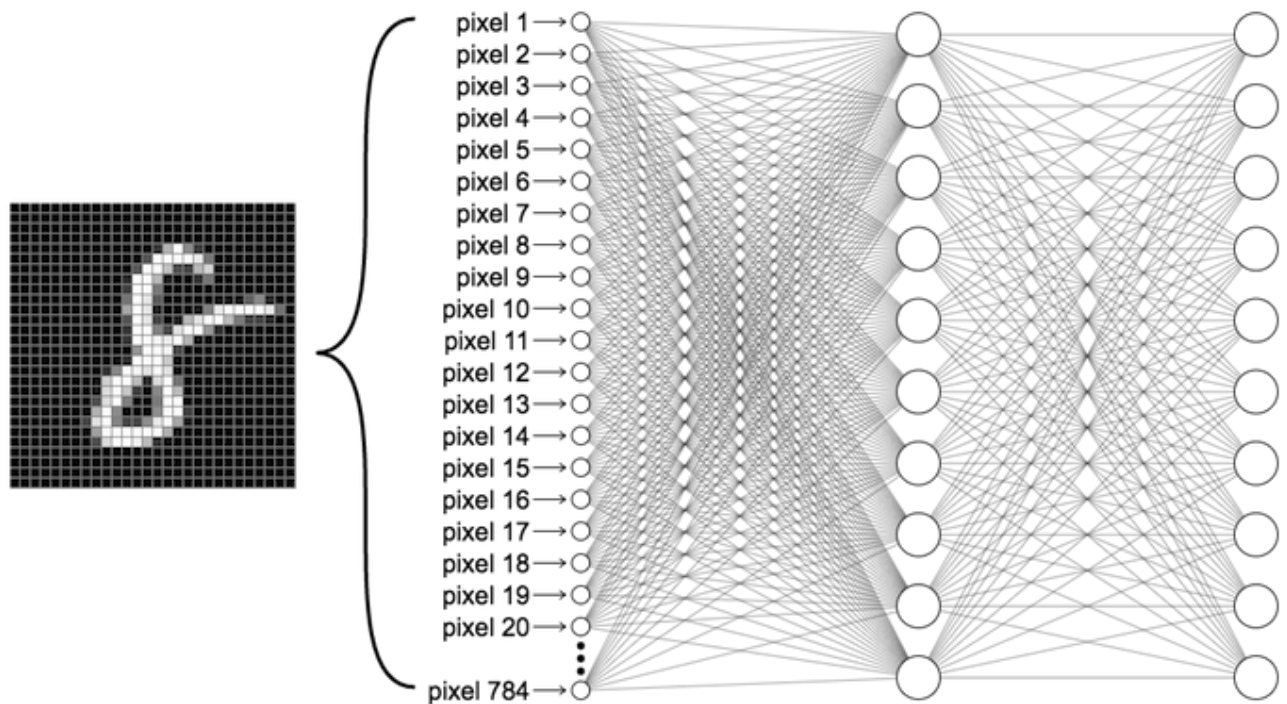


Intro. to Machine Learning - Portfolio Pt.III

Neural Networks



(Image source: Lavanya, 2019)

Programming, research, and report content produced by
Cameron Scott – csc0012
Monash College (Monash University).

Table of Contents

1.0	Abstract.....	page 3
2.0	Introduction.....	page 3
3.0	Methodology.....	page 4
3.1	Data & Software Review	
3.2	Data Processing.....	page 5
3.2.1	Generating and Preprocessing	
3.2.2	Formatting and Post-Processing	
3.2.3	Re-Processing Data	
3.3	Neural Network Design.....	page 6
3.3.1	Principal Design Considerations	
3.3.2	Model Development.....	page 7
3.3.3	Network Redesign	
3.3.4	Final Structure; Size & Complexity...	page 8
3.4	Functional Configuration.....	page 9
3.4.1	Activation Functions	
3.4.2	Loss Functions	
3.4.3	Dropout	
3.5	Hyperparameter Calibration.....	page 10
3.5.1	Learning & Bias Updater Rates	
3.5.2	Epoch vs K-fold vs LR	
3.5.3	Optimisation Algorithm: Gradient Descent & Seeding	
3.6	Model Evaluation.....	page 11
3.6.1	Train-Test Options	
3.6.2	Percentile Splits	
3.6.3	K-Fold Cross Validation.....	page 12
3.6.4	Dataset Aggregation: Train-Test Set Separation	
3.6.5	Model Summary, per Train- Test Mode.....	page 13
3.7	Improving Performance.....	page 14
3.7.1	Prediction (↑ data) vs Integrity (K-fold or Separate Test Data)	
3.7.2	Generalisation & Scale	
3.7.3	Computational: Power, Bitrate & Bandwidth.....	page 15
3.7.4	Increasing NN Scale for Larger Problems	
4.0	Conclusion	
5.0	Appendix	
5.1	Image Dataset.....	page 17
5.2	Image Data Guide.....	page 18
5.3	References.....	page 19

1.0 Abstract

This paper seeks to explore Artificial Neural Network (ANN) techniques of the deep learning subfield of machine learning. The problem being addressed is to devise an image recognition model which accurately predicts the category of shapes inputted to the neural network.

Information filtering and neurological activation through linear scanning is the cognitive processes behind biological neural networks that underpin general pattern recognition, and which inspire the field of deep learning AI.

Extensive deep learning applications of filtering and processing of imagery are currently experiencing rapid development in both industry and research. The challenge of this report is to focus more narrowly on the machine learning applications of Intelligent Character Recognition (ICR) for encoding handwritten text into digitally processed classifications.

2.0 Introduction

Proceeding from this challenge, a sequence of data science research processes is employed which ultimately bind together learning outcomes that form the learning pathway of Monash College/University's 'Introduction to Machine Learning' industry intensive program (2022).

First, data collation and assembly are performed in the form of generating individual 'hand drawn' images using basic computer graphic image editing software. A balanced set of four simple shapes symbols – circle, square, triangle and cross – are created and the set is multiplied, maintaining an equally balanced distribution of the four shape categories.

The collection of image data is inspected and cleaned for standardisation and to enhance model integrity, then the data is used to train a simple neural network image classification model.

Having executed an ANN abridged software-specific profile of the dataset and analysed the default model classification performance, the neural network structure is designed and calibrated in accordance with the particularities of the model's initial fitting of its input imagery.

The above processes are iteratively repeated with necessary cleaning and transformation of the input dataset until the model reaches a basic structure that consistently produces acceptable baseline performance. A robust neural network having been constructed, it is then tested across expanded datasets of differing versions of the same four shapes.

With hundreds of test executions performed, the model's hyperparameters are then calibrated - the focus of design shifting from consistently and tightly fitting its testing data; toward **optimisation**, such that the model may reliably **generalise** its test performance across 'unseen' data for which it was not calibrated.

Techniques of neural network design, training and testing procedures are discussed in the *Methodology* section below. Comparisons of inferential power between techniques receives considerable focus in the paper's closing, due to the **inordinate effect of model Train-Test choices**, and its implications for machine learning **data integrity**.

Leading to the paper's conclusion, it is observed that while satisfying performance results could be achieved – **some exceeding 90% classification success** – generalisable inferential value of such results should be interpreted with scepticism appropriate to any science.

In the broader context of developing a more generalised artificial intelligence system, the greatest insights from this challenge centre around **true generalisation potential**. In particular, widely differentiated performance capabilities of neural network models when tested across data splitting techniques, as contrasted with more intransigent model testing upon designated unseen datasets.

