Deep Learning Binomial Expansions

Stuart Whipp
Capgemini
stuart.whipp@capgemini.com

June 1, 2025

Abstract

This paper aims to highlight mathematical capabilities of Neural Networks, and to showcase reusable, modular convolutional layers that may be applied to a wide range of problems. A system capable of binomial expansions is provided which leverages a neuron architecture exhibiting Pascal's triangle. A convolutional network without trainable weights is presented, along with an indexing mechanism which allows various patterns in the triangle to be output from the model such as Fibonacci, Triangle, Tetrahedral and Pentatope numbers as well as their higher order analogues such as Simplex numbers. A convolution layer performing cumulative sum enables an indexing method as well as Bernoulli's triangle as a summation of binomial terms in Pascal. This in turn yields famous sequences useful in mathematical analysis such as A000124 Lazy Caterer's Sequence, A000125 Cake Numbers and A000127 Dividing Circles into Areas. Finally, a convolutional network with 5 concurrent Softmax layers is trained upon images of latex equations, representing factorised binomials ready for expansion. This model may be integrated into the Pascal module to illustrate the usefulness of modular AI mathematics systems within a wider problem context.

1 Introduction

Neural Networks, despite their impressive capabilities in pattern recognition and data processing, face significant challenges when it comes to performing mathematical tasks. One major difficulty is their struggle with abstract reasoning and symbolic manipulation, which are crucial for solving complex mathematical problems. Their training is based on statistical patterns rather than exact calculations, and so even trivial computations aren't well suited to typical supervised learning approaches. A 2021 IEEE Spectrum feature '7 Revealing Ways AIs Fail' notes: "Neural networks can be disastrously brittle, forgetful, and surprisingly bad at math" [2]. The feature references a paper [3] in which Hendrycks and his colleagues attempt 12,500 high-school mathematics problems, with circa 5% accuracy. The paper's abstract notes "our results show that accuracy remains relatively low, even with enormous Transformer models" and "To have more traction on mathematical problem solving we will likely need new algorithmic advancements from the broader research community".

Large Language Models (LLM) have soared in popularity in recent years, with sites such as Hugging Face containing over 1 million models at the time of writing [4]. These models are typically transformer based, and work upon vectors representing tokenised words. LLM are often trained to predict the next words in a sequence, and consequently are highly performant in text completion, summarization and 'retrieval augmented generation' (RAG) tasks. LLM are common in natural language translation tasks, and they are increasingly used as 'code assistants' with tools such as Amazon Q (previously code-whisperer), GitHub Copilot and Google Gemini (formerly Bard). They do not perform computation upon inputs and therefore are not designed to conduct mathematical analysis. By incorporating a large corpus of mathematics in their training data, they may be capable of completing or retrieving formulae, however they will not directly compute solutions as aforementioned

1.1 Background

Pascal's triangle is named after the French mathematician Blaise Pascal, which he wrote about in 1654 'Traité du triangle arithmétique' [7]. The triangle was in fact referenced hundreds of years earlier in Persian and Chinese culture by mathematicians such as Al-Karaji and Jia Xian. Al-Karaji wrote a now lost book with the first formulation of the triangle and binomial coefficients, which was later developed by Omar Khayyám, leading to the name 'Khayyam's triangle' in Iran. Similarly, Jia's work was developed by a later Chinese mathematician and termed 'Yang Hui's triangle' though he cited Jia in his 'Jade Mirror of the Four Unknowns' [1].

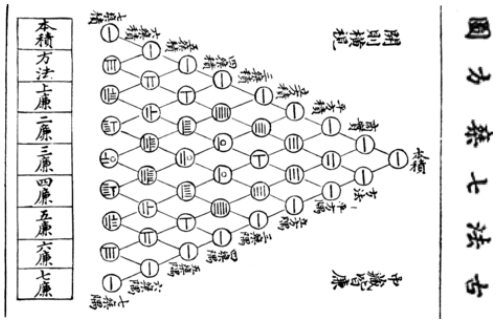

Figure 1: Jia Xian triangle from Jade Mirror of the Four Unknowns

The triangle is a mathematical construct with applications across algebra, combinatorics, and number theory. It is an infinite array of numbers arranged in a triangle where each number is the sum of the two numbers directly above it. The structure yields numerous patterns and properties, such as binomial coefficients, which are pivotal in algebraic expansions and probability theory.

