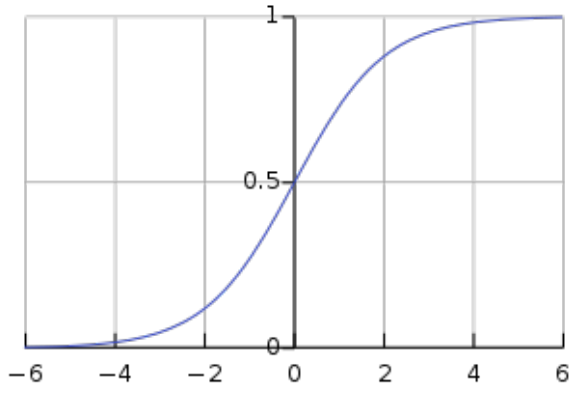
2.3 Nodes and Activations

Within a CNN, nodes function identically to nodes in a standard neural network where nodes outputs values after being subject to activation. Activations are performed on a weighted sum of all inputs to the node and can be defined by equation 4. Activation functions are arbitrary defined which also extends to the function ranges.

$$f(\sum_{i=0}^{Nodes(l-1)} w_i^{l-1} x_i^{l-1} + b^{l-1}) \quad (4)$$

The most basic activation is a linear/identity activation where input values are passed through with no alteration ($f(x) = x$) whose application in a network is according to how optimal the function is for the networks tasks. Given the nature of the network, the activations most adapt are those used for convolutional operations and classification.

Classification problems commonly utilise the sigmoid or logistic activation function. This function produces an s shaped curve with an output range of 0-1 which can be seen in the equation below. Sigmoid activations are ideal for classification due to the function range. The range maps to probability values correctly, combined with the function having differentials and it's a very suitable candidate for use in artificial neural networks.



$$f(x) = \frac{1}{1 + e^{-x}} \quad (5)$$

Figure 8: *A basic description and demonstration of sigmoid (*Sigmoid function *2019)*

However when performing multiple category classifications this activation function performs poorly as this is a binary function where activations may total above 1, meaning it is only suitable for activation between two categories. Therefore multi-classification functions utilise an activation called softmax activation (Duan et al., 2003). Softmax activation is a mathematical extension of sigmoid where binary classifications are performed across all categories and combined together, making it suitable for use in multiple classification scenarios.

$$f_i(x) = \frac{e^{x_i}}{\sum_{j=0}^{J} e^{x_j}} \quad (6)$$

Convolutional neural networks utilise a specialised family of activation functions known as Rectified linear unit (ReLU) activations. There are several variants of ReLU, however the fundamental version is defined by equation 2.3 (Nair and Hinton, 2010).

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

ReLU activations have been demonstrated to be highly effective in image classification (Nair and Hinton, 2010) hence their widespread support and use.

2.4 Loss and training

Training in mathematical terms can be interoperated as an optimisation operation with all predictions existing in a search space. Training in a CNN is identical to existing feed-forward neural networks (Lecun et al., 1998), comprised of several stage:

- Calculation of loss/cost
- Backpropogation of error
- Optimisation

Loss or a cost function (C) is a method for determine the error of a network of which there are several according to the network architecture. Multi-category Classification networks commonly utilise multicategorical cross-entropy cost. Cross-entropy is also known as log loss and utilises logarithms to estimate error. This function is defined by equation 7 (Lin et al., 2017) (*Loss Functions*)

$$C(p_t) = -\sum_{c=1}^{M} \alpha \log p_t \quad (7)$$

Cross entropy can also be modified according to whether or not the output categories are one hot categories where outputs can have multiple properties. In the case of single category output sparse categorical optimisation can be used (20 – check for better source).

After calculating loss the contribution to loss must be determined and in all cases this is performed via backpropagation. Backpropogation is a delta function which distributes error throughout the neural network which is later used as part of optimisation. The goal of this algorithm at each stage is to find the partial derivatives of $\partial C_x / \partial w$ and $\partial C / \partial b$ (Nielsen, 2018). These differentials are calculated via application of the chain rule as shown in equation 9.

$$\frac{dy}{dx} = \frac{du}{dx} \times \frac{dy}{du} \quad (8)$$

This application results in the derivative allowing the gradient to be quantified and then subsequently carried through to earlier nodes.

$$\frac{\partial C_x}{\partial w_x} = a_{in} \delta_{out} \quad (9)$$

This proceeds recursively through the network informing the optimiser the gradient of error, allowing gradient descent to proceed.