3.0 Methodology

The Methodology section of this paper covers the traditional *Report Body*. Beginning with (3.1) a **review of the project data**, moving through (3.2) **data processing**, re-processing, and formatting techniques.

Next, (3.3) is a discussion of the evolutionary development of the Neural Network's basic **Structure, Shape and Size**, following which is a thorough decomposition of the Network's refinement: **Functional configuration** and **Hyperparameter calibration**.

In Section 3.6, this Methodology encompasses an iterative **Evaluation of the Model**; preceded with a justification of this report structure, as affirmed by conventional practice in current-day Machine Learning research (Brownlee, 2020).

This Model Evaluation section focuses on the selection of **Test Options** and unravels their implications upon model **performance & integrity**, before explicating **Performance Improvements** and **Scaling Solutions**.

3.1 Data & Software Review

A set of symbols were generated for the training and test phases of the Neural Network input data.

The symbols represent each of four simple Shapes: Circle, Square, Triangle and Cross. These four shapes are repeated in a scaled set, initially recommended to comprise a minimum of 10 variations of each shape (Monash College, 2022).

However, the initial dataset for this project was a total of 80 images, divided equally among the four shapes. Each shape is 'hand-drawn' in basic graphic image editing software and saved in bitmap format (".bmp"), of 10x10 pixel resolution, for consistent file size of 102 bytes.

During model development this dataset is expanded to 30 of each shape, totalling 120 images.

Various dataset combinations were utilised through model evolution, including: randomised divisions of Train-Test data ("**% Splits**"); systematic combinations ("**K-Fold Cross Validation**"); as well as distinctly **separate sets of Training and Test data** ("selected Test Set").

This '*separate Test Dataset*' (80 images) was solely imported from a peer, Richard Maher (2022).

Ultimately, the expanded dataset utilised is a concatenation of Richard's and my own image files. The complete image dataset can be found in Appendix 5.1.

These data are defined by category according to their corresponding Shape label:

- 0 - Circle
- 1 - Square
- 2 - Triangle
- 3 - Cross

Appendix 4.2 contains the guide (Monash College, 2022) utilised for creating of the image data files.

The *Waikato Environment for Knowledge Analysis* ("Weka") 'Practical Machine Learning' software application is used to explore the challenges posed of this report, and to create the Neural Network.

Weka's '*DL4jMlpClassifier*' algorithm is deployed here for image recognition and classification.

3.2 Data Processing

3.2.1 Generating and Preprocessing

Pre-processing of the data follows form discussed in the previous subsection. The dataset has expanded with development of the Neural Network described as follows.

The original dataset of Shape imagery grew from a length of 80 images, to 120 for much of model training. A secondary and **separate data source** was introduced for isolated generalisation testing.

Combined Train and Test sets totalled 200 image files; later experimentation conducted across the two datasets in aggregate.

3.2.2 Formatting and Post-Processing

Post-processing of the data for dimensionality reduction via matrix pooling algorithms was trialled within Matlab integrated development suite. Transformation of higher resolution images down to the 10x10 recommendation were abandoned in light of new information suggesting this approach may be outside the scope of the project.

A preliminary **performance improvement to the Neural Network** could be achieved in pursuing with this approach: initial experimental considerations being that a higher resolution image transformed into lower resolution may carry greyscale pixelation (via 'average pooling' algorithm), which may allow for 'rounded edged' symbols and therefore allow Hidden Layer networks to train on these more subtle higher-level image characteristics, producing a more robust model.

3.2.3 Re-Processing Data

Initial image data were modified in response to very poor model performance. First phase model executions resulted in severely differentiated performance metrics per shape category; consistently resulting in failure to categorise Triangle shapes better than 50% probability ($P\{\Delta\} < 0.5$).

Such poor performance results firstly motivated many of the images to be edited or redrawn entirely, such as the triangles. The dataset was initially expanded when a performance limitation appeared to have been reached – acknowledging that more data upon such a low base most often results in a better trained model.

3.3 Neural Network Design

Design considerations are primarily based upon the methodology to '**overbuild**' the network: loading excessive layers and neurons relative to estimated prediction complexity requirements; and then '**scaling down**' to a network complexity until significant performance losses are observed (Lavanya 2019).

Consistent with expected outcomes of the 'overbuild' technique, noticeable **overfitting** of the model to the data is observed: pleasing prediction metrics achieved, while hyperparameter calibration had limited effect compared to adjustments to network structure.

3.3.1 Principal Design Considerations

Foremost considerations for the neural network framework centre on the imperative structure for the network. This is dictated by fundamental **Input & Output neuron features**, followed by **Hidden Layer depth** expansion trade-offs:

Input

- Base problem requirements specify a 100-pixel (10x10) image as read input data.
- Neural Network character/image recognition design of this kind necessitates a single neuron per datapoint (ie. per pixel) on the input side. This specification therefore requires **100 input neurons**.
- As inputs to the model, these neurons' state is defined by each variable's present condition through each train/test iteration.

Output

- Four shape categories for classification require a complete set of four output neurons, for which
- The **Softmax activation function** will assign a probability to each, totalling 100% exhaustible set of outcomes: $P\{\mathcal{E}\} = 1.0$

Hidden Layer

The number of hidden layers selected is far less definitive, more of 'an art than a science', and is here determined by myriad 'best practice' heuristics (Hijazi, Kumar, Rowen, 2015; Karpathy, 2015).

For an initial 'rule of thumb' the guiding formula was utilised:

Number of hidden layer neurons: $(\#Inputs + \#Outputs)^{1/2} + \#[1:10]$ (Monash College, 2022)

