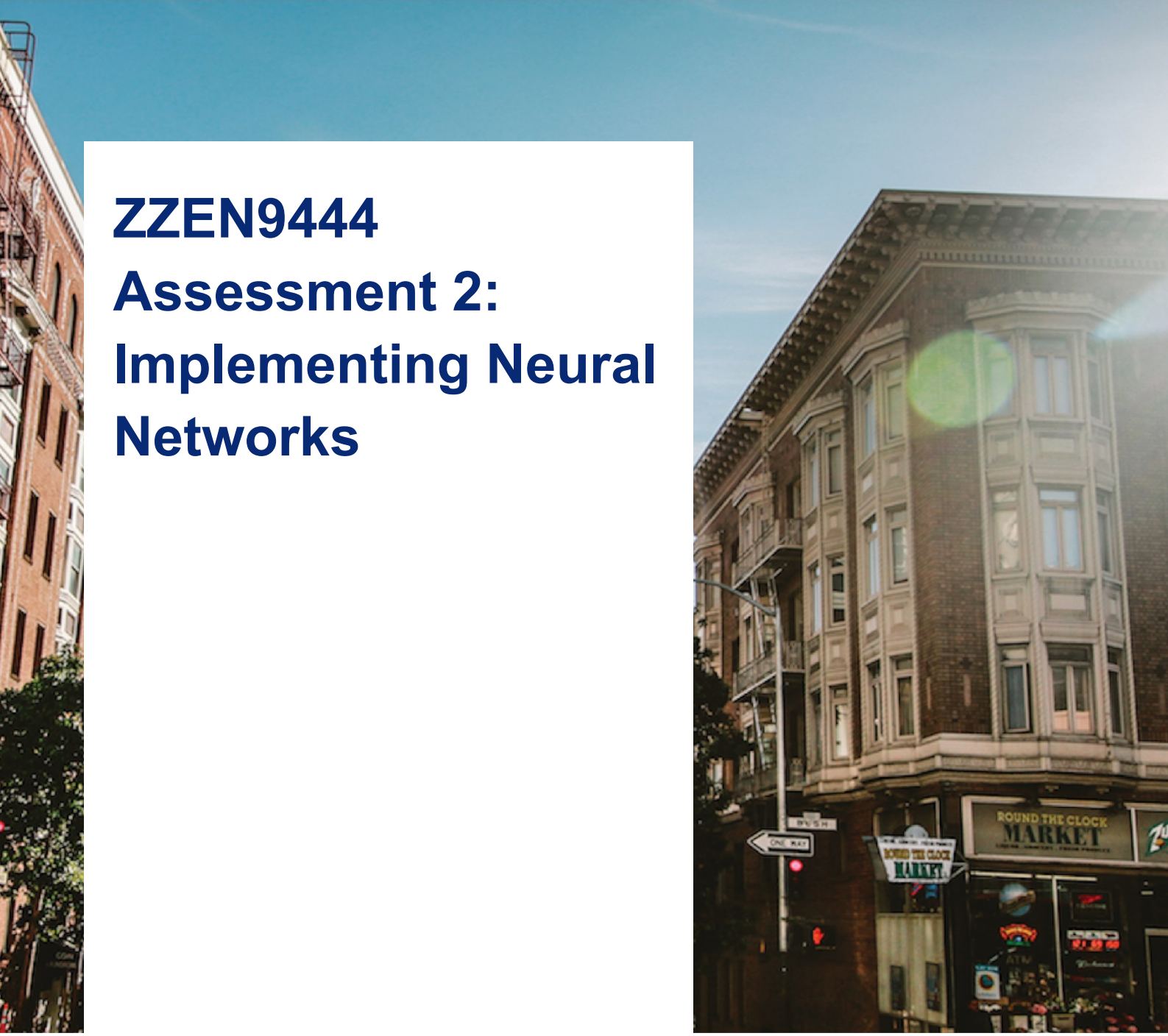


ZZEN9444 Assessment 2: Implementing Neural Networks

27 SEPTEMBER, 2021
PAUL JOHN CRONIN
Z5330951

PART 1. JAPANESE CHARACTER RECOGNITION

Step 1 – NetLin Model

The NetLin model was implemented as requested.

NetLin accuracy

Test set: Average loss: 1.0099, Accuracy: 6953/10000 (70%).

NetLin confusion matrix

	o	ki	su	tsu	na	ha	ma	ya	re	wo
o	768	6	8	12	30	64	2	62	29	19
ki	7	669	109	20	28	23	57	13	23	51
su	8	61	692	27	25	19	46	38	44	40
tsu	5	38	58	757	15	56	15	19	28	9
na	62	51	82	19	623	20	33	36	19	55
ha	8	27	124	17	20	725	26	10	33	10
ma	5	23	147	10	26	24	720	21	9	15
ya	16	28	25	13	87	17	53	626	87	48
re	11	39	97	44	6	29	43	6	704	21
wo	8	53	90	3	53	33	18	33	40	669

NetLin confusion matrix discussion

Being an extremely simple linear network, NetLin did not perform terribly well. Clearly the character “su” (Martins et al., 2019) is being confused with the characters “ha” and “ma”. This is because the characters “ha” and “ma” contain geometry similar to the “su” character – but not vice versa.

Step 2 – NetFull (optimal hidden nodes)

The NetFull model was implemented as desired.

To determine the optimal number of hidden nodes, multiple nets with different numbers of hidden nodes were tested, starting with 10 hidden nodes, then 20, 40, 80, 160, 320, 640 and even 748 (the number of inputs). It was clear that the classification accuracy initially increased with the number of hidden nodes, before rolling off, with additional hidden nodes greater than 320 given little extra benefit – in fact, it may have had a reduced test accuracy.

I then took additional data in the "roll-off" region, with accuracy measurements acquired with hidden nodes of 240, 220, 120 and 190. While there is some noise, the best result was at 240 hidden nodes, but little was gained in exceeding 160 hidden nodes. See Appendix A for data and plot.

With 240 hidden nodes, the results were as follows:

NetFull accuracy

Test set: Average loss: 0.4908, Accuracy: 8522/10000 (85%)

NetFull confusion matrix

	o	ki	su	tsu	na	ha	ma	ya	re	wo
o	857	4	2	5	29	28	3	37	30	5
ki	6	820	38	3	14	11	56	5	19	28
su	8	11	841	40	11	19	23	10	17	20
tsu	3	9	24	928	2	13	4	1	7	9
na	40	30	16	7	823	6	24	18	20	16
ha	8	15	72	10	11	841	19	1	16	7
ma	3	17	58	9	12	10	875	7	2	7
ya	21	19	21	3	14	12	24	838	19	29
re	8	30	28	40	5	7	27	6	842	7
wo	2	17	41	6	26	5	22	14	10	857

Step 3 – NetConv

The NetConv network, with convolution layers, was implemented as instructed.

NetConv accuracy

Test set: Average loss: 0.2650, Accuracy: 9520/10000 (95%)

NetConv confusion matrix

	o	ki	su	tsu	na	ha	ma	ya	re	wo
o	966	2	2	0	18	2	0	4	2	4
ki	3	943	12	0	5	2	17	1	5	12
su	10	5	926	28	5	8	6	2	2	8
tsu	0	0	11	968	1	6	6	0	4	4
na	19	7	2	11	920	12	7	2	8	12
ha	2	4	20	4	2	942	8	2	4	12
ma	3	3	10	0	3	5	973	0	0	3
ya	7	4	8	0	1	3	6	943	5	23
re	4	13	8	5	3	2	1	1	961	2
wo	5	5	4	3	2	0	1	1	1	978

Notes

The algorithm to obtain these results were:

1. A 2D convolution with 6 5x5 filters and two padding with ReLu,
2. A 2D max pooling of size 2,
3. Another 2D convolution, with 16 5x5 filters and two padding with ReLu,
4. Another 2D max pooling of size 2,
5. A linearization of the data,
6. A fully connected hidden layer with 240 hidden nodes, with ReLu,
7. Another linear function, given the ten output, with a softmax.

Several of the metaparameters were changed; increasing the learning rate to 0.1 and reducing the momentum to 0.25 improved the accuracy to 95%, but the selection of filters, metaparameters and overall architecture could be explored to a much greater extent with perhaps a corresponding slight increase in accuracy.