The triangle may be formulated naively using perceptrons (neurons), with two inputs each consisting of unit weights and zero bias (constant additive term). In this analogy, only the number 1 is fed to the model, with any bordering neurons having just one input connection or else receiving zeros as a secondary input. In many ways, this constitutes the simplest possible Neural Network (NN) with only the model's structure giving rise to an abundance of mathematical patterns. The first column contains all natural numbers, with

preceding columns containing Triangular numbers, Tetrahedral, Pentatope and thereafter Simplex numbers. These figurate numbers describe the number of objects in equilateral triangles of specified length in 2D, 3D or higher-dimensional space. They are therefore integrally related to understanding areas, volumes and hypervolumes.

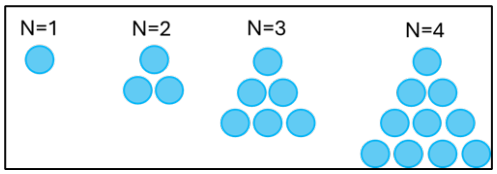

Figure 2: Triangle Numbers

The triangle numbers are shown above pictorially. If we stacked each of these as layers in a 3D volume, we would have Tetrahedral numbers i.e. the number of spheres within an equilateral Tetrahedron.

NNs have been around since the mid twentieth century. In 1958, Rosenblat's Perceptron described a single hidden layer of neurons with random weights [8]. Following what is sometimes termed an 'AI winter', a great deal of progress was made in the 1980s and 90s in training models with gradient descent. In 1986, Rumelhart et al popularised the use of backpropagation [9]. In 1991, Park and Sandberg showed that single hidden layers with Radial Basis activation could be Universal Function Approximators [6]. This was ground-breaking, but it also led to received wisdom that stacking layers was unnecessary, particularly due to 'vanishing gradient' making backpropagation with gradient descent impossible, when using saturating non-linear activations such as tanh and Radial Basis (an approximation of a normal distribution shape).

In 2012, AlexNet was released which helped popularise Deep Learning [5]. It consisted of convolution layers, wherein 'neurons' are locally connected according to a receptive field, and weights are 'shared' across output neurons of a layer. This greatly reduced weight parameters for large inputs such as images where each pixel is a NN input. Importantly, it also used ReLU activation (rectified linear unit) which is piece-wise linear and has no saturation characteristic.

Whilst convolutions are still common in Computer Vision, Recurrent models were most common in sequence modelling. This has since changed with Transformer models, since the seminal paper 'All you need is Attention' [14]. This heralded the age of Large Language Models, which tend to increase in parameters by 10x each year [11]. These are highly capable with natural language, treating words as tokens, in turn converted to sequences of vectors. Whilst they are incredibly powerful at a range of tasks, they are not performant with maths problems nor were they ever designed to be.

1.2 Motivations

Warren McCulloch, a cybernetician who contributed innovative work in the 1940s and 50s once said, "The only question I've wanted to know in my scientific life: What is a number that a man may know it, and man that he may know a number". [13]

The initial creation of Pascal's triangle with perceptrons and subsequently convolutions was formulated in MATLAB. It was discussed with tutors during an MSc in Applied AI, and presented to Junior Data Scientists in work, during tenure as a Lead Data Scientist there. It was seen as an interesting educational piece and served to illustrate how a Deep Learning model could be crafted rather than trained. In the intervening years, LLMs have grown increasingly deep with large computational resources needed to train and even inference them. This provided motivation to revisit this work with Tensorflow, enhance it and demonstrate an approach that combined a reusable Pascal-module, and learnable layers typical to a Computer-Vision use-case. The author of this paper hopes it is clear this may be taken much further, with many other combinatorial problems, and statistics applications. The model is fully explainable and readily auditable.

In the Conclusion section of this document, we evaluate the usefulness of this approach, versus integrating conventional learnable AI systems, with dedicated external functionality such as calculators or equation solvers. In short, the author believes that demonstrating NN can perform math-

ematics has inherent scientific value and should future model architectures exist such as Neuromorphic computing [15, 10] for mainstream use of AI, this methodology works perfectly well on neuronal-architecture and may therefore be embedded within AI systems as opposed to requiring external dependencies.

2 Methodology

2.1 Convolution Approach

When performing convolution operations in NN, it is typical to have values initialised with pseudo-random numbers, and have these weights 'tuned' during training, during which the models' outputs are mapped to a known 'ground-truth'. The error or 'loss' is measured, and gradient descent is performed in an effort to minimise the difference between model predictions and the desired outputs. This is known as supervised learning.