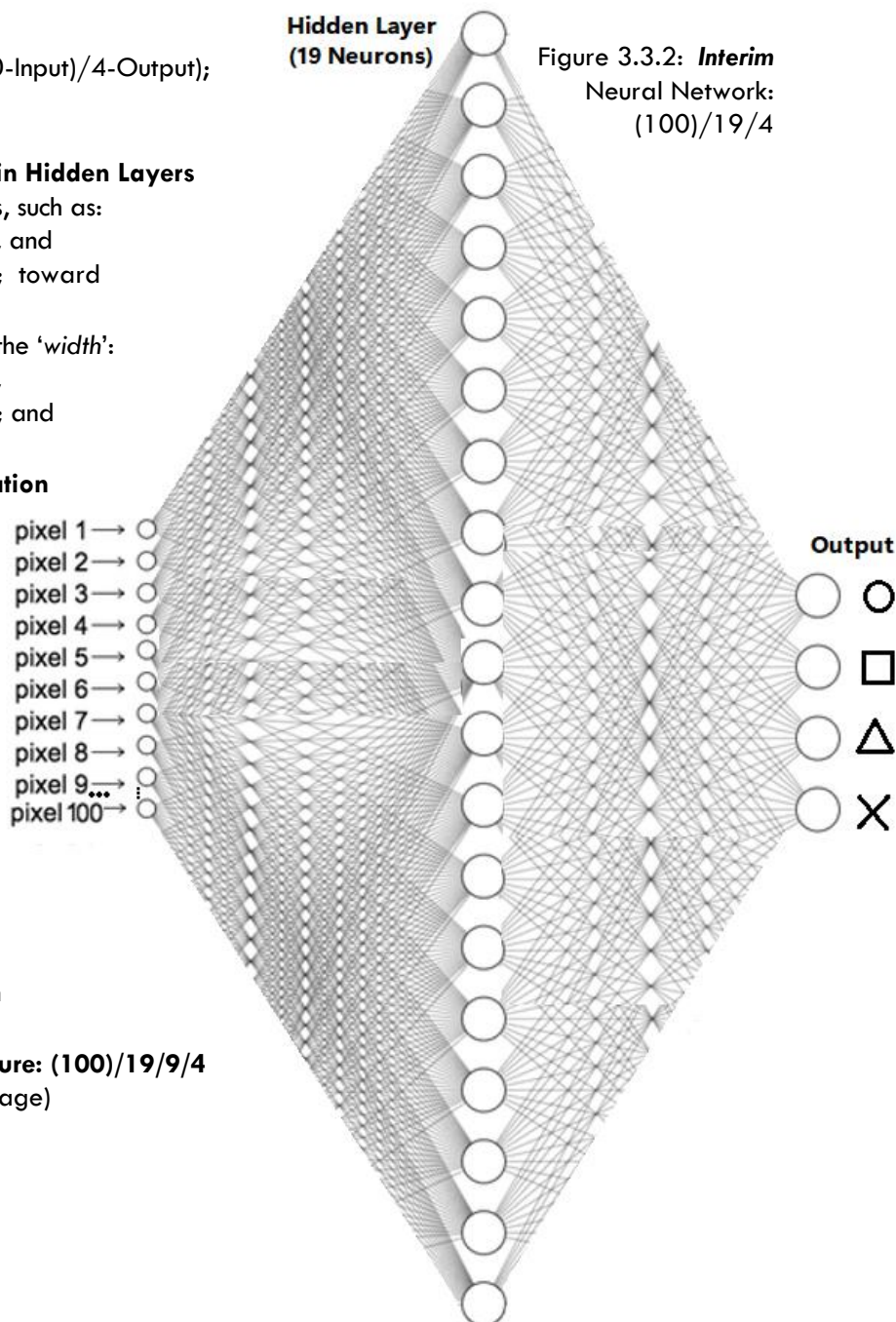
3.3.2 Model Development

The above formula is intentionally amenable to structure modifications. As such, and consistent with 'Size and Complexity' (3.3.4) considerations describe below, initial versions in model development evolved from:

- Zero hidden layers ((100-Input)/4-Output); into a
- Heavy structure - '**deep**' in **Hidden Layers** and '**wide**' with **neurons**, such as:
(100)/22/22/22/22/4, and
(100)/48/56/40/52/4; toward
- Systematically reducing the '**width**':
(100)/22/10/22/22/4,
(100)/14/16/14/12/4; and
- Simplifying via **Optimisation** processes discussed later (Section ##, ##, ##):
(100)/22/10/4,
(100)/19/4
(depicted here)
- Re-expanded in concert with the **dataset's later expansion and concatenation**:
(100)/19/19/4,
(100)/19/10/4,
(100)/13/13/4
(100)/9/10/4,
(100)/9/9/4... to reach
- **The final network structure: (100)/19/9/4**
(depicted on following page)

**Hidden Layer
(19 Neurons)**

Figure 3.3.2: *Interim*
Neural Network:
(100)/19/4



3.3.3 Network Redesign

The final two stages in Model Development above (3.3.2) represent the network's redesign to accommodate the expanded and concatenated datasets.

This design revision is informed by widely cited heuristics (Brownlee, 2020), characterised by an iteratively decreasing number of neurons per layer, from Input to Output: (100)/19/9/4

Being a "frontloaded" network, higher-level image features are identified and processed 'early' in the upper Hidden Layers; whilst backpropagating error minimisation through later layers poses lesser risk to eradicating these subtle features (mostly imperceptible to the human eye) if these neurons were instead loaded into the deeper network layers (Suryansh, 2018).

3.3.4 Final Structure; Size & Complexity

Furthering the Network Redesign considerations specified in Section 3.3.3, the final Neural Network architecture is designed with foremost consideration of its size and complexity:

Large enough to handle disguised attributes detected between 10x10 pixel relations, but not so large as to stifle backpropagation gradation and engender overfitting.

A depth of 3 to 6 layers (including input and output layers) is found herein to produce best results.

Consistent with Andrej Karpathy's 'recipe' (2019), the approach adopted here was to focus on Training Loss by designing a performing NN large enough to overfit, and then target Validation Loss at the expense of Training Loss through regularisation techniques.

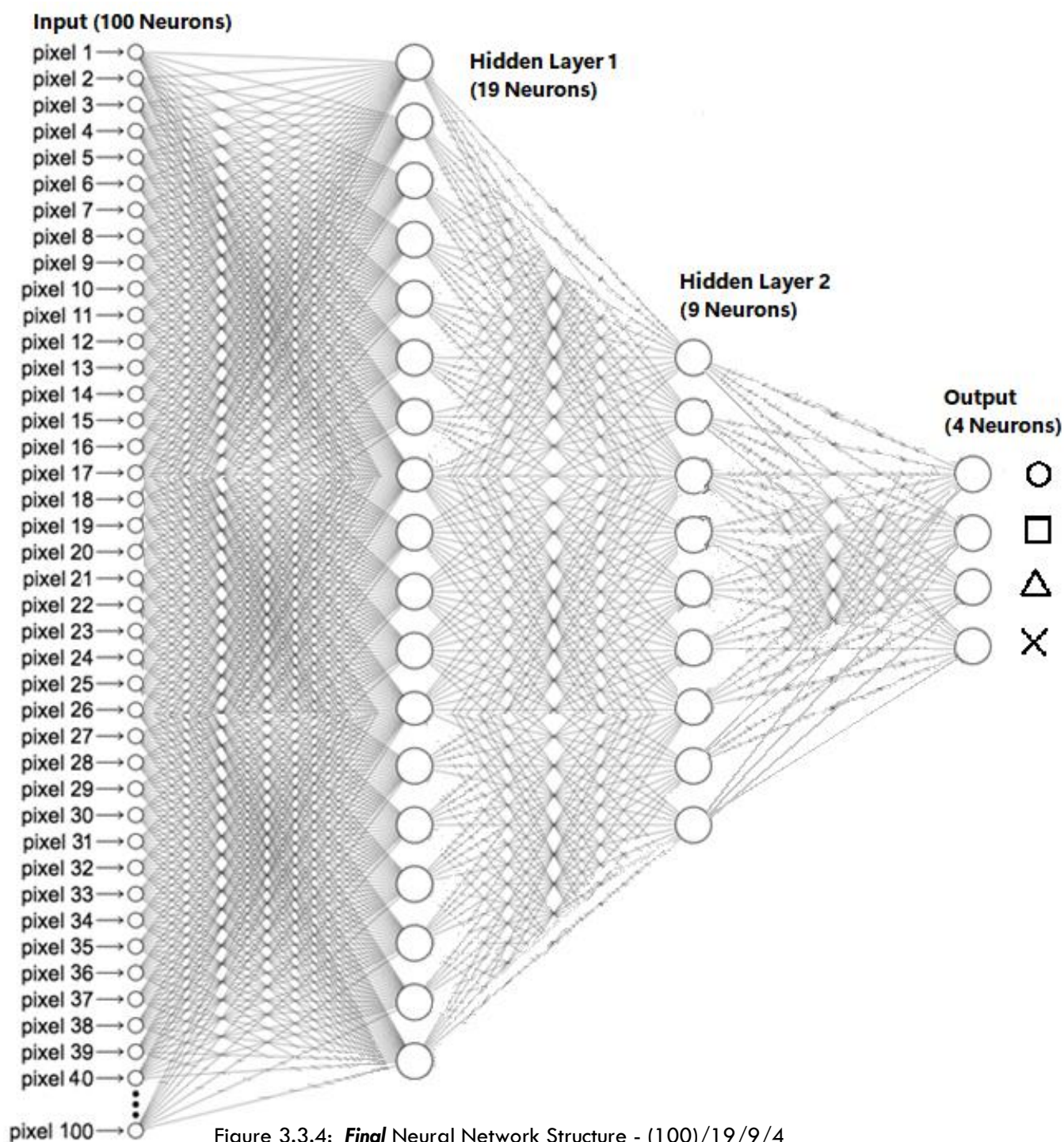


Figure 3.3.4: **Final** Neural Network Structure - (100)/19/9/4

3.4 Functional Configuration

The following design considerations were fundamental in model development - through all its pitfalls, its overinflated and misleading performance results.