Step 4 – Discussion

The simplistic NetLin algorithm performs the worse (70% accuracy), then the two-layer NetFull model performs better, when a sufficiently large number of hidden nodes is incorporated (85% with 240 nodes), while the NetConv algorithm performs the best (with greater than 95% accuracy).

An interesting result from evaluating the confusion matrices is that they are not symmetric. For example, the "su" character often gets confused with other characters (such as "ha" and "ma", but these other characters do not get confused with the "su" character equally.

The way I think about these various approaches is to visualize each result in a multidimensional space. The linear approach can only separate the clusters with lines or planes – which gives a poor result. The NetFull, with at least one hidden layer, allows those clusters to be separated by curves, rather than planes, giving a better result. Finally, the convolution network is even more abstract, finding ways to separate each cluster further from other clusters, by creating new metrics, allowing for a cleaner division.

A little research (Clanuwat et al., 2018) shows that this data set and classification is actively being examined with many approaches, including branching / merging CNNs, ensembling, efficient CapsNet, SOPCNN, Tsetlin machines, etc.

One popular approach was some variation on a ResNet – where the results from earlier layers were directly connected to the current nodes. I thought I would experiment with the simplest possible such ResNet, implementing a slight modification of the NetFull algorithm, where the initial input feeds were also fed to the final connected layer. I called this extraordinary simple residual net "NetFullRes".

I had expected a slight improvement to the NetFull algorithm, but I was surprised with the result:

NetFullRes Accuracy

Test set: Average loss: 0.2482, Accuracy: 9383/10000 (94%)

NetFullRes Confusion matrix

	o	ki	su	tsu	na	ha	ma	ya	re	wo
o	954	5	1	1	26	1	1	9	0	2
ki	1	931	3	0	7	3	35	4	3	13
su	8	10	889	22	6	20	10	20	5	10
tsu	2	2	14	948	4	11	5	6	3	5
na	15	12	3	6	938	6	7	5	6	2
ha	3	11	38	5	6	913	11	3	3	7
ma	3	2	14	2	8	4	964	1	0	2
ya	9	6	1	1	3	1	7	952	2	18
re	8	18	6	2	11	3	5	3	938	6
wo	8	3	7	0	7	0	5	12	2	956

NetFullRes Discussion

As can be seen, this simple linear residual network nearly gave the best result of all nets tested, only the more complex and sophisticated NetConv was slightly superior to NetFullRes. This unexpected result shows the true power of residual networks – something that I would be very curious to explore further.

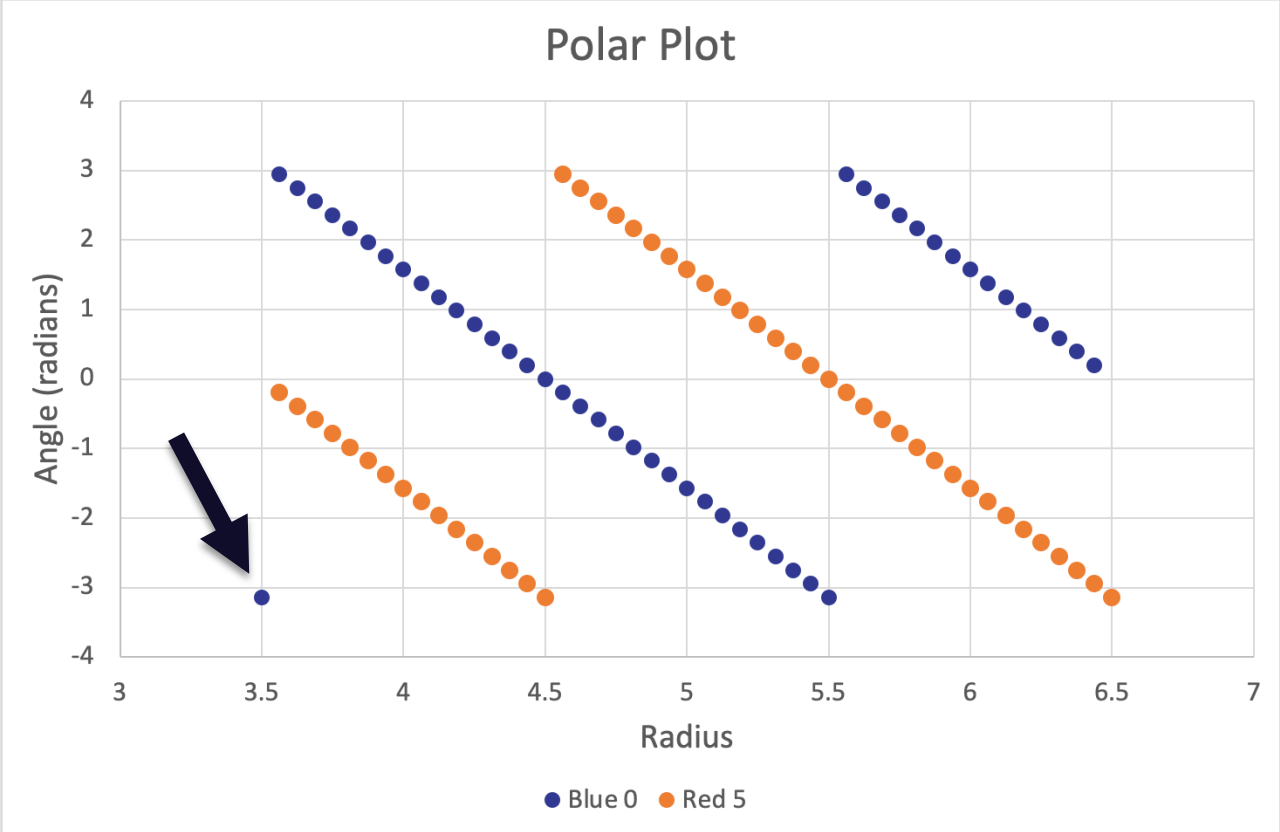
Part 2. Twin Spirals

Step 1 – PolarNet implementation

PolarNet was implemented in spiral.py.

Please see the figure below, where the spirals are imperfectly “unwrapped”, that is, plotted in polar coordinates. As can be seen, there are discontinuities in the lines due to the atan2 function.

Of great importance is the point with radius 3.5 and angle -π. That singular point is another quirk of the atan2 function, which becomes important when the minimum number of hidden nodes is discussed. It is possible that point could have been at radius 3.5 with angle π, which would change the next section analysis.



Step 2 – optimize hidden nodes

Minimum hidden nodes

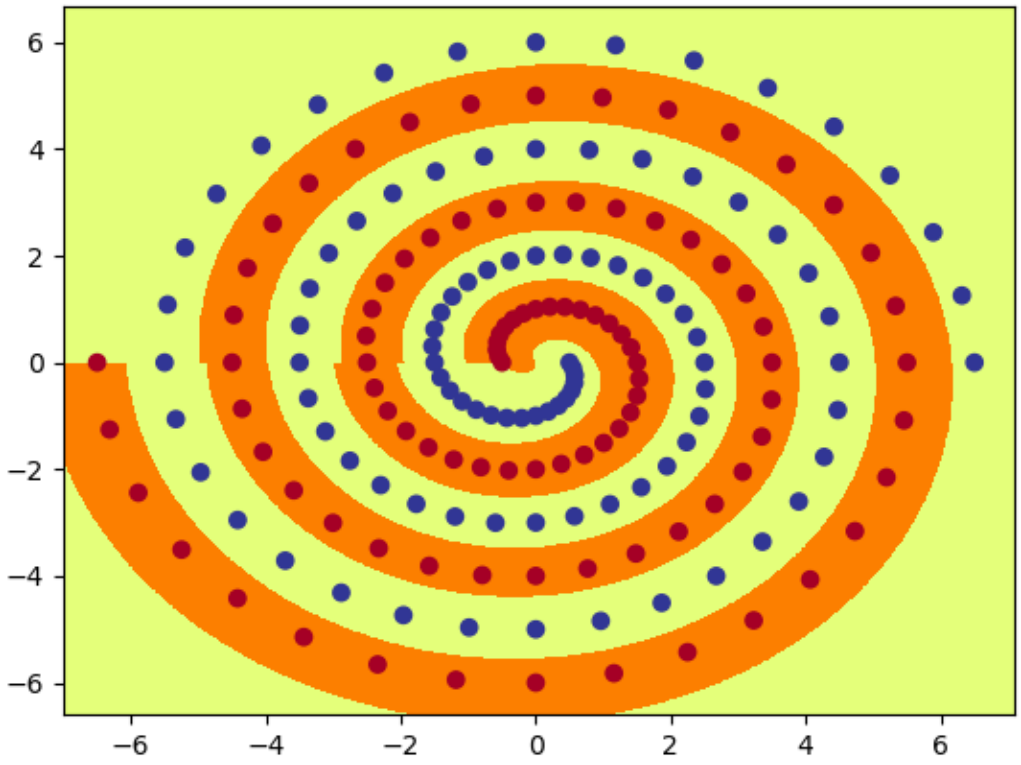
I found the minimum number of hidden nodes required to accurately classify all the training data consistently, within 20000 epochs, was 6.