In this methodology, no training is required to produce Pascal's triangle. Instead, we choose convolution weights as [[1,1],[0,0]] in a 2x2 matrix. We utilise 'padding' of 1 pixel above, and to the left of incoming data which helps to produce the triangular shape in outputs. As shown in the diagram below, using these same weights in sequential 'Convolution layers' produces each row of Pascal. Use of 'Addition layer' then yields the full triangle, to a given number of rows. This may be repeated ad-infinitum for larger representations of Pascal. Pascal's triangle may also be

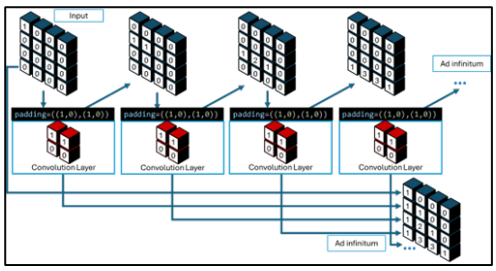

Figure 3: Pascal's triangle derived from convolutions

cumulatively added from left-to-right, to yield Bernoulli's triangle. This contains other mathematical sequences as aforementioned, which may be used to solve a variety of other mathematical problems. To achieve cumulative-sum with a Convolution layer, we can use the identity matrix, and reshape this to a 4D tensor with shape [1,N,1,N] where N is the number of rows in Pascal's triangle. For this to work, we utilise Padding of [[0,0],[N-1],0]. This effectively creates an offset, such that 1s within the weights matrix, are selecting proceeding numbers from a row in Pascal, and since convolution is effectively 'sum-product', we have cumulative sum. We need to use 'lower triangle' (np.tril function) to maintain the triangle shape however this can trivially be represented as NN connection weights also. This approach is visualised in the schematic below. This yields the

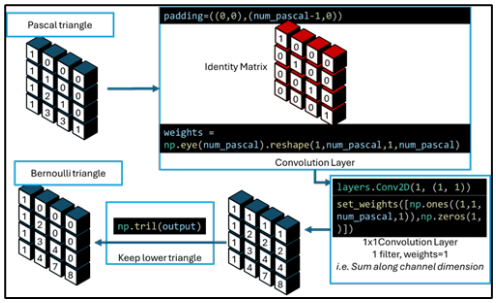

Figure 4: Bernoulli Triangle

Lazy Caterer's Sequence, Cake Numbers, circles divided into areas (A000124, A000125, A000127 from On-Line Encyclopedia of Integer Sequences [12]). In general counting from zero, a given column (k+1) and row n gives the maximum number of regions in k dimensional space formed by (k-1)(n-1) dimensional hyperplanes. Alternatively, it reveals the ordered partitions (compositions) of (n+1) into a maximum of (k+1) parts.

These sequences within Bernoulli are integral to mathematical analysis, and how geometry applies to spatial dimensions. If one considers a Support Vector as a common way to define a boundary in classification problems, then the use of partitioning feature spaces optimally, is of critical importance to Machine Learning. We know from Peano axioms that when defining number theory, each natural number is incremented from a preceding number with a 'success function' i.e. (s+1). Appreciating composition is somewhat more advanced but confers all potential mappings

for a given integer, when constituted from smaller positive integers. It essentially encodes all possible combinations for arriving at a given integer using summation. This paper contends that embedding Pascal, along with Bernoulli's triangle within a Neural Network is encapsulating mathematical understanding which may be leveraged for downstream problem solving, in a modular, reusable fashion.

2.2 Index Methodology

For Pascal's and Bernoulli's triangle to be most useful, they are to be indexable arrays, where specific matrix elements are retrieved corresponding to a question at hand. This might be retrieving a specific Nth term in a Fibonacci sequence, or in the case of Binomial expansions, reading from a specific row and column index, according to the exponents used.

For indexing to work, the cumulative sum convolution weights are used again. This highlights functional reusability, since the weights are identical however the method is used in another context. An integer is to be supplied as the desired index key. This might be the number 4, if we desired the 4th row for instance, should indexing begin at 1 (or similarly the 3rd row, should indexing begin at 0).

At the same time, a negated unit input of -1 is supplied, 'tiled' (repeated for the triangle size), and cumulatively summed. For a Pascal's triangle of 4 rows, this might yield [-1,-2,-3,-4]. An addition layer is used to combine the index key, with the cumulative negative row numbers. Where they match, a value of 0 will result since they perfectly cancel each other. In this example, the result of addition would be [3, 2, 1, 0] since we added [-1,-2,-3,-4] with [4,4,4,4].

In order to convey that a match is found, we require an activation function which activates strongly with zero input. Radial basis function serves this purpose. It can be parameterised such that an input of 0 returns 1, whilst any other integer input returns 0. If we now multiply this result with our Pascal's triangle, it will nullify all results besides the one multiplied by 1 which will be returned unchanged. This can be performed as row, or column indexing. It can be performed against any array (or matrix/Tensor) that we wish to index, against any axis (dimension).

The method outlined requires 'Maxpool' and 'Minpool' layers, to extract nonzero elements from an otherwise sparse array. For example, for a 16-layer Pascal's triangle, when requesting coefficients for a given layer we shall find 15 other empty rows in the output. Maxpool is common in Computer-Vision, typically for downsizing rows and columns to more manageable sizes. Here, we use it simply for extracting relevant results.