3.4.1 Activation Functions

Given this is a Classification problem in which the outcome must be classified as one of the four specified shapes, we therefore use 4 Output Neurons, each utilising the Softmax activation function. This function appropriately sums the total output layer final probabilities to 100% (Lavanya, 2019).

Internal/hidden layer activation functions were generally constrained to ELU, ReLU, Leaky ReLU and Swish, after thorough experimentation.

Limited performance improvement is achieved through this time-intensive calibration of activation functions. This is generally found to be due to the evolving model's overfitting and tight backpropagation.

Here the '**vanishing gradient problem**' becomes pertinent: Excessively deep models can be moderately improved through employing the variations of ReLU that accommodate negative values on their activations (L-ReLU, Swish, etc).

Once the models were optimised, however, backpropagation through deeper layers is less an issue (given the models' reduced depth). Therefore ReLU is found to be more reliable when calibrating hyperparameters in less complex optimised iterations, wherein managing overfitting is more consequential than backpropagation depth.

3.4.2 Loss Functions

Various Loss Functions (LF) were trialled through model development. These are listed below. The **Cross Entropy LF performs most consistently** across extensive changes to the NN structure, core functionality and hyperparameter configuration.

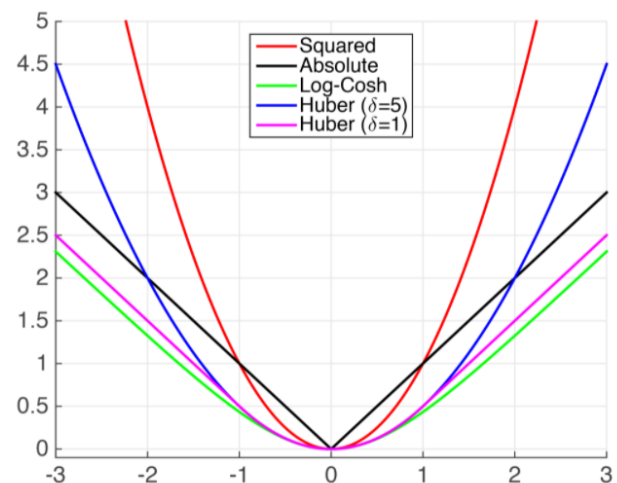
A summary of the performance of LFs that were trialled is listed as follows:

Cross Entropy ('XENT') – This is the most widely used function and found to deliver better performance.

K-L Divergence (KLD) – Similar to XENT, but found to offer worse results in larger datasets.

Loss MultiLabel – too slow.

LossNegLikelihood – low error, quick, but inadequate returns relative to number of epochs.



(Image source: Lavanya, 2019)

3.4.3 Dropout

The model's developmental build from deliberate overbuild to optimisation relied on utilising the Dropout function. This is a regularisation technique that 'switches off' random neurons in each layer during training, to determine which neurons are redundant and therefore contributing to overfitting (Lavanya, 2019).

The algorithm seems to have most impact when reducing from the original build of 6 layers and reached diminish returns to scale once the model had reduced to 2 or 3 hidden layers.

3.5 Hyperparameter Calibration

Significant adjustments to hyperparameter calibrations are retained in favour of major structural and functional changes to the Neural Network.

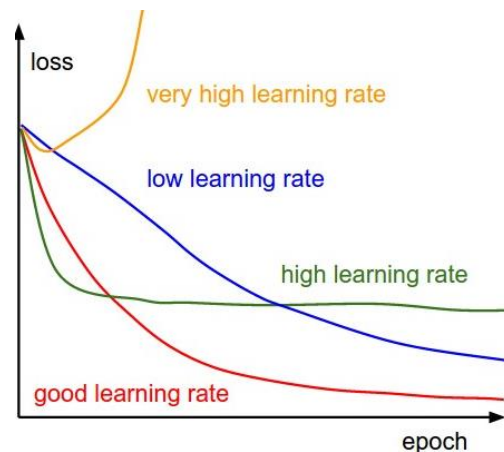
Often modifications in hyperparameters either have insubstantial effects to relative performance improvement, otherwise appearing to convey significant mutual dependencies upon one another.

The hyperparameters have thus been grouped where most applicable.

3.5.1 Learning & Bias Updater Rates

Iterations of the Learning Rate were processed in tandem with adjustments to the number of Epochs, starting at very low ranges from 1E-5 and stepping up gradually to 1E-2.

Each evolution of the model required a Learning Rate contingent upon many other hyperparameters. With a high number of Epoch's (>300) meanwhile employing the 'Early Stopping' algorithm, the Learning Rate generally reached optimality approximating 5E-3.



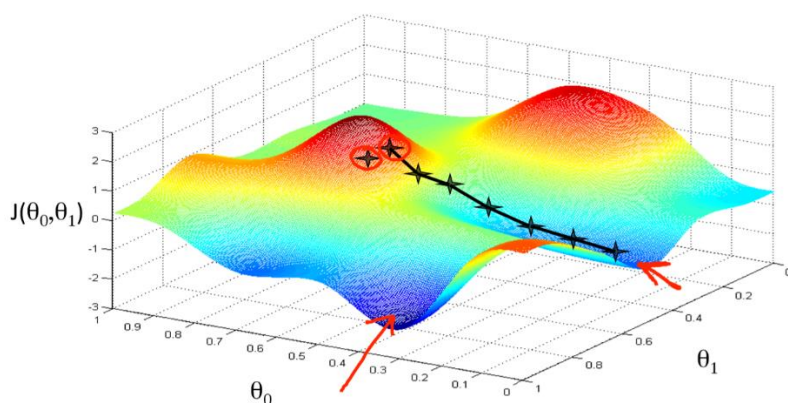
(Image source: Lavanya, 2019)

3.5.2 Epoch vs K-fold vs LR

Large variation in number of epochs carry perhaps the most substantial computational costs of all hyperparameter adjustments. Again these are not an isolated feature; Epoch numbers and processing constraints are interdependent on both Learning Rates, and pose most time intensive consequences when compounded by high 'K' fold numbers in Cross Validation techniques.

Evolutions of this model, especially when overbuilt with deep layers, appear well served when a high number of Epochs and K-folds allow for extended run time, pushing the model deep into gradient descent – suggesting retained capacity for learning improvements.

3.5.3 Optimisaiton Algorithm: Gradient Descent & Seeding



(Image source: Suryansh 2018)

Gradient Descent algorithms invoke trade-offs for time vs performance.

Here it is found that Line & Conjugate Gradient Descent slower but more robust; Stochastic Gradient Descent fastest, with marginally poorer performance.

The two-speed determinant was key to utilising Stochastic for high frequency modelling, whereas refining testing in model development revisited LGD.