However, this value of 6 is due to that single datapoint at radius 3.5 and angle -π. The atan2 function arbitrarily gives an angle of -π, however, it would be equally accurate to be +π.

I went into the code and artificially forced this value to be +π, which resulted in the number of necessary nodes to be only 5.

PolarOut plot

The resulting PolarOut plot is below:



Step 3 – Implement RawNet

The RawNet module was implemented in spiral.py as desired.

Step 4 – RawNet optimization

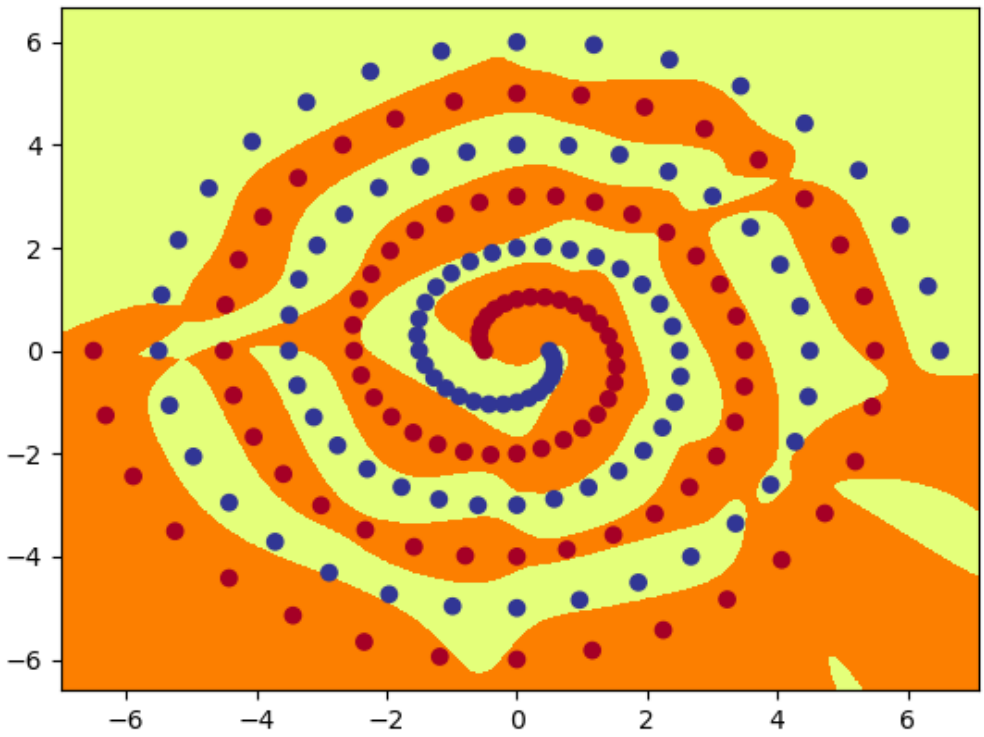
Parameters

While it was possible to achieve 100% accuracy with 9 hidden nodes semi-regularly, I found that 10 hidden nodes would be necessary for achieving 100% accuracy consistently in under 20,000 epochs.

Additionally, a learning rate of 0.011 and an initial weight of 0.15 seemed to minimize the number of epochs necessary to achieve 100% accuracy. The net was very sensitive to these values – variation from these values quickly caused the number of epochs necessary to exceed the 20,000 limit.

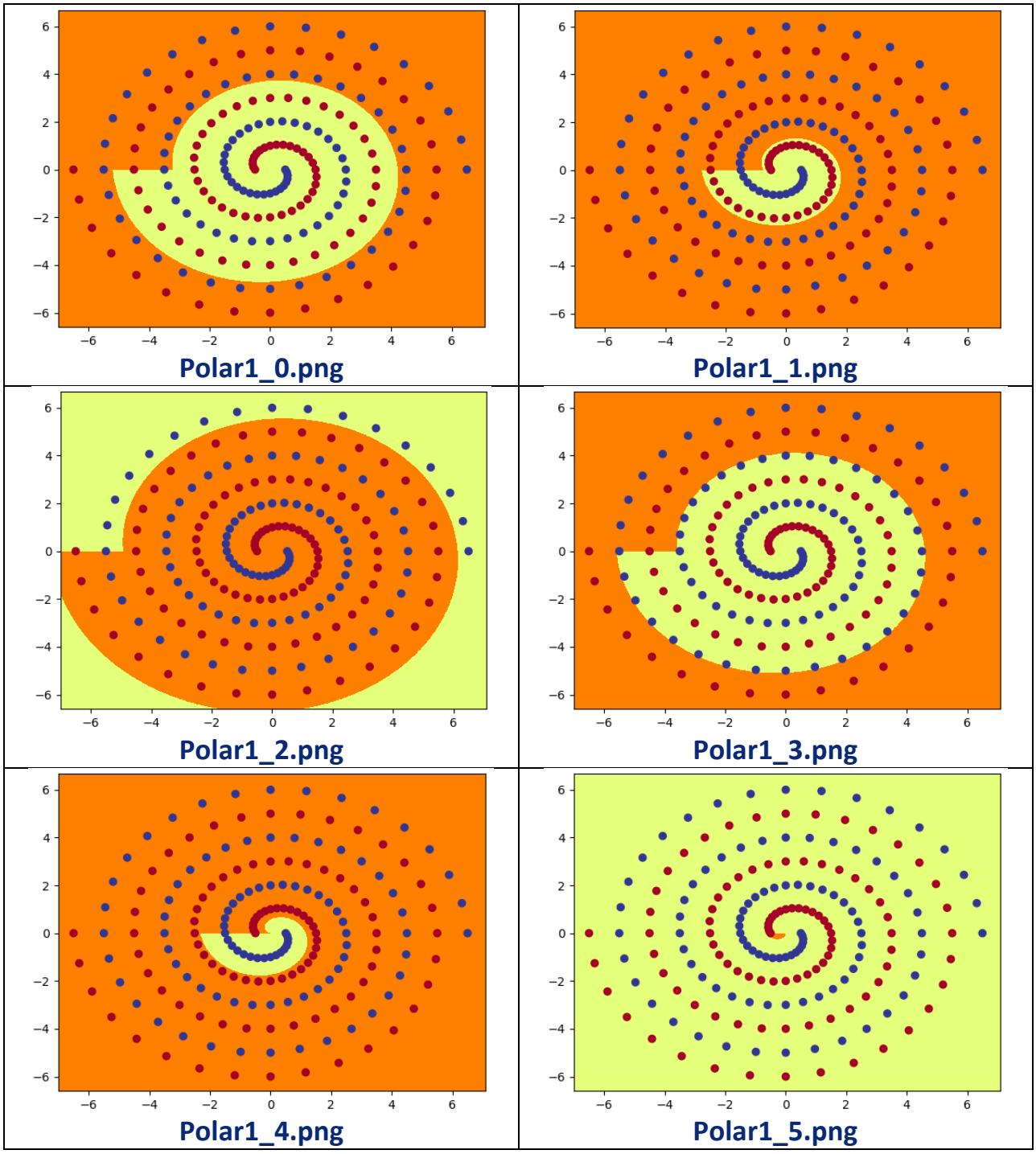
Graph_output

The following figure was produced by graph_ouput() showing the classification.



