Project 1

Part 1

Question 1

The screenshot below shows the confusion matrix and the final accuracy is **70%**.

```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.820638
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.635077
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.600236
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.607185
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.322518
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.519428
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.668361
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.613730
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.349444
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.670886
<class 'numpy.ndarray'>
[[768.   5.   7.  12.  30.  63.   2.  63.  31.  19.]
 [  7. 670. 108.  18.  28.  24.  59.  13.  24.  49.]
 [  7.  62. 695.  25.  26.  20.  45.  37.  45.  38.]
 [  4.  36.  62. 755.  16.  58.  13.  18.  27.  11.]
 [ 61.  50.  81.  19. 627.  18.  34.  34.  20.  56.]
 [  8.  29. 123.  17.  19. 725.  28.   7.  33.  11.]
 [  5.  24. 147.   9.  25.  24. 722.  21.  10.  13.]
 [ 16.  28.  26.  12.  86.  17.  54. 622.  91.  48.]
 [ 11.  37.  94.  43.   7.  29.  44.   7. 705.  23.]
 [  8.  52.  83.   3.  55.  33.  18.  31.  41. 676.]]

Test set: Average loss: 1.0104, Accuracy: 6965/10000 (70%)
```

Question 2

Here is the confusion matrix. The number of hidden nodes is 100 and the final accuracy is **84%**.

```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.383490
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.271974
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.218164
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.217182
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.112650
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.251787
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.249270
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.316001
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.140767
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.290292
<class 'numpy.ndarray'>
[[847.   3.   2.   8.  27.  33.   4.  36.  32.   8.]
 [  5. 806.  40.   6.  20.  10.  60.   2.  22.  29.]
 [  6.  27. 834.  37.  12.  18.  24.  12.  15.  15.]
 [  4.  14.  24. 921.   2.   9.   6.   5.   5.  10.]
 [ 41.  31.  21.   8. 802.   9.  28.  18.  20.  22.]
 [  9.  10.  75.  14.  14. 832.  25.   3.  11.   7.]
 [  3.  14.  62.   9.  12.   9. 874.   6.   3.   8.]
 [ 19.  14.  19.   6.  23.  11.  38. 806.  20.  44.]
 [ 13.  23.  30.  55.   2.  10.  29.   3. 828.   7.]
 [  5.  23.  42.   4.  39.   6.  24.  15.  11. 831.]]

Test set: Average loss: 0.5251, Accuracy: 8381/10000 (84%)
```

Question 3

The screenshot below shows the confusion matrix and the final accuracy is **93%**.

```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.048384
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.012531
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.082190
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.037992
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.031331
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.052708
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.038554
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.147762
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.016352
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.059846
<class 'numpy.ndarray'>
[[956.   5.   1.   0.  22.   1.   0.   9.   3.   3.]
 [  2. 911.   8.   0.  14.   1.  42.   5.   5.  12.]
 [ 10.   5. 904.  33.   8.  10.  12.   5.   5.   8.]
 [  2.   3.  14. 959.   0.   9.   6.   4.   1.   2.]
 [ 25.   5.   6.   7. 926.   0.  10.   5.  10.   6.]
 [  4.   8.  65.  10.   7. 878.  15.   4.   3.   6.]
 [  2.   2.  17.   2.   8.   2. 961.   4.   0.   2.]
 [  8.   1.  14.   0.   3.   1.   8. 947.   6.  12.]
 [  1.  15.  12.   0.   7.   2.   9.   1. 952.   1.]
 [  8.   2.  12.   2.   9.   1.   8.   7.  13. 938.]]

Test set: Average loss: 0.2531, Accuracy: 9332/10000 (93%)
```

Question 4

a. The relative accuracy of **NetConv** is the highest.

   The result of model **NetConv** is the most accurate among the three models, its final accuracy is **94%**. Next co**mes to** model **NetLFull (84%)**. The accuracy of the model **NetLin** is **70%**.

b. The table below shows the confusing words according to the confusion matrix for each model.

Model	True characters	Predict characters
NetLin	ma	su
NetLFull