Whilst most of this paper uses row indexing to derive Binomial coefficients from Pascal's triangle, we can use this same method to derive various sequences as columns of the left-justified triangle also, as shown pictorially above.

Given that Bernoulli's triangle has been illustrated, we can also retrieve sequences via indexing columns here also. These columns of Bernoulli would be prefixed by the leading diagonal, up to the nth term of a given sequence (where n denotes the column).

2.3 Coefficient Multipliers and Exponents

Pascal's triangle is useful for a myriad of things as noted earlier in this paper. One common application is the solution of Binomial Expansions. In this case, a particular row of Pascal will yield coefficients for an expanded equation. To fully answer such a question, however, each term will require exponents applied, along with the coefficients for multiplication. The following figure aims to demonstrate an indexable array approach, with an example given.

$$(Ax + By)^c = \sum_{k=0}^{c} \binom{c}{k} A^{c-k} B^k \qquad (1)$$

In the equation above, we have utilised A, B and C as terms. Often 'n' is used in place of C, and the function 'nchoosek' returns Binomial Coefficients.

We have shown thus far that Pascal's triangle rows may be indexed to yield coefficients for Binomial expansions. Should an equation contain unit

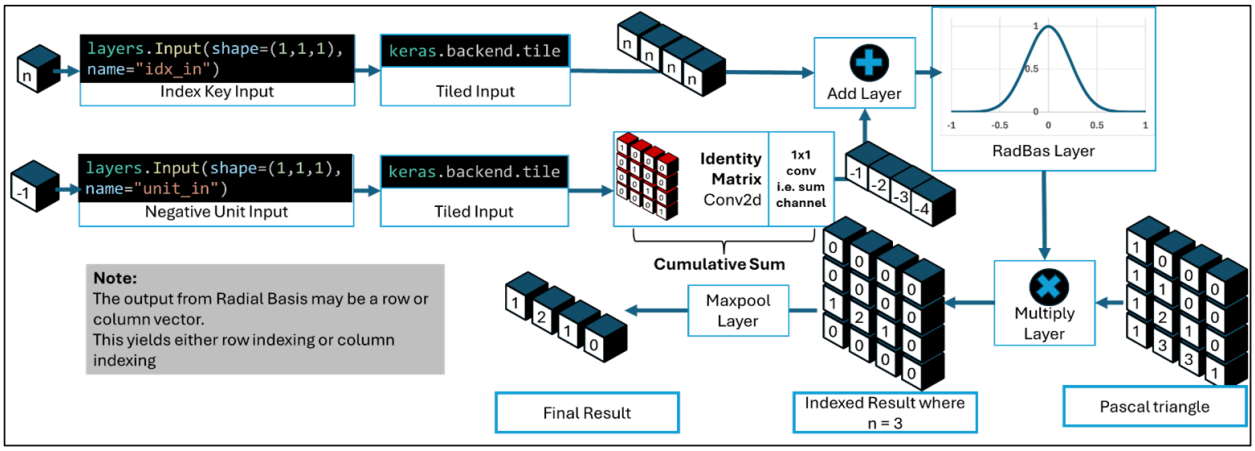

Figure 5: Indexing Operation

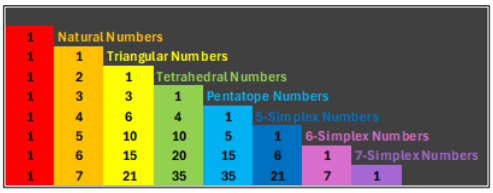

Figure 6: Columns of Pascal's triangle

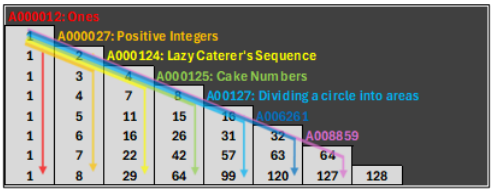

Figure 7: Bernoulli triangle sequences

values for A and B, then this is already sufficient to solve x and y raised to a given exponent. For any model outputs intended in natural language, we would merely have to attribute suitable x and y exponents for these numeric outputs. This may be performed such that for a given power C, we decrement terms of x from C down to 0, and increment y from 0 to C.