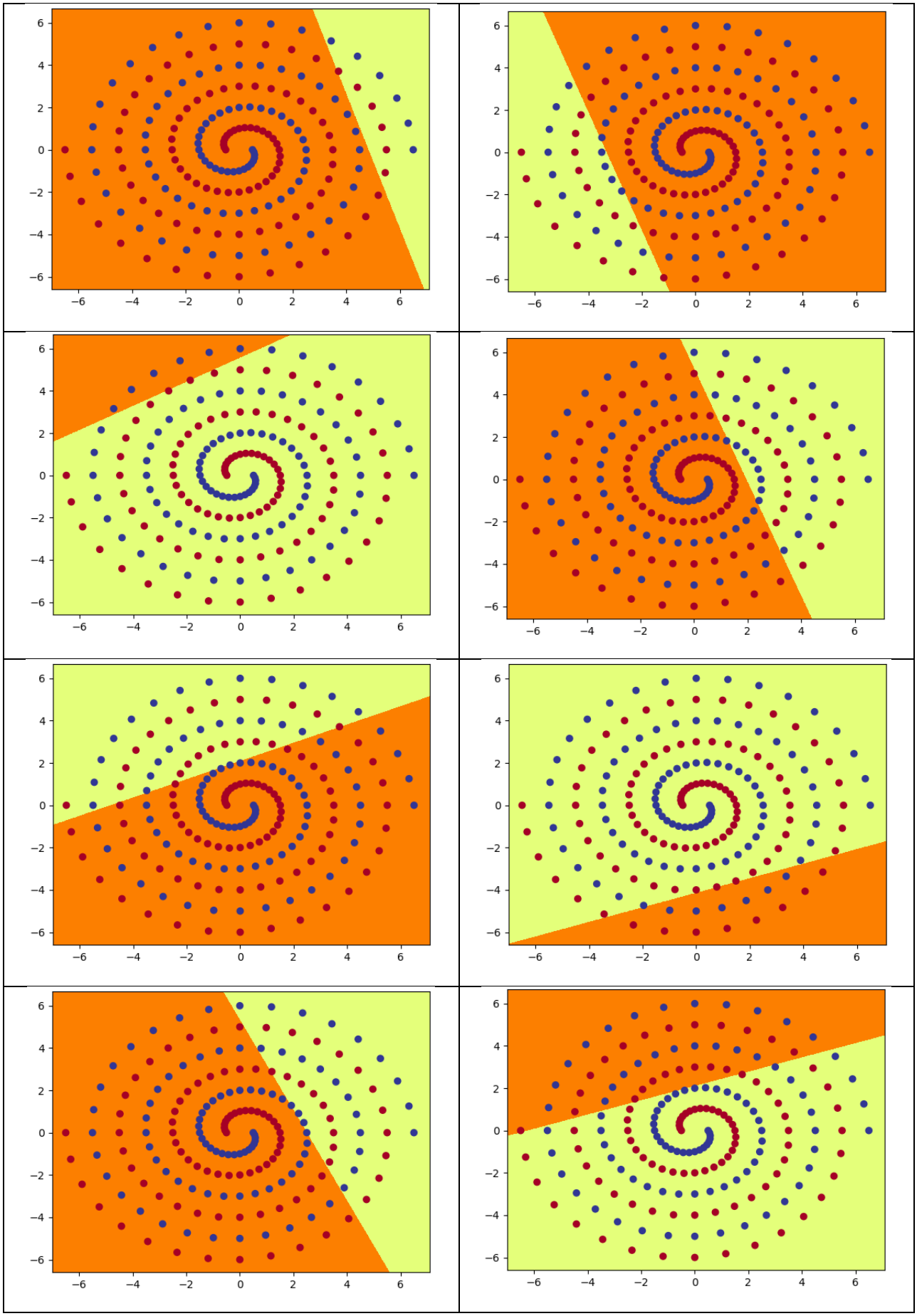
Step 5 – graph_hidden() implementation

PolarNet plots

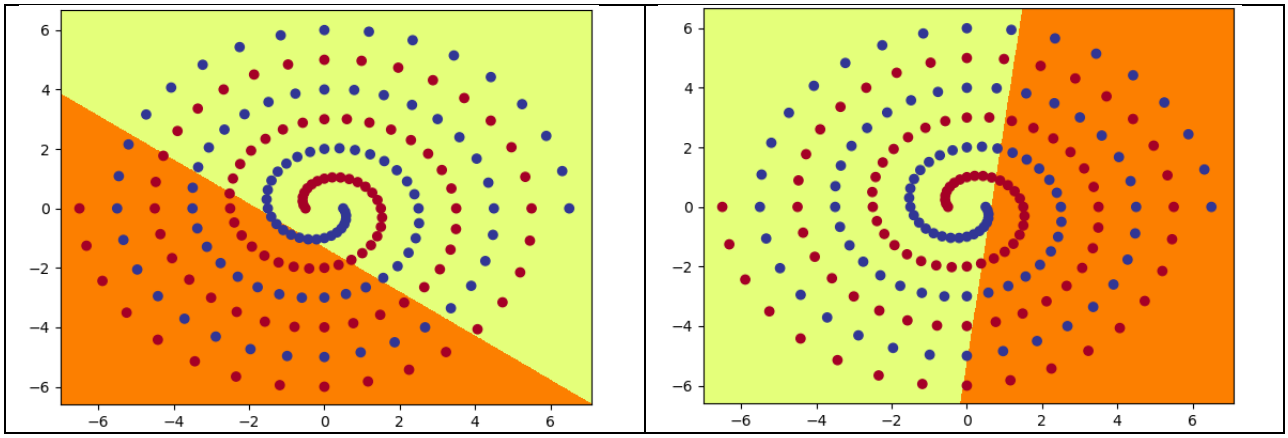


Note the single data point near the origin in Polar1_5, that requires an additional hidden node to account for it – this is an arbitrary quirk of atan2(). A more sophisticated phase unwrapping function (Asundi, 2002) would solve this, reducing the number of necessary hidden nodes to 5.

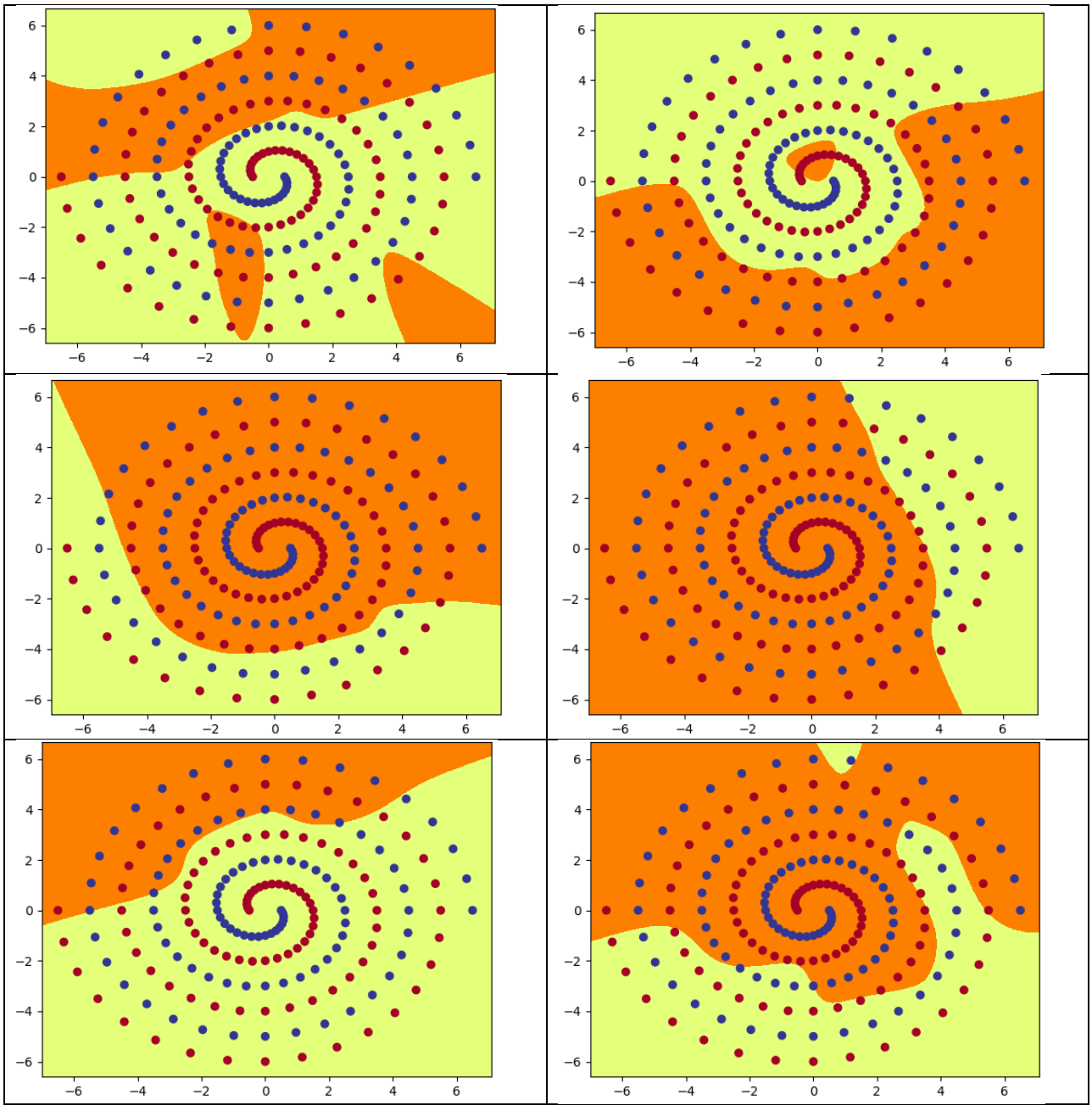
RawNet plots (1_0 – 1_7)