Optimisation represents the core of the training process via adjusting the weights in the neural network in order to locate minimum costs which in a neural networks case equates to greater accuracy. In neural networks this is completed via gradient descent where the differential is computed that represents a movement in the search space from the initial cost function. There are several optimisers which are derived from the same principle such as RMSprop, AdaGrad. However one approach now commonly used is the Adam optimiser as it has been demonstrated to correct loss significantly more efficiently than other optimisers (Kingma and Ba, 2014). Optimisers contain a number of hyper-parameters, the core of them being the learning rate which is a multiplier that determines the rate of ajustment to weights.

3 Method

This system was designed with two components, the game and the neural network module. Games will be constructed from a list of predefined categories with these categories being used during the training phase to train a predefined neural network how to recognise images. During runtime the game will pass selected images to the neural network module which will return a prediction. This prediction is then used to construct the game.

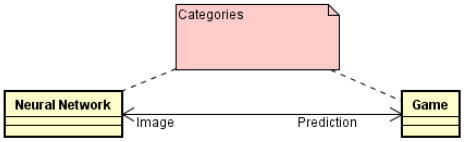


Figure 9: *The architecture of the system.*

Categories are defined within a text file and are comprised of a list of strings representing each category. This list is identical for both the game and the NN and is used for the respective data structures in each.

3.1 Game

The game was created using the Unity3D game engine. The game several scripts to both interface with the NN and create the games. The games were defined by two feature categories – The character and the map, this informed the list which was comprised of five items.

Image input in the game comes from two sources either via loading directly from the user's filesystem or from a local webcam where the user can take a picture. These files are stored in the games local filesystem

Class	Feature Category
Anime	Characters
Scifi	
Skeletons	
City	Maps
Forest	

where piping to the neural networking module is used utilising the Process functionality in Unity. Upon the neural networking module issuing a prediction it is then retrieved in the form of an array where the probability of each category is recorded.

These predictions are scanned by feature category with the highest prediction for an item in a feature category being used. Once recognised the game will then be configured according to the selected categories. So if a human character for example is shown then you would play as a human.

3.2 Neural Network module

The NN module was created using Google's Tensorflow API utilising the Keras subsystem within a collection of Python Scripts. Tensorflow incorporates all of the NN components and principles as discussed in related work in an assemble manner, making it ideal for this purpose (Goldsborough, 2016).

The neural network module is comprised of two individual scripts, a training script and a prediction script. All input data to the neural network is required to be formatted in the form of a 4D tensor or multi-dimensional data structure. The loading of data and processing into a tensor is performed by a DataLoader script.

The training script defines the neural networking model and performs neural network optimisation resulting in an output model via the SaveModel functionality within Tensorflow. The prediction script loads the corresponding model performing and outputting a prediction. These scripts were created according to recommendations from the Tensorflow Documentation.

4 Architecture Design

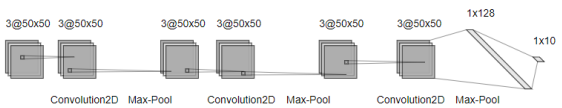


Figure 10: *A visual summery of my Neural network architecture*

The network architecture was a custom design based on the existing research performed. This architecture is comprised of three convolutional layers with each convolutional layer followed by a pooling layers allowing

for feature sampling and performing down sampling on the data. This is then followed by a dropout layer and a flatten layer which is designed to reduce overfitting from feature maps for category predication.

This is performed by two standard fully connected layers interconnected with flatten layers to allow for output to a select range of categories. It's this output that will then be fed to the game which is outputted in the form of a standard out.

All convolutional layers along with hidden nodes use ReLU activation with the final output layer utilising softmax activation given there recognised effectiveness for both convolutions and multi-class output. Given the stated performance the network was optimised using the Adam optimiser with a learning rate (Lr) set at 0.01 in order to avoid excessive gradient descent preventing overfitting and local minimums. The loss function given the need for multi category classification is a sparse cross entropy function.

5 Evaluation

5.1 Parameters

The primary focus in this case was the NN system as its predictions would have an impact on the behaviour of the main game. As this was going to be used for the game the network was trained using an identical set of parameters:

- 100 epochs
- The already mentioned set of categories
- A training set of 5000 images

The network was evaluated on both loss and accuracy metrics. Five individual training sessions were performed in order to test for any anomalies in the network.

6 Results

The networks approached proved to be very effective in the five tests that were performed. Convergence in both accuracy and loss was fairly quick with accuracy approaching above 90% within the allotted set of epoch. This was also true for minimisation of the loss functions which showed a similar fast rate of convergence as demonstrated in Figures 11 and 12.

However given these results it is possible that there may be an issue of premature convergence. Despite these issues this approach of interfacing between applications was fully functional and resulted in a stable working system.

7 Conclusion

In conclusion a successful implementation of a procedural game that is capable of recognising images and

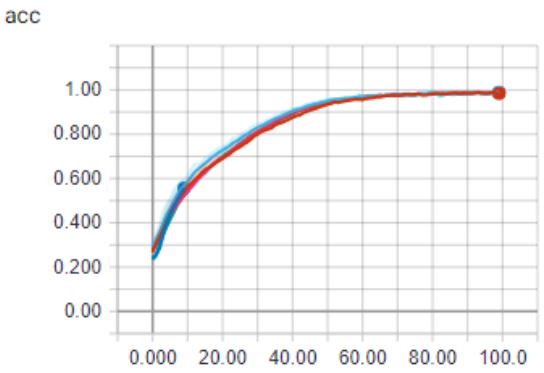


Figure 11: *Accuracy across the 100-epoch 5-run training*

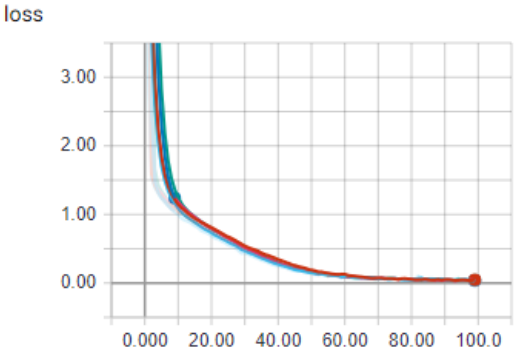


Figure 12: *Loss across the 100-epoch 5-run testing*

generating a subsequent game. The neural network architecture in many respects proved effective with a fast convergence rate. However this approach whilst effective could be enhanced with some additional changes to the activation function. As was found in research if multi-class crossentropy had been used its likley it would have resulted in more nuanced predictions as the loss function used was designed more for one-hot predictions where one preiction is chosen.

Bibliography

Cornelisse, Daphne and Daphne Cornelisse (2018). *An intuitive guide to Convolutional Neural Networks*. URL: https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050.

Covolutional Neural Networks. URL: http://cs231n.github.io/convolutional-networks/.

Duan, Kaibo et al. (2003). "Multi-category Classification by Soft-max Combination of Binary Classifiers". In: *Proceedings of the 4th International Conference on Multiple Classifier Systems*. MCS'03. Guildford, UK: Springer-Verlag, pp. 125–134. ISBN: 3-540-40369-8. URL: http://dl.acm.org/citation.cfm?id=1764295.1764312.

GIMP. *Convolution Matrix*. URL: https://docs.gimp.org/2.8/en/plug-in-convmatrix.html.

Goldsborough, Peter (2016). "A Tour of TensorFlow". In: *CoRR* abs/1610.01178. arXiv: 1610.01178. URL: http://arxiv.org/abs/1610.01178.

Gu, Jiuxiang et al. (2018). "Recent advances in convolutional neural networks". In: *Pattern Recognition* 77, pp. 354–377.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Lecun, Y. et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. ISSN: 0018-9219. DOI: 10.1109/5.726791.

Lin, Tsung-Yi et al. (2017). "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988.

Loss Functions. URL: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html.

Nair, Vinod and Geoffrey E. Hinton (2010). "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, pp. 807–814. ISBN: 978-1-60558-907-7. URL: http://dl.acm.org/citation.cfm?id=3104322.3104425.

Nielsen, Michael A. (2018). *Neural Networks and Deep Learning*. misc. URL: http://neuralnetworksanddeeplearning.com/.

Rubikscode (2018). *Introduction to Convolutional Neural Networks*. URL: https://rubikscode.net/2018/02/26/introduction-to-convolutional-neural-networks/.

Scherer, Dominik, Andreas Müller, and Sven Behnke (2010). "Evaluation of pooling operations in convolutional architectures for object recognition". In: *International conference on artificial neural networks*. Springer, pp. 92–101.

SIE (2005). *EyeToy Play 3*.

Sigmoid function (2019). URL: https://en.wikipedia.org/wiki/Sigmoid_function#/media/File:Logistic-curve.svg.

Smith, Steven W. (1997). *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego, CA, USA: California Technical Publishing. ISBN: 0-9660176-3-3.

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15, pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.