Where A and B are terms greater than 1, we also need to exponentiate these terms before multiplying by the provisioned Binomial coefficients. Figure 8 shows 2 arrays which may be input to our model, and indexed just as Pascal's triangle is, to determine the suitable exponent notation for x and y, as well as calculating substituted A and B exponents prior to multiplying with Binomial

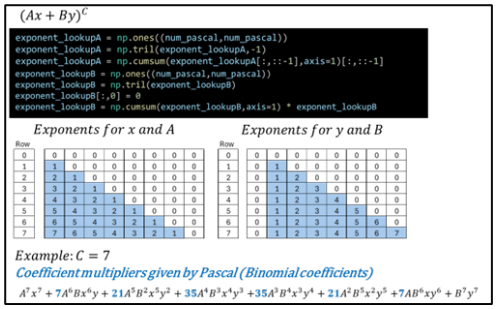

Figure 8: Exponents for expansions

coefficients from Pascal.

To produce these arrays, we can cumulative sum horizontally a NumPy array of ones and make use of 'lower triangle' function. For A and x, we cumulatively sum from right to left with [::-1] and keep the lower triangle excluding the main diagonal (-1 argument in tril function). For B and y, we drop the first column, then cumulatively sum the same axis from left to right (default) and then multiply by the lower triangle, main-diagonal inclusive. The figure shows an example where the exponent is 7 hence Binomial coefficients are 1, 7, 21, 35, 35, 21 and 7. Our supplied lookup arrays allow us to exponentiate any substituted values of A and B and thereafter multiply by these coefficients.

Whilst a neuron method could be used to construct these arrays, and exponents could be calculated by sequential multiplication layers, a custom 'exponentLayer' is utilised for ease of implementation in this paper. The same neuron

indexing for Pascal is utilised on these arrays, with Radial Basis activation function.

2.4 Latex Image Inferencing

To illustrate that the Pascal network and exponent arrays can be used modularly to solve real world problems, a further NN was developed that transforms image inputs into coefficients. This Convolutional Neural Network (CNN) receives images of pixel size [32,128] and outputs 5 classifications. This requires 5 Softmax layers concurrently being trained with their own loss functions. The data used to train these, relies on Pythons

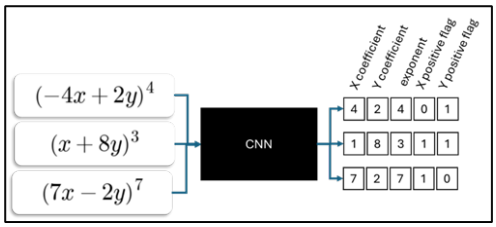

Figure 9: Computer Vision with LaTeX images

symbolic maths library 'SymPy'. An iterative loop builds equations, using x and y coefficients from -8 to +8 excluding zero. Two Boolean 'flags' are used to denote whether x or y coefficients are positive or negative. Exponents are also used in the parameter sweep, from 2 to 8. Whilst for ease of reference the figure above presents results in the range 1-8 for coefficients and 2-8 for exponents, the model outputs 0-7 and 0-6 respectively (indexing at 0).

To improve regularization on a relatively small image dataset of 1792 images, Data-Augmentation is used whereby images may be zoomed randomly up to 20% and translated in x and y co-ordinates by 10%. The model was trained for 15 epochs, with 1434 used for training and 358 for testing. Figure 10 illustrates some typical inputs, along with typical augmentation.

2.5 Testing Pascal's Binomial Expansions

In Figure 5, 'Maxpool' was highlighted as a mechanism to retrieve non-zero values from an otherwise sparse array (post use of Radial Basis for

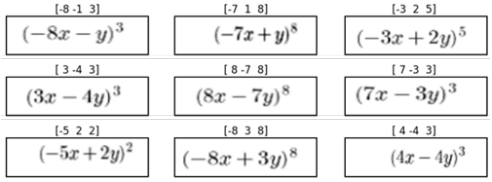

Figure 10: Typical LaTeX inputs

indexing desired subsets). This, however, leads to a limitation, where only positive numbers may be extracted. To remedy this, two parallel DAG branches are executed, with 'tf.reduce-max' and 'tf.reduce-min' to effective max and minimum pool; thereby extracting all non-zero terms. This allows us to test negative coefficients in Binomial Expansions.

To test the Binomial Expansion separately to the image inferencing noted in the prior section, we again make use of SymPy to develop factored equations and their expanded counterparts.

3 Results

3.1 Inferencing Latex Images

Having performed a classic 'Computer Vision' approach with CNN upon images of factored equations, the model performed well on a relatively small dataset. Oftentimes, researchers will split data into three subsets, Training, Validation and Test. The Validation subset is not revealed to the model during parameter tuning, however in the case of 'auto-ML' where hyperparameters are tuned, it is prudent to include a final 'test' where no changes are made since relative merits of hyperparameters are typically assessed against Validation data. In this illustrative example, no attempt to optimise hyperparameters is made, nor any exhaustive attempts to yield the best possible model. Here, Validation serves as the 'Test' dataset.