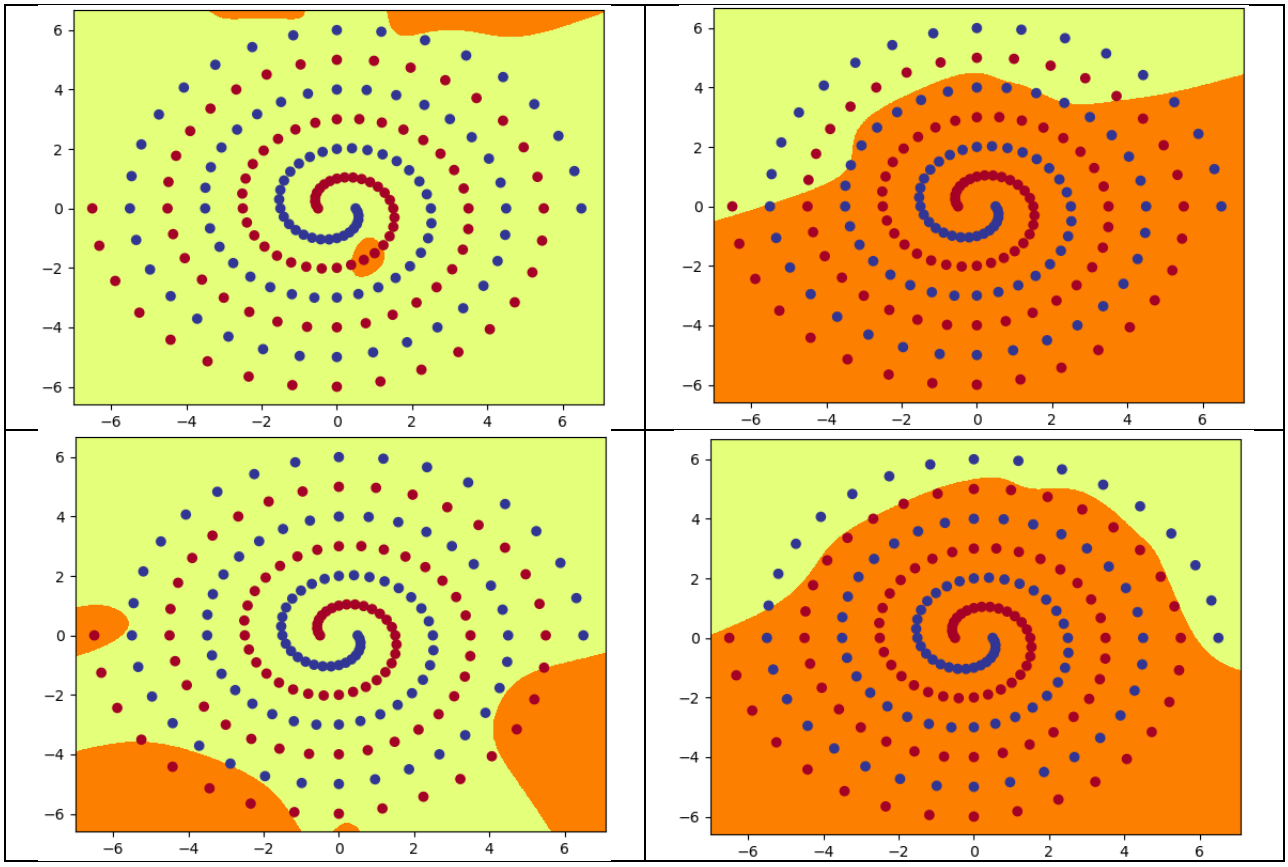
RawNet plots (1_8 – 1_9)



RawNet plots (2_0 – 2_5)



RawNet plots (2_6 – 2_9)



Step 6 – Discussion

Examining the difference between PolarNet and RawNet, it is clear that PolarNet can achieve the same goals with only one layer, instead of RawNet's two layers, and with many fewer nodes. One might argue that PolarNet can only achieve this result by "cheating", with the transformation of the spatial data to polar co-ordinates. I don't think that's true – the data is the same, just transformed in form for human consumption. In fact, with a "phase unwrapping function" more sophisticated than atan2(), such as that used in optics (Asundi, 2002), classification might be done with as little as two hidden nodes. I think this says something about neural nets themselves – we humans prefer cartesian co-ordinates, and that bias may be present in how we build, implement and use neural nets.

The qualitative difference between PolarNet and RawNet is that PolarNet slices the multidimensional data space with lines (or planes), which it can do successfully because the spirals in polar space form perfect lines. RawNet, with its single hidden

layer, carves out complex curves in cartesian space that can “slip between” datapoints to connect the members of the class. This process of slipping between datapoints is “brittle”, any small deviation will include a member of the incorrect class.

There is a balance between several factors when exploring the effects of initial weight and learning rate for RawNet, especially in a complex, noisy space of the twin spirals, especially as it is laden with many local minimums. Too slow a learning rate, or too high an initial weight, and the 100% accuracy is never achieved before the 20,000 epoch limit. It is quite easy to spend many epochs trapped in a local minimum.

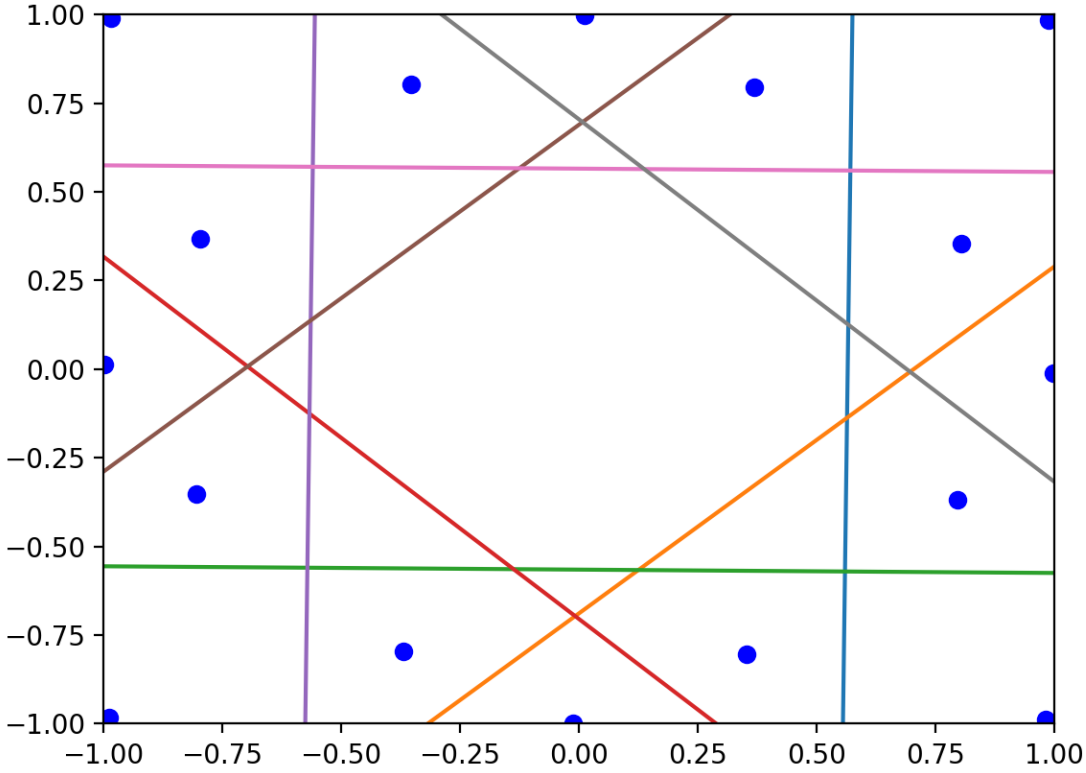
Additional tests were undertaken:

- When the learning rate is increased to 0.11, the net does not converge to 100% - rather it seems to rattle around 80% accuracy permanently.
- Changing the batch size from 97 to 194 increased the speed of each epoch but seems to make the net more sensitive to local minimums. Decreasing the batch size to 48 slowed each epoch.
- The net did not converge to 100% accuracy within 20,000 epoch when SGD optimization was utilized instead of Adam optimization, no matter how the learning rate, the momentum or even batch size were changed. Hence, it was concluded that, for this problem at least, ADAM is superior to SGD.
- Likewise, changing from tanh to ReLu also failed, so again, for this example, tanh is superior to ReLu.
- Adding a third hidden layer (with the same number of hidden nodes), enabled the number of hidden nodes to be reduced from 10 to 7, while still achieving 100% accuracy in less than 20,000 epochs.

Part 3 – Hidden unit dynamics

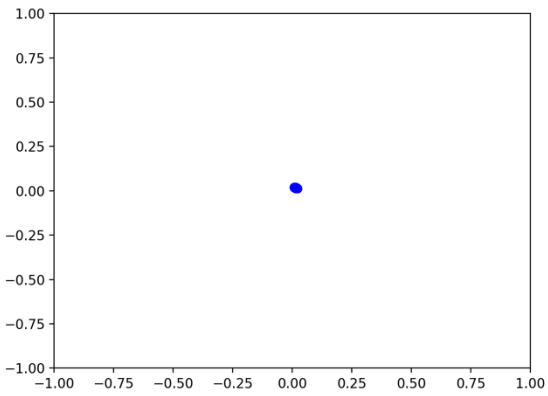
Step 1 – Star16 image

The code was executed, generating this image, as desired.

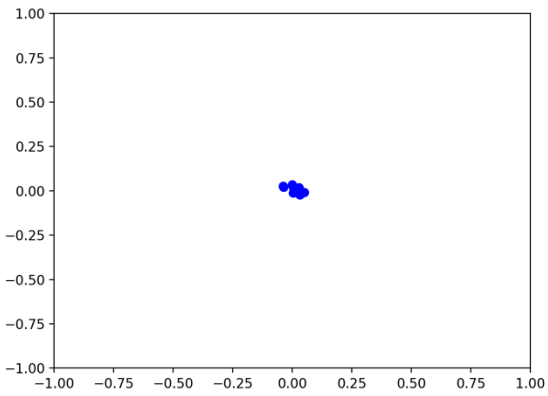


Step 2 – Encoder evolution 9-2-9

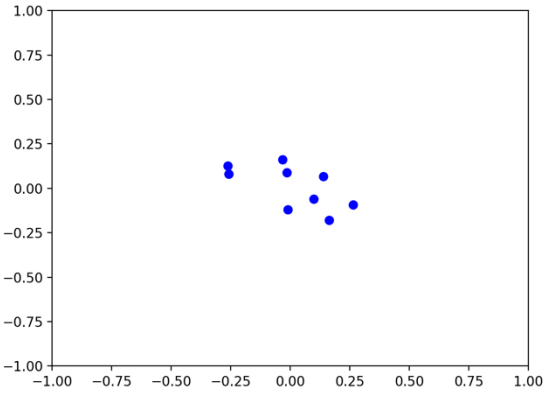
The images generated for epochs 50 – 300, plus the final image, as desired.