Random seed value set varying from 1 appeared only degradational to the model's overall performance.

3.6 Model Evaluation

According to Jason Brownlee (2020) from the Machine Learning Mastery blog, evaluating a model goes beyond what is ostensible in this heading. “It includes testing different **data preparation** schemes, different **learning algorithms**, and different **hyperparameters for well-performing learning algorithms**.”

$$\text{Model} = \text{Data Preparation} + \text{Learning Algorithm} + \text{Hyperparameters}$$

By this definition, Model Evaluation for machine learning seems more appropriately housed within the Methodology section of science or engineering papers, than it does a standalone section.

Intuitively this makes sense, considering the iterative, and ‘reverse engineering’ approach of the machine learning paradigm: the objective is building a model given inputs and results; an inversion of traditional mathematics based scientific approaches.

In keeping with this structure, evaluation is integrated into each model development stage and testing category. Within which, two key questions will be addressed:

1. How could the performance of your NN be improved?
2. How would this approach be scaled up to handle larger problems?

3.6.1 Train-Test Options

K-Fold cross validation and Train-Test split are the preferred techniques of data scientists due to their general effectiveness. However, when selected without due consideration, will often generate misleading and occasionally misleading results when tested on imbalanced classification datasets (Ma & He, 2013).

3.6.2 Percentile Splits

Percentile Splits are effective if when the dataset is very large and representative of the problem. Notwithstanding, these conditions are certainly not applicable to the challenge at hand – especially not in the model’s early evolutions of <100 data instances.

Preliminary model testing across the initial very small dataset of 80 image instances demonstrated stark weakness of the Percentile Split procedure. Namely that it returned 90% and 100% classification results that hold no legitimate implication upon the model integrity.

The adjacent image, exported from Weka Explorer, is presented to demonstrate this performance misrepresentation. This is a **large complex and probably overfit model**.

The original test (*not captured at the time*), was similarly trained upon 90% of the 80-image dataset and **produced a 100% Instance Classification result**.

However, this equates to **only 8 images tested**, wherein **zero Triangles were classified**.

This demonstrates that while **maximal (%) Classification milestones can be achieved**, meaningful inferential value under these dataset size-split conditions is **extremely limited**.

```

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correctly Classified Instances    7  87.5 %
Incorrectly Classified Instances  1  12.5 %
Kappa statistic                  0.8095
Mean absolute error              0.1038
Root mean squared error          0.2156
Relative absolute error          27.2124 %
Root relative squared error      48.8893 %
Total Number of Instances       8

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC  ROC Area  PRC Area  Class
      1.000    0.000    1.000    1.000    1.000    1.000  1.000    1.000    0
      0.750    0.000    1.000    0.750    0.857    0.775  1.000    1.000    1
      ?        0.000    ?        ?        ?        ?        ?        ?        2
      1.000    0.167    0.667    1.000    0.800    0.745  1.000    1.000    3
Weighted Avg.   0.875    0.042    0.917    0.875    0.879    0.824  1.000    1.000

=== Confusion Matrix ===

 a b c d  <-- classified as
 2 0 0 0 | a = 0
 0 3 0 1 | b = 1
 0 0 0 0 | c = 2
 0 0 0 2 | d = 3

```

(The depicted exported results were modelled upon a differing 80-item dataset for demonstration: ‘symbol_data_RICHARD.arff’ (attached). The original dataset was subsequently overwritten.)

These test results flag that **high-Percentile T-T Splits** frequently provide more **favourable estimates** of model performance upon small datasets compared to K-folding.

Informed by the issues described, it is apparent that singular Percentile Splits should **strictly be avoided** for **splitting small sets of data**, unless utilised in tandem with ameliorating techniques such as **'Stratified Sampling'**.

For purposes of communication and verification of model evolution, the Weke exported Training-run summary of the overfit '(100)/22/10/22/22/4' demonstration model is depicted here too.

```

Classifier output
=== Run information ===
Scheme: weka.classifiers.functions.D43MlpClassifier -S 1 -cache-mode MEMORY -early-stopping "weka.d43.earlystopping.Ea
Relation: Symbols
Instances: 80
Attributes: 2
  filename
  class
Test mode: split 90.0% train, remainder test

=== Classifier model (full training set) ===

Network Configuration:
NeuralNetConfiguration(weightInit=XAVIER, biasInit=0.0, dist=weka.d43.distribution.Disabled@66, 11=NaN, 12=NaN, dropout=Disab
Model Summary:
=====
VertexName (VertexType)      min,nOut  TotalParams  ParamsShape  Vertex Inputs
=====
input (InputVertex)          --,--      -            -            [input]
Dense layer 2 2 1 1 2 2 2 (DenseLayer)  100,22     2,222        W:[100,22], b:[1,22] [Dense layer 2 2 1 1 2 2 2]
Dense layer 2 2 1 1 2 (DenseLayer)      22,10      230          W:[22,10], b:[1,10] [Dense layer 2 2 1 1 2 2]
Dense layer 2 2 1 2 (DenseLayer)        10,22      242          W:[10,22], b:[1,22] [Dense layer 2 2 1 2]
Dense layer 2 2 2 (DenseLayer)          22,22      506          W:[22,22], b:[1,22] [Dense layer 2 2 2]
Output layer (OutputLayer)             22,4        92          W:[22,4], b:[1,4]   [Dense layer 2 2 2]

Total Parameters: 3,292
Trainable Parameters: 3,292
Frozen Parameters: 0

=====
Time taken to build model: 0.23 seconds

```

3.6.3 K-Fold Cross Validation

As the model evolves, these data length constraints reveal K-Fold Cross Validation conveys the strongest representational integrity of the model to the data.

Testing of iterative models is executed across a variety of fold numbers ('K' value) – colloquially understood as the number of times the deck of playing cards is 'cut'.

While a 10-fold cross validation is most often adopted (Brownlee 2020), this model tends to **optimise** at 19 folds. This is understood to be hitting the optimal 'sweet spot' of this particular model's configuration, due to the increased (K) folds accommodating more extensive backpropagation epoch iterations.

The K-fold Cross Validation technique proved **more rigorous than singular Percentage splitting**. However, this gain is offset by significant losses in computational processing power and time requirements.

By taking the average estimation performance across the K-fold algorithm's many iterations, it also generates a tighter fitting model, yet accentuates the risk of overfitting.

3.6.4 Dataset Aggregation; Train-Test Set Separation

Observable in the adjacent Weka Explorer output is the updated dataset size, concurrent with model evolution. Here seen applied to a (100)/13/13/4 neural network.

Throughout development, the dataset is experimentally scaled up to more 'useable' lengths – first from 80 to 120 images, and subsequently combined with a separate set of 80 items.

These total 200 images were trained and tested under differing models and hyperparameter configurations, of differing variations of T-T splitting techniques.

The aggregated dataset was finally divided (80/20) into two separate bin allocations for Training (160 items) and Testing (40). This is depicted next as the final version of model development.

Albeit 4 times the size of the Project's originally recommended total dataset size, this final expanded and aggregated dataset still remains far smaller than preferable, and demands that resampling strategies be utilised for deriving any significant data science insights.

```

Classifier output
=== Run information ===
Scheme: weka.classifiers.functions.D43MlpClassifier -S 1 -cache-mode MEMORY -early-stopping "weka.d43.earlystopping.EarlySt
Relation: Symbols
Instances: 200
Attributes: 2
  filename
  class
Test mode: split 80.0% train, remainder test

=== Classifier model (full training set) ===

Network Configuration:
NeuralNetConfiguration(weightInit=XAVIER, biasInit=0.0, dist=weka.d43.distribution.Disabled@64, 11=NaN, 12=NaN, dropout=Dropout(),
Model Summary:
=====
VertexName (VertexType)      min,nOut  TotalParams  ParamsShape  Vertex Inputs
=====
input (InputVertex)          --,--      -            -            [input]
Dense layer 2 2 2 1 2 2 1 (DenseLayer)  100,13     1,313        W:[100,13], b:[1,13] [input]
Dense layer 2 2 2 1 2 2 2 (DenseLayer)   13,13      182          W:[13,13], b:[1,13] [Dense layer 2 2 2 1 2 2 1]
Output layer (OutputLayer)       13,4        56          W:[13,4], b:[1,4]   [Dense layer 2 2 2 1 2 2 2]

Total Parameters: 1,551
Trainable Parameters: 1,551
Frozen Parameters: 0

=====
Time taken to build model: 1.69 seconds

```

```

=== Evaluation on test split ===
Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances      31          77.5 %
Incorrectly Classified Instances     9          22.5 %
Kappa statistic                     0.7052
Mean absolute error                  0.159
Root mean squared error              0.2597
Relative absolute error              42.165 %
Root relative squared error          66.5035 %
Total Number of Instances           40

=== Detailed Accuracy By Class ===

```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0.692	0.000	1.000	0.692	0.818	0.777	0.932	0.906	0	
1.000	0.235	0.429	1.000	0.600	0.872	0.956	0.877	1	
0.636	0.034	0.875	0.636	0.737	0.672	0.868	0.813	2	
0.900	0.000	1.000	0.900	0.947	0.933	0.983	0.959	3	
Weighted Avg.	0.775	0.045	0.880	0.775	0.795	0.756	0.931	0.889	

```

=== Confusion Matrix ===
 a b c d <-- classified as
9 3 1 0 | a = 0
0 6 0 0 | b = 1
0 4 7 0 | c = 2
0 1 0 9 | d = 3

```

3.6.5 Model Summary, per Train- Test Mode

Consistent with expectations, at this stage the model does not sustain the high estimation success of either K-Fold Cross Validation technique, nor the split Percentage Train-Testing results.

This observation follows on from 'overfitting' discussions in subsection 3.6.3:

- The size and complexity of the neural network had initially and intentionally been 'overbuilt'.
- Then, in 3.6.1, it is severely narrow-fitted onto an invalidly small Test % Split, of 8 instances.
- Subsequently, the dataset is expanded; the model re-tested through K-fold techniques.
- The more rigorous methodology of K-fold Cross Validation allows for the model's structure and functionality to progressively evolve.
- As the **network is optimised**, paring back its layer depth and neuron width, the model becomes increasingly well fit - **albeit remaining excessively large and complex**.
- This latter-stage overbuild is masked by the nature of K-fold algorithm: Trained and Tested extensively across a specified number of folds for each execution, test results reflect a model that is particularly well optimised and error-minimised over the best performing folds **precise to the training data**.
- Thus, the distinct Training and Test datasets are created to **configure and calibrate a more generalisable model**, which can be tested upon 'unseen' data (unrecognised imagery).

This summary elucidates the need for separate Training and Test datasets. The below Weka Explorer classifier output represents the **final iteration in this Neural Network model development**.

NN Model (100)/19/9/4:

```
Classifier output
=== Run information ===

Scheme:      weka.classifiers.functions.Dl4jMlpClassifier -S 1 -cache-mode MEMORY -early-stopping "weka.dl4j.earlystopping.EarlySt
Relation:     Symbols
Instances:    200
Attributes:   2
              filename
              class
Test mode:    split 90.0% train, remainder test

=== Classifier model (full training set) ===

Network Configuration:
NeuralNetConfiguration(weightInit=XAVIER, biasInit=0.0, dist=weka.dl4j.distribution.Disabled@66, l1=NaN, l2=NaN, dropout=Dropout())
Model Summary:

=====
VertexName (VertexType)      nIn,nOut  TotalParams  ParamsShape      Vertex Inputs
=====
input (InputVertex)          -,-       -             -                 [input]
Dense layer 2 2 2 1 2 2 2 1 (DenseLayer) 100,19    1,919        W:[100,19], b:[1,19] [Dense layer 2 2 2 1 2 2 2 1]
Dense layer 2 2 2 1 2 2 2 2 (DenseLayer) 19,9      180          W:[19,9], b:[1,9]   [Dense layer 2 2 2 1 2 2 2 1]
Output layer (OutputLayer)    9,4       40           W:[9,4], b:[1,4]    [Dense layer 2 2 2 1 2 2 2 2]

-----
Total Parameters: 2,139
Trainable Parameters: 2,139
Frozen Parameters: 0
=====

Time taken to build model: 4.05 seconds
```

```
=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances      17          85  %
Incorrectly Classified Instances     3          15  %
Kappa statistic                     0.7993
Mean absolute error                  0.1108
Root mean squared error              0.2729
Relative absolute error              29.4817 %
Root relative squared error          62.8695 %
Total Number of Instances           20

=== Detailed Accuracy By Class ===

              TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
0.833    0.000    1.000    0.833    0.909    0.882    0.988    0.976    0
1.000    0.118    0.600    1.000    0.750    0.728    0.961    0.867    1
0.833    0.071    0.833    0.833    0.833    0.762    0.952    0.917    2
0.800    0.000    1.000    0.800    0.889    0.866    0.987    0.967    3
Weighted Avg.    0.850    0.039    0.890    0.850    0.857    0.819    0.973    0.940

=== Confusion Matrix ===

a b c d  <-- classified as
5 1 0 0 | a = 0
0 3 0 0 | b = 1
0 1 5 0 | c = 2
0 0 1 4 | d = 3
```


3.7 Improving Performance

3.7.1 Prediction (↑ data) vs Integrity (K-fold or Separate Test Data)

Fundamentally – far more data. If impracticable to significantly expand the dataset, Train-Test Percentile Splitting should be strictly avoided for balanced distributions – unless the model is trained sufficiently well at lower Train allocation.

The model's performance deceptively appears to *reduce* when executed via K-fold splitting. However, the integrity of the model is of paramount importance.

Whilst there may be a concerning decline in instances correctly classified, the model's aggregated performance can be seen below to encompass the *entire* dataset.

Furthermore, after 'K' models are fit and evaluated, the equivalent performance metric returned is the average (mean) of the total K-folded test executions (Brownlee 2020).

```
Classifier output
=== Run information ===

Scheme:      weka.classifiers.functions.Dl4jMlpClassifier -S 0 -cache-mode MEMORY -early-stopping "weka.dl4j.earlystopping
Relation:     Symbols
Instances:    160
Attributes:   2
              filename
              class

Test mode:    user supplied test set: size unknown (reading incrementally)

=== Classifier model (full training set) ===

Network Configuration:
NeuralNetConfiguration(weightInit=XAVIER, biasInit=0.0, dist=weka.dl4j.distribution.Disabled@66, l1=NaN, l2=NaN, dropout=0)
Model Summary:

=====
VertexName (VertexType)      nIn,nOut  TotalParams  ParamsShape  Vertex Inputs
=====
input (InputVertex)         -, -      -            -            [input]
Dense layer 2 2 2 2 1 2 2 1 (DenseLayer) 100,9     909          W:[100,9], b:[1,9] [Dense layer 2 2 2 2 1 2 2 1]
Dense layer 2 2 2 2 1 2 2 2 (DenseLayer) 9,10      100          W:[9,10], b:[1,10] [Dense layer 2 2 2 2 1 2 2 2]
Output layer (OutputLayer) 10,4      44           W:[10,4], b:[1,4]  [Dense layer 2 2 2 2 1 2 2 2]

-----
Total Parameters: 1,053
Trainable Parameters: 1,053
Frozen Parameters: 0
=====

Time taken to build model: 11.88 seconds
```

```
=== Evaluation on test set ===

Time taken to test model on supplied test set: 0.01 seconds

=== Summary ===

Correctly Classified Instances      31           77.5 %
Incorrectly Classified Instances    9           22.5 %
Kappa statistic                    0.7
Mean absolute error                 0.1086
Root mean squared error             0.3146
Relative absolute error             28.9659 %
Root relative squared error        72.6477 %
Total Number of Instances          40

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall   F-Measure  MOC     ROC Area  PRC Area  Class
-----
0.500    0.033    0.833    0.500    0.625    0.566    0.777    0.736    0
0.900    0.133    0.692    0.900    0.783    0.709    0.878    0.694    1
0.800    0.100    0.727    0.800    0.762    0.679    0.950    0.900    2
0.900    0.033    0.900    0.900    0.900    0.867    0.993    0.981    3
Weighted Avg.  0.775    0.075    0.788    0.775    0.767    0.705    0.900    0.828

=== Confusion Matrix ===

a b c d  <-- classified as
5 3 1 1 | a = 0
0 9 1 0 | b = 1
1 1 8 0 | c = 2
0 0 1 9 | d = 3
```

The distinction is crucial: it demonstrates the invalidity of comparing performance results between Train-Test techniques – it is not an ‘apples for apples’ comparison.

K-folding returns less optimistic estimates of the model's performance on small datasets than T-T Percentile splitting, given that it draws on randomly split data that likely fails to retain class proportionality.

Especially when considering the ‘majority classification’ issues of Percentage splits discussed previously.

3.7.2 Generalisation & Scale

The predictive strength of the model, and its iterations, have been extensively performed through broad Generalisation testing.

Executing this approach draws on peer collaboration, from which a partnership was arranged for input data sharing (Maher, 2022). This process allowed for independent data creation, model design and evaluation; after our respective independent models were sufficiently developed, they could then be tested on each other's datasets.

Building upon this theme, enhanced performance improvements would be realised by more diversification and scale in dataset design - perhaps drawn by more people, of a greater diversity of handwriting, for example.

3.7.3 Computational: Power, Bitrate & Bandwidth

Model input and processing improvements offer the next step-change performance potential. Increasing the number of pixels fed into neural nets carry very significant improvements in terms of broadening the model's input bandwidth, although carries inherent constraints.

While increasing the number of pixels equates to higher number of input layer neurons, and therefore a heavier and more powerful model; it also demands more intensive corresponding computational power.

Current technological hardware advancements in the ML and broader AI space are offering tremendous developments to handle these computation and power demands.

Cutting edge dedicated GPUs commonly address widespread demand for greater parallelism in NN processing capabilities; Analogue Computers offer the highspeed power delivery solutions; while Quantum computing is reaching commercial production price-points to meet AI industry requirements.

Greater hardware capacity opens further opportunity to get more 'bang for the buck'. One example is the opportunities that dedicated GPU processing delivers for increasing NN batch sizes.

This becomes most relevant when reducing training instances of image classification neural networks down to tens of thousands (Lavanya, 2019), and therefore serves NN models that are orders of magnitude greater in **scale** than the problem requirements at hand.

3.7.4 Increasing NN Scale for Larger Problems

The scale of the NN should be contingent upon the complexity of the problem needing to be solved.

Following on from the example provided in subsection 3.5.2, should the dataset be significantly expanded to accommodate a far more diverse range of symbol (or expanded set of character drawings), this would likely invoke a larger neural network.

Growing the network scale could be achieved through breadth of neurons per layer, or depth of increasing layers. Again, the option to pursue here is contingent upon the Problem requirements.

For instance, if the images requiring recognition are similar shape symbols but of higher resolution, then more layers may have greater impact to detect the finer recognition characteristics (greyscale arclength, degrees of curvature, etc); whereas a much greater diversity of shapes, or imagery more generally, would require at least much greater width and probably network depth as well.

4.0 Conclusion

Exploring Artificial Neural Network techniques of the deep learning subfield of machine learning, image recognition models are devised to predict the category of shapes inputted to the neural network.

A sequence of data science research processes is employed to collate and assemble 'hand drawn' images using basic computer graphic image editing software, into a repeating set of four simple shapes symbols.

The collection of processed image data is used to train a neural network image classification model, calibrated in accordance with specific results of initial model fitting to its input imagery data.

Key conclusions drawn centre around the marginal effects to the Neural Network - structural, functional and hyperparameter modifications - upon the model's ultimate performance.

Identified flaws in the K-fold cross validation algorithm – that it breaks down when applied to imbalanced class distributions - similarly affect the Percentage Split technique when small percentiles are allocated to model testing.

In the case of the K-fold approach, skewed distributions distort a model to focus predictive success upon more common classifications (Brownlee, 2020). However, with an obstructively small dataset in this case, the **Percentage Split technique cannot reasonably be applied with integrity** by similar logic.

To an untrained practitioner, early model performance metrics presented herein ostensibly indicated a well-fitting model, with 90-100% correctly classified instances. Closer inspection of these results reveals that the preliminary model repetitively fails to classify a single triangle image.

Redressing the issue by rebalancing the Train-Test division causes further issues by training the model upon few instances, such that prediction accuracy is so degraded that marginal Training set reductions converge to 'flip of the coin' classification results.

With the final expanded and aggregated dataset still far smaller than preferable, the solution must be for the allocation of distinct separate Test and Training datasets; enabling generalisable potential to train a rigorous Neural Network model upon unseen data.

Any model is only as good as the data fed into it, and the conditions specified to test that data.































































































































































































Meaningful data science insights must be derived with caution when machine learning datasets are small and Training set allocations are reduced to paltry Percentile Splits.

Artificial intelligence systems, with generalisation potential, will only carry the requisite import when models are truly representative of the data.

5.0 Appendix

5.1 Image Dataset

The complete input dataset is presented below. This dataset incorporates Richard Maher's (80) images combined with my own (120). Those generated by me can be identified by the shorter filenames.

									
circ1.bmp	circ2.bmp	circ3.bmp	circ4.bmp	circ5.bmp	circ6.bmp	circ7.bmp	circ8.bmp	circ9.bmp	circ10.bmp
									
circ11.bmp	circ12.bmp	circ13.bmp	circ14.bmp	circ15.bmp	circ16.bmp	circ17.bmp	circ18.bmp	circ19.bmp	circ20.bmp
									
circ21.bmp	circ22.bmp	circ23.bmp	circ24.bmp	circ25.bmp	circ26.bmp	circ27.bmp	circ28.bmp	circ29.bmp	circ30.bmp
									
Circle11_10x10.bmp	Circle12_10x10.bmp	Circle13_10x10.bmp	Circle14_10x10.bmp	Circle15_10x10.bmp	Circle16_10x10.bmp	Circle17_10x10.bmp	Circle18_10x10.bmp	Circle19_10x10.bmp	Circle20_10x10.bmp
									
Cross1_10x10.bmp	Cross2_10x10.bmp	Cross3_10x10.bmp	Cross4_10x10.bmp	Cross5_10x10.bmp	Cross6_10x10.bmp	Cross7_10x10.bmp	Cross8_10x10.bmp	Cross9_10x10.bmp	Cross10_10x10.bmp
									
Cross11_10x10.bmp	Cross12_10x10.bmp	Cross13_10x10.bmp	Cross14_10x10.bmp	Cross15_10x10.bmp	Cross16_10x10.bmp	Cross17_10x10.bmp	Cross18_10x10.bmp	Cross19_10x10.bmp	Cross20_10x10.bmp
									
Crs1.bmp	Crs2.bmp	Crs3.bmp	Crs4.bmp	Crs5.bmp	Crs6.bmp	Crs7.bmp	Crs8.bmp	Crs9.bmp	Crs10.bmp
									
Crs11.bmp	Crs12.bmp	Crs13.bmp	Crs14.bmp	Crs15.bmp	Crs16.bmp	Crs17.bmp	Crs18.bmp	Crs19.bmp	Crs20.bmp
									
Crs21.bmp	Crs22.bmp	Crs23.bmp	Crs24.bmp	Crs25.bmp	Crs26.bmp	Crs27.bmp	Crs28.bmp	Crs29.bmp	Crs30.bmp
									
sqr1.bmp	sqr2.bmp	sqr3.bmp	sqr4.bmp	sqr5.bmp	sqr6.bmp	sqr7.bmp	sqr8.bmp	sqr9.bmp	sqr10.bmp
									
sqr11.bmp	sqr12.bmp	sqr13.bmp	sqr14.bmp	sqr15.bmp	sqr16.bmp	sqr17.bmp	sqr18.bmp	sqr19.bmp	sqr20.bmp
									
sqr21.bmp	sqr22.bmp	sqr23.bmp	sqr24.bmp	sqr25.bmp	sqr26.bmp	sqr27.bmp	sqr28.bmp	sqr29.bmp	sqr30.bmp
									
Square1_10x10.bmp	Square2_10x10.bmp	Square3_10x10.bmp	Square4_10x10.bmp	Square5_10x10.bmp	Square6_10x10.bmp	Square7_10x10.bmp	Square8_10x10.bmp	Square9_10x10.bmp	Square10_10x10.bmp
									
Square11_10x10.bmp	Square12_10x10.bmp	Square13_10x10.bmp	Square14_10x10.bmp	Square15_10x10.bmp	Square16_10x10.bmp	Square17_10x10.bmp	Square18_10x10.bmp	Square19_10x10.bmp	Square20_10x10.bmp
									
tri1.bmp	tri2.bmp	tri3.bmp	tri4.bmp	tri5.bmp	tri6.bmp	tri7.bmp	tri8.bmp	tri9.bmp	tri10.bmp
									
tri11.bmp	tri12.bmp	tri13.bmp	tri14.bmp	tri15.bmp	tri16.bmp	tri17.bmp	tri18.bmp	tri19.bmp	tri20.bmp
									
tri21.bmp	tri22.bmp	tri23.bmp	tri24.bmp	tri25.bmp	tri26.bmp	tri27.bmp	tri28.bmp	tri29.bmp	tri30.bmp
									
Triangle1_10x10.bmp	Triangle2_10x10.bmp	Triangle3_10x10.bmp	Triangle4_10x10.bmp	Triangle5_10x10.bmp	Triangle6_10x10.bmp	Triangle7_10x10.bmp	Triangle8_10x10.bmp	Triangle9_10x10.bmp	Triangle10_10x10.bmp
									
Triangle11_10x10.bmp	Triangle12_10x10.bmp	Triangle13_10x10.bmp	Triangle14_10x10.bmp	Triangle15_10x10.bmp	Triangle16_10x10.bmp	Triangle17_10x10.bmp	Triangle18_10x10.bmp	Triangle19_10x10.bmp	Triangle20_10x10.bmp

5.2 Image Data Guide

The below guide (Monash College, 2022) was utilised for the creation of the symbolic Shape data files.

Consider the following tips when creating you drawings:

- Vary the placement of your drawing within the canvas
- Ensure each drawing passes the human check. Do you think it looks like a square/etc?
- If you're struggling to create a variety of symbols, ask someone else to draw some.
- Don't worry if some of the images are very similar, this just emphasises the desired shape for the network
- Don't be afraid to use some "perfect" examples of the shapes using the shape tools as well
- Similarly, your training data shouldn't try to capture every possible variation of the symbol as this will create an overfitted network that will struggle to generalise with data that falls outside of the test data.

5.3 References

- Brownlee, J. 2020, 'How to Fix k-Fold Cross-Validation for Imbalanced Classification', [MachineLearningMastery.com](https://machinelearningmastery.com/cross-validation-for-imbalanced-classification/) blog, <https://machinelearningmastery.com/cross-validation-for-imbalanced-classification/>
- Brownlee, J. 2020, 'Weka Machine Learning, How to Run Your First Classifier in Weka', Machine Learning Mastery, <https://machinelearningmastery.com/how-to-run-your-first-classifier-in-weka/>
- Hijazi S, Kumar R, Rowen C, 2015, 'Using Convolutional Neural Networks for Image Recognition', IP Group, Cadence
- Karpathy, A. 2015. "Neural Networks Part 1: Setting Up the Architecture." Notes for CS231n Convolutional Neural Networks for Visual Recognition, Stanford University. <http://cs231n.github.io/neural-networks-1>
- Karpathy, A. 2019. "A Recipe for Training Neural Networks", Andrej Karpathy blog. [A Recipe for Training Neural Networks \(karpathy.github.io\)](http://karpathy.github.io/2019/04/25/recipe/), <http://karpathy.github.io/2019/04/25/recipe/>
- Lang S, et al, 2019, WekaDeepLearning4j: a Deep Learning Package for Weka based on DeepLearning4j, In Knowledge-Based Systems, Volume 178, 15 August 2019, Pages 48-50. DOI: 10.1016/j.knosys.2019.04.013
- Lavanya 2019, Training a Neural Network? Start here!', LAVANYA AI blog, <https://lavanya.ai/2019/08/10/training-a-neural-network-start-here/>
- Maher, Richard 2022, 'Portfolio 3 – Neural Networks' report data, 'Introduction to Machine Learning - 2022 JAN', Monash College (Monash University)
- MathWork Help Centre, 'Imread: Read image from graphics file' [Read image from graphics file - MATLAB imread - MathWorks Australia](#)
- Monash College (Monash University) 2022, "Introduction to Machine Learning - 2022 JAN", Learning Management System, course content.
- Redmon, J. and Farhadi, A. (2018) YOLOv3: An Incremental Improvement. IEEE Conference on Computer Vision and Pattern Recognition, 89-95.
- Suryansh S, 2018, [Gradient Descent: All You Need to Know | HackerNoon](https://hackernoon.com/gradient-descent-aynk-7cbe95a778da), Hackernoon, <https://hackernoon.com/gradient-descent-aynk-7cbe95a778da>
- Weka, 2019, '[Training a CNN - WekaDeepLearning4j \(waikato.ac.nz\)](https://deeplearning.cms.waikato.ac.nz/examples/train_cnn/)', Weka Deep Learning documentation, https://deeplearning.cms.waikato.ac.nz/examples/train_cnn/
- Yunqian Ma & Haibo He, 2013, 'Imbalanced Learning: Foundations, Algorithms, and Applications', The Institute of Electrical and Electronics Engineers, Inc., p.188.