The model is 100% accurate on Validation data, in identifying the power the brackets are being raised to, the x coefficient, y coefficient and whether x coefficient is positive or negative. It is extremely close to 100% on whether y happens to be positive. This is no surprise because

even where data is split into Training and Validation subsets, it is in reality a problem where all constituents of inputs will occur in other combinations, across different inputs. For instance, whilst (3x+5y) is unique and may appear in a test subset, (2x+5y) contains the same coefficient for y, and (3x+2y) contains the same coefficient for x.

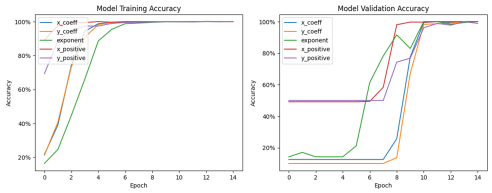

Figure 11: Training Accuracy on Latex Images

Table 1: Inference Results on LaTeX Equation Images

Metric	Exponent	x Coeff.	x Positive	y Coeff.	y Positive
Validation (Test) Accuracy					
Accuracy (%)	100.00	100.00	100.00	100.00	99.09
Validation (Test) Loss					
Loss	0.0149	0.0172	0.0045	0.0231	0.0651

We could test this more meaningfully, by introducing handwritten equations with even greater scope however what we sought to demonstrate, is that a model could simply ingest Latex equations, with coefficients ranging from -8 to 8, and exponents 2 to 8. We included image scaling and translation as Data Augmentation. We are therefore satisfied that this task is easily accomplished, and that model outputs may be handed to the Pascal module, which performs Binomial Expansions.

3.2 Testing Pascal's Binomial Expansions

The Binomial Expansions using the novel Pascal Module outlined in this paper, were flawlessly 100% correct, without any training required. This is due to the well-established connection between Pascal's triangle and Binomial coefficients, and as such this model is fully verifiable and congruent with mathematical laws. What is novel here, is the triangle construct within a neuron paradigm only. The model was tested against 1792 equations, varying x and y coefficients from -8 to 8 (excluding zero), and the exponents from 2 to 8. This is aligned to the Latex images dataset. As before, coefficients of 0 and exponents of 1 are discounted as they would not form a viable Binomial. Whilst Data-Augmentation was used on Latex image data to simulate greater samples, here we have precisely 16x16x7 combinations only.

Table 2: Binomial Expansion Results

Metric	Accuracy	Loss	Time Taken
Test Accuracy	100.00%	0.0	6 ms (0.003 ms/sample)

This took an average of 6 milliseconds across all samples, which is faster than the equations were created and expanded using SymPy library (noting that broadcasting is used in model inferencing, but an iterative loop is used when using SymPy). Speed could be increased by using a larger batch-size (32 here), and by using model quatization however care would need to be taken with float precision. Here, the test data needed to be cast to float32 in order to match the model outputs during accuracy calculation.

4 Conclusion

This paper has established that NN are capable of mathematics, and in a verifiable manner. The main component is an emergent model, with no training mechanism invoked, and uses only integer weights of 1 and 0. The supplied data for building the triangle is the unit input; number 1 itself. One might question the virtue in computing maths, even if current state-of-the-art AI models are shown to struggle; since these functions can be external to a model which otherwise focuses on Natural Language generation. The author of this paper would argue that in the pursuit of scientific research, our method has intrinsic value, but that it may have genuine commercial usage in future, should neuron-based hardware become common. In these cases, requiring external functionality with conventional computing may seem extraneous effort, with interfaces and integrations to be managed. Aside from achieving the main aim, of

Binomial Expansions, along the way, we yielded reusable modular features, such as 'cumulative sum' with convolutional layers, and an indexing mechanism. These will be used again in upcoming publications. The multi-output Computer Vision model used to classify Latex images uses a common design pattern, however 5 concurrent Softmax outputs is rather unusual. This demonstrates that feature extraction performed in the convolution layers is useful to each of the distinct classification tasks, which is another form of functional reuse (albeit learned implicitly whilst the Pascal module has explicit capabilities with design-intent, as noted).

5 Further Work

The author intends to write a series of research papers, focusing on AI Maths, and working towards 'Large Maths Models' (LMM). These will require stateful memory, a symbolic engine and embedded concepts such as fractions and surds. Work is underway for models which generate pseudo-randoms and conduct Monte-Carlo experiments internally. So too, is research concerning neural logic gates, and neural inference engines for 'Logic and Automated Reasoning' where axioms are provided and thereafter may be 'entailed' from a knowledgebase. This paper is one of the first steps on this journey, and it is hoped it will complement other strands of concurrent work by the broader community. In terms of further work relating to this piece alone, the author would like to demonstrate that the 'exponent arrays' can be generated 'neuronally' from first principles. We could have demonstrated calculating exponentials within the model, without invoking a bespoke 'lambda function', but by using repeated 'Multiply layers'. This would have seemed very convoluted and detracted from the main purpose of this paper. We would also like to expand the image dataset with handwritten equations, which would enhance the virtue of this aspect. We consider the main value to be the expansion of Binomials however, which this paper has demonstrated satisfactorily. The code for this work is available in GitHub, and a few additional items are included. Upon first realising the indexing method, it is demonstrated that a model can trivially identify whether an input is a multiple of another number. This was not discussed in depth in this paper; however, it can be shown that a typical NN struggles with this task when trained naively in supervised learning fashion. The code base shows that we can index columns of Pascal's triangle not simply rows, for extracting things like Simplex numbers. We demonstrated also that Pascal's triangle can be cumulatively summed to deduce Bernoulli's triangle which contains many interesting sequences. Various other maths-based use-cases could be contrived here, to extol the virtues of approaches outlined in this paper.

6 Appendix

For ease of reference, the schematics below illustrate layers used in the Pascal Module. These are available in the provided code repository, for closer inspection. The code will be made publicly available upon publication / feedback from conference received.

7 Model Schematics

The figure above shows how the model appears when using keras' plot-model, purely against the layers which formulate Pascal's triangle. As noted in the method section, this constitutes various repeating 'cumulative sum' layers, with 'zero-padding' creating an offset for each triangle row, such that it sums the number 'above' and the one immediately preceding that. Subsequently, each of the triangle rows are aggregated with an 'Add layer'. This same information is summarised in the following diagram, with each NN layer feeding into the next, as well as parallel DAG branches converging to the 'Add layer' aforementioned. Please note that the triangle is parameterised and shown here with 12 rows, however 16 were used in the main experiment. Once integrated with row indexing via 'Radial Basis', and provided exponential lookup arrays, the model's layer structure looks more like Figure 14. For completeness, the following figure shows the lay-

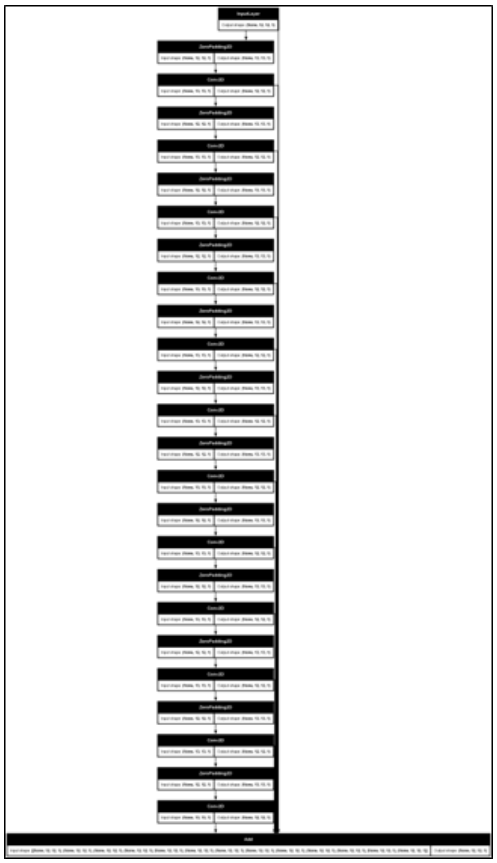

Figure 12: Pascal Layers - Binomial Coefficients

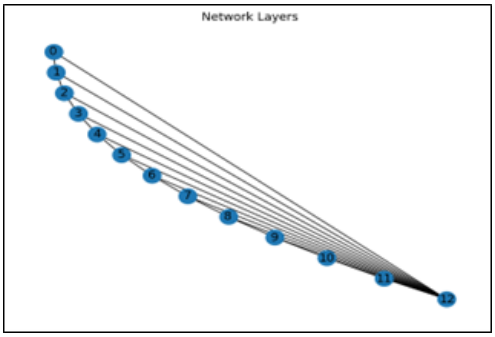

Figure 13: Pascal's triangle DAG

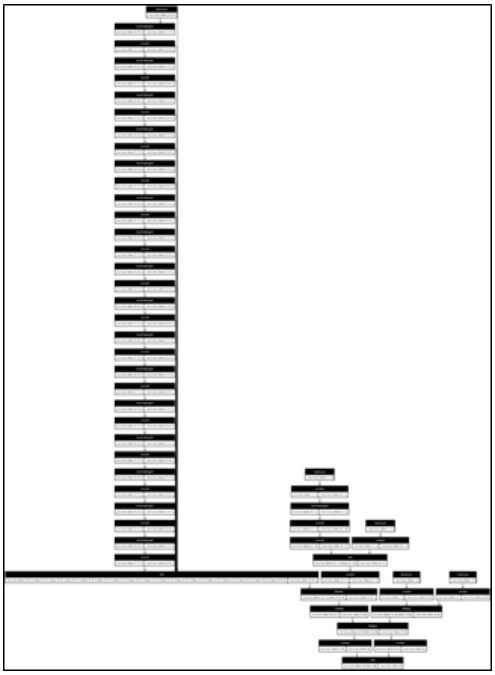

Figure 14: Pascal 'full model'

ers used for the Latex image classification. This is a fairly common Computer Vision design pattern, using blocks of Convolution, Batch-Normalisation and ReLU activation. The use of 5 concurrent Softmax outputs is somewhat uncommon, but suits this problem well, with each component of the equation (coefficient values, whether positive or negative and exponents) forming distinct classification problems. The layers upstream of outputs, develop feature extraction that is useful to each of these concurrent tasks.

8 Acknowledgment

This work was conducted without funded sponsorship, in the author's spare time. Some of the ideas first manifested during a Part-Time MSc in Applied Artificial Intelligence at Cranfield University, which was funded by a previous employer, BAE Systems. These ideas have been greatly developed since, with the inclusion of indexing, exponent arrays and Latex imagery seen in this paper. I would like to thank Cranfield University, and tutors such as Professor Weisi Guo, Professor Saba Al-Rubaye and Dr Ivan Petrunin for their academic guidance and support throughout the MSc process. I would like to thank BAE Systems, and particularly Stewart Webb, Andrew Gordon and Nick Colossimo for their support during my tenure as Lead Data Scientist at BAE Systems.

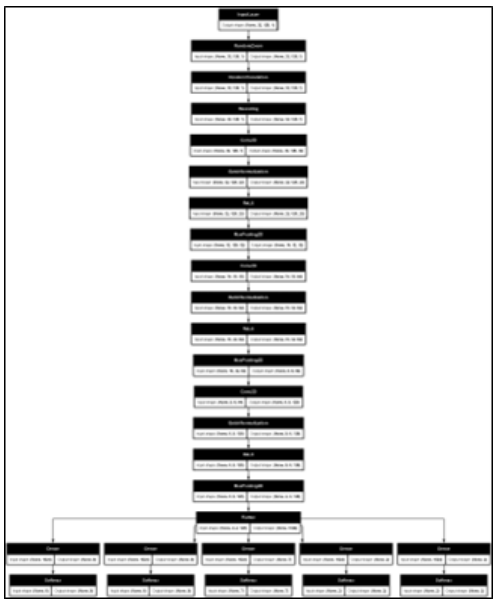

Figure 15: LaTeX image classifier

References

[1] Andrea Breard. Jia xian. `https://www.britannica.com/biography/Jia-Xian`, 2024. Accessed: 2024-09-11.

[2] Charles Q. Choi. 7 revealing ways ais fail. `https://spectrum.ieee.org/ai-failures`, 2021. Accessed: 2025-03-11.

[3] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[4] Hugging Face. Models. `https://huggingface.co/models`. Accessed: 2025-03-11.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, 2012. Also in: Communications of the ACM 60: 84–90, doi: 10.1145/3065386.

[6] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3(2):246–257, 1991.

[7] David Pengelley. Pascal's treatise on the arithmetical triangle: Mathematical induction, combinations, the binomial theorem and fermat's theorem. In B. Hopkins, editor, *Resources for Teaching Discrete Mathematics*, pages 185–196. Mathematical Association of America, 2009.

[8] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. Accessed: 2025-03-11.

[9] David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[10] A. Shrestha, H. Fang, Z. Mei, D. P. Rider, Q. Wu, and Q. Qiu. A survey on neuromorphic computing: Models and hardware. *IEEE Circuits and Systems Magazine*, 22(2):6–35, 2022.

[11] Julien Simon. Large language models: A new moore's law? `https://huggingface.co/blog/large-language-models`, 2021. Accessed: 2025-03-12.

[12] N. J. A. Sloane. The on-line encyclopedia of integer sequences (oeis). `https://oeis.org/`, 2025. Accessed: 2025-03-12.

[13] trwrappers. Interview with inventor of neural nets warren mcculloch. YouTube, 2024. Accessed: 2025-03-11.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[15] Z. Yu, A. M. Abdulghani, A. Zahid, H. Heidari, M. A. Imran, and Q. H. Abbasi. An overview of neuromorphic computing for artificial intelligence enabled hardware-based hopfield neural network. *IEEE Access*, 8:67085–67099, 2020.