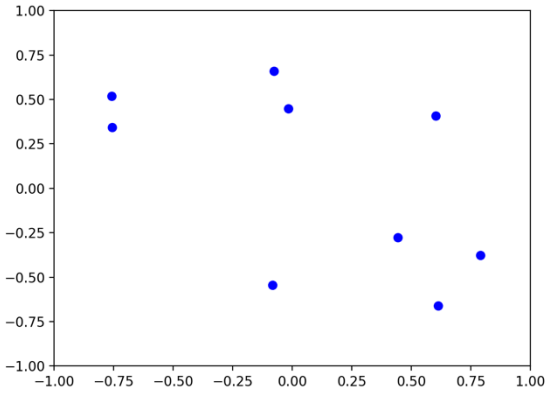
Epoch 50



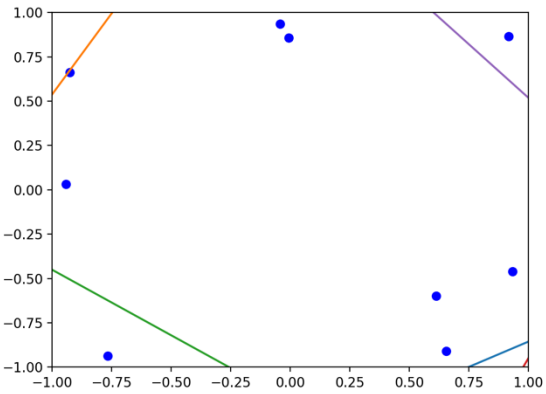
Epoch 100



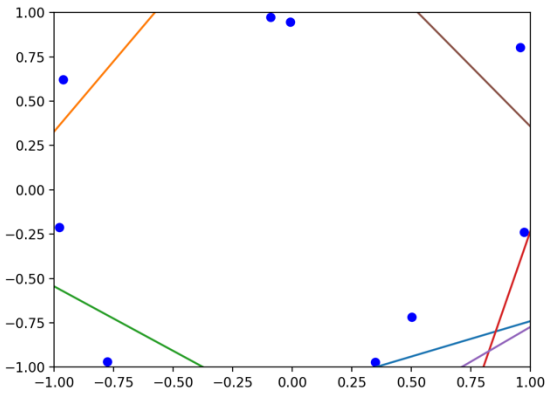
Epoch 150



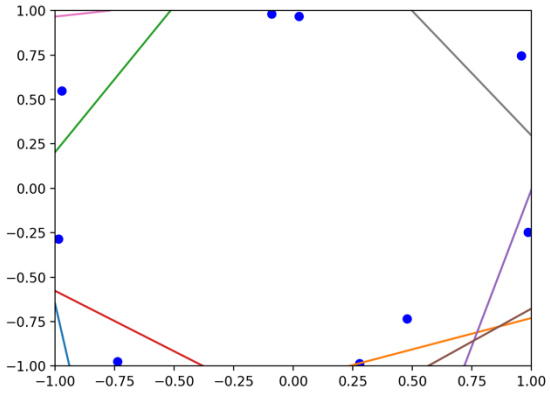
Epoch 200



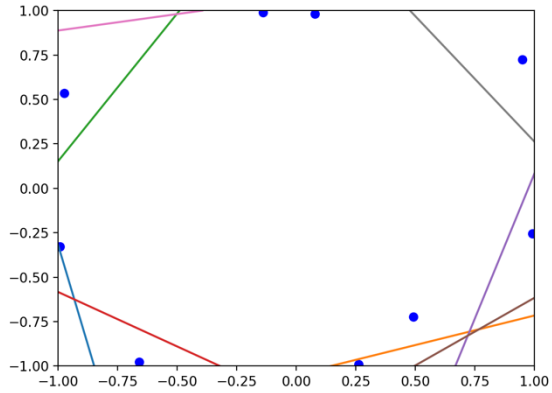
Epoch 300



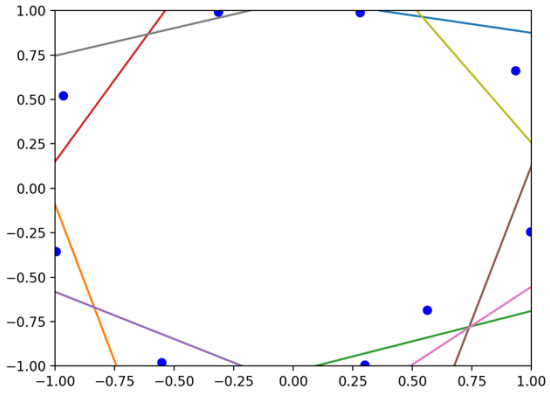
Epoch 500



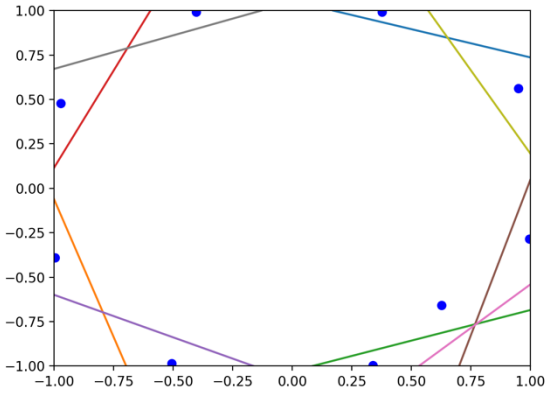
Epoch 700



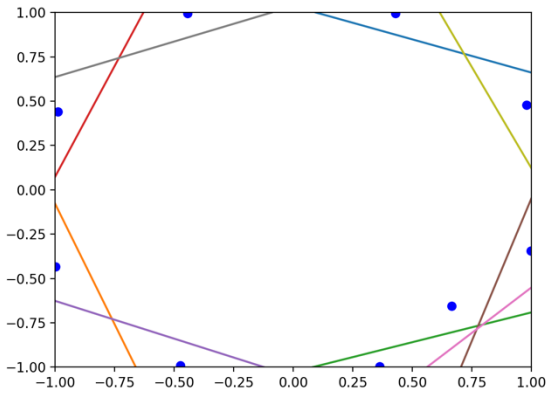
Epoch 1000



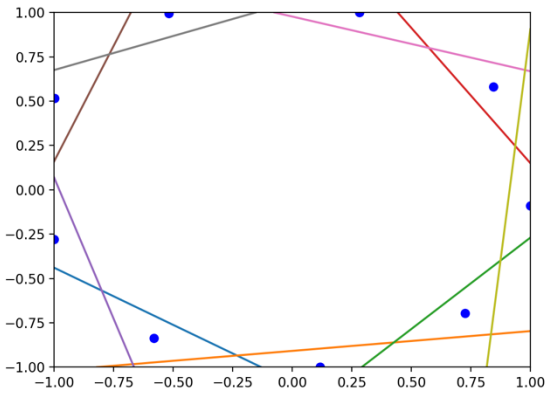
Epoch 1500



Epoch 2000



Epoch 3000



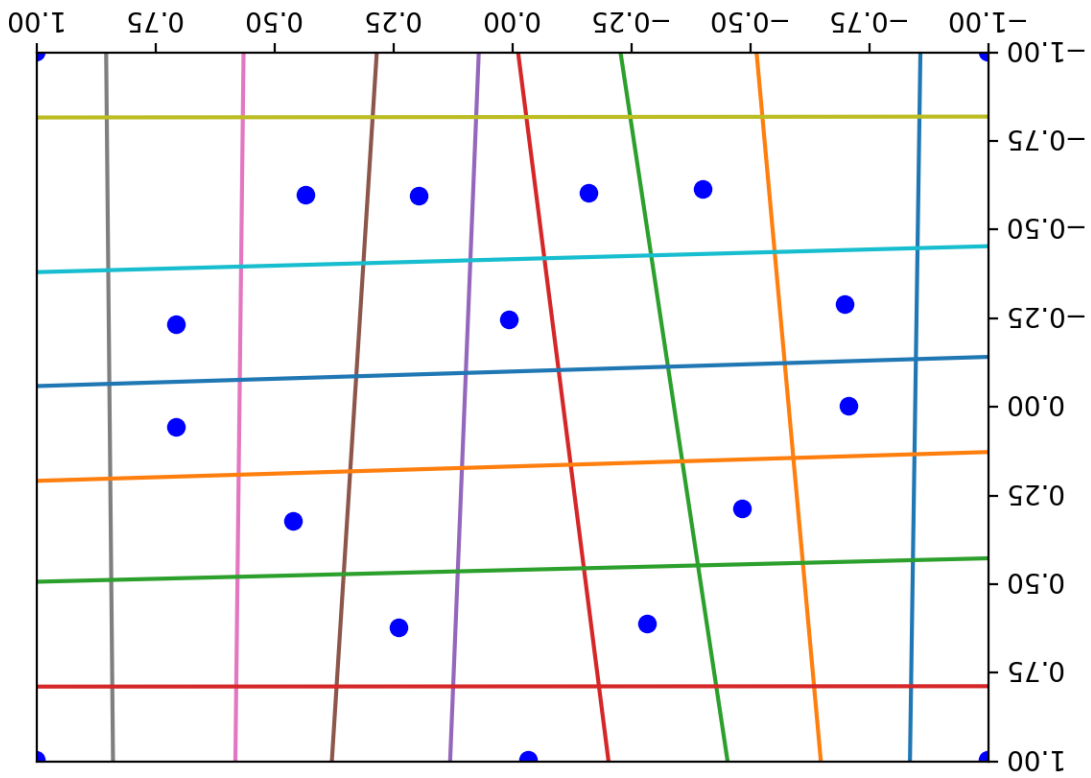
Final

As the initial weights are 0.001, and the activation is tanh, the initial distance from the origin is very small, as tanh(0.001) ≈ 0. If the code is executed with larger initial weights, the unit activations are greater than zero, and the data points are no longer clustered around the origin.

The lines (output boundaries move) from outside the viewing window closer towards the data points (hidden unit activations), and the data points move relative to the lines, looking to minimize the error with the target tensor.

Step 3 – Hearts

The tensor to generate the heart shape was encoded by hand, producing the result below. Please note I rotated the result to give the desired image.



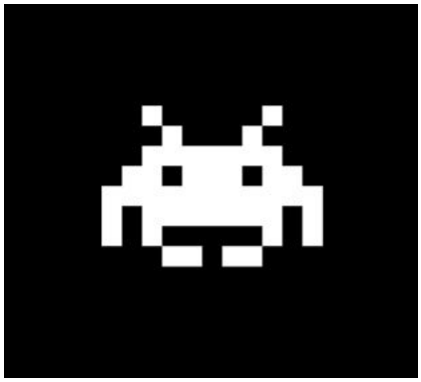
Step 4 – Create tensor art!

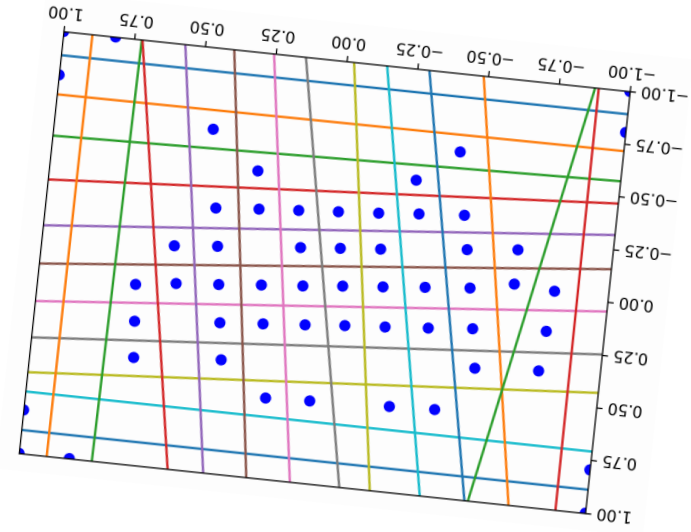
This section asked for two tensor sets to be generated, such that when executed, art is formed – it did not specify that these tensors had to be created by hand. Hence, I wrote code that converted black & white, two-dimensional images into the tensors structure (see Appendix B for the code).

Large black and white images were originally chosen (the UNSW logo – see Appendix B.1), however the processing time was exorbitant. Hence a switch to smaller images chosen and processed: the “crab” character from Space Invaders, and a pixel art drawing of Yoda. Note that “corner hidden unit activations” were added to the Space Invader image to improve image recognizability.

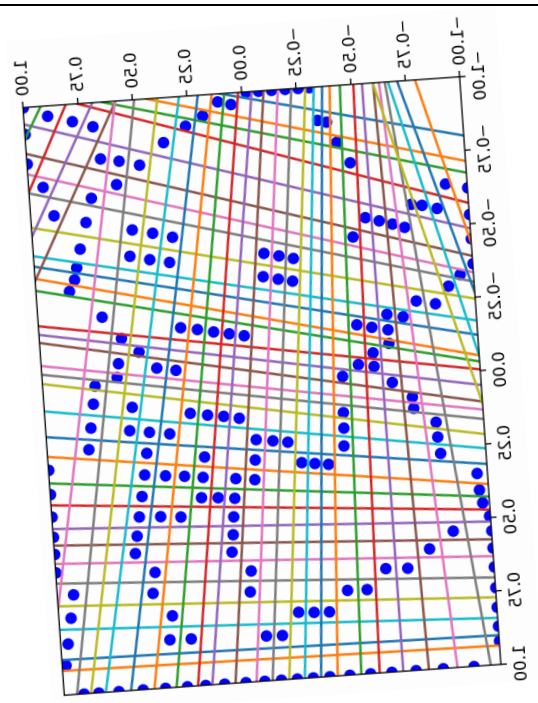
Both were processed, resulting in tensor formed images (rotated for best visualization). The original black and white images are also shown.

Target 1 – Space Invaders[1]





Target 2 – Yoda[2]



[1] The original Space Invaders characters were created by Tomohiro Nishikado in 1978. https://spaceinvaders.fandom.com/wiki/Crab_(Medium_Invader)

[2] This line drawing of Yoda from: https://www.pinclipart.com/pindetail/bxwRih_yoda-pixel-art-star-wars-yoda-clipart/

Bibliography

Asundi, A.K., 2002. MATLAB® for Photomechanics (PMTOOLBOX), in: MATLAB® for Photomechanics- A Primer. Elsevier. https://doi.org/10.1016/B978-008044050-7/50050-1

Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., Ha, D., 2018. Deep Learning for Classical Japanese Literature [WWW Document]. Papers with code. URL https://paperswithcode.com/sota/image-classification-on-kuzushiji-mnist (accessed 9.27.21).

Martins, L., Júnior, M., Serapião, A., 2019. Classificação de imagens de ideogramas Kuzushiji com Redes Neurais Convolucionais, in: Anais Do Encontro Nacional de Inteligência Artificial e Computacional (ENIAC 2019). Sociedade Brasileira de Computação - SBC. https://doi.org/10.5753/eniac.2019.9293

Appendix

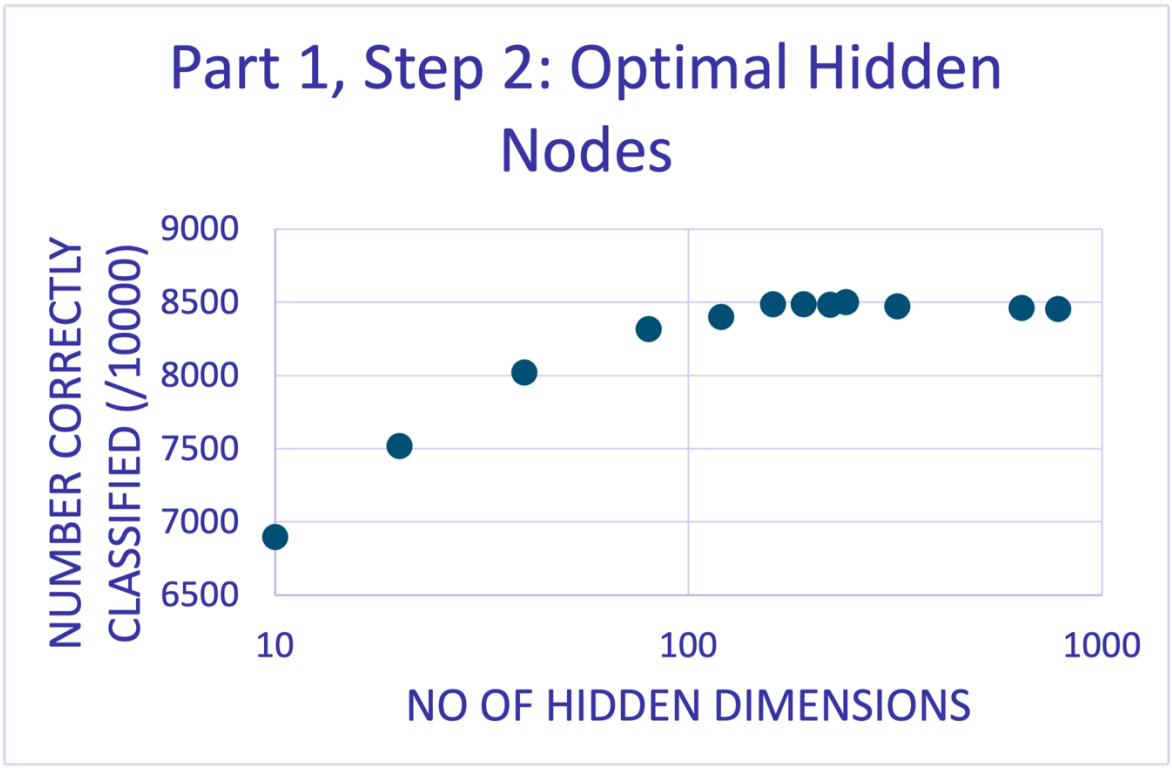
Appendix A – Optimal hidden nodes

This appendix gives the data taken and plot generated from Part 2, Step 2.

Appendix A.1 - Data

Hidden node	Accuracy (/1000)
10	6898
20	7519
40	8019
80	8315
160	8486
320	8470
640	8460
240	8502
220	8483
120	8401
784	8452
190	8487

Appendix A.2 - Plot



Appendix B – Tensor art

Appendix B.1 - Image to tensor code

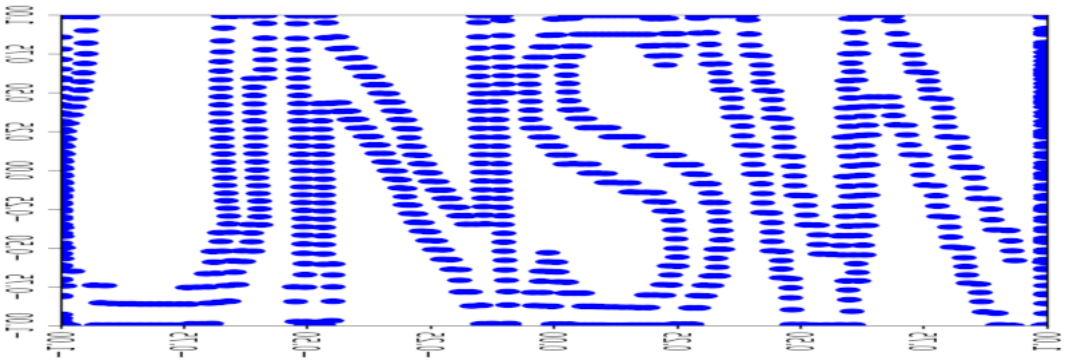
The following piece of code accepts a two dimensional, 1-bit image array, and produces an output in a quasi-tensor form (with some ‘find & replace’ text modifications necessary), which could then be inserted into the encoder.py code.

```
dims = size(data);
out =zeros(sum(sum(data)),dims(1)+dims(2));
count = 0;
for x=1:dims(1)
   for y=1:dims(2)
      if data(x,y)==1
         count = count+1;
         index = 1;
         out(count,index)=2;
         for s=2:dims(1)
            index = index+1;
            if x<s
               out(count,index)=1;
            end
         end
         for t = 2:dims(2)
            index = index+1;
            if y<t
               out(count,t+dims(1))=1;
            end
         end
         index = index+1;
         out(count,index)=3;
      end
   end
end
```

Appendix B.2 - Unused artwork (UNSW logo)

Initially, a large format image was chosen – that being the UNSW logo below.

However, this image had 900 hidden unit activations, and was taking far too long to process. I paused it after a day, giving the result below. Please note, for clarity, I removed the output boundaries.



Given sufficient time, I am confident the result would have been correct.

While the overall number of hidden unit activations comprising this image would remain constant, I believe that the number of output boundaries could have been significantly reduced if the grid digitization was not used: for example, diagonal output boundaries could define the strokes of the 'N' or 'W', and possibly circular boundaries could define the curves of the 'S'.

The CSV file containing the tensor for this image is available upon request – but it is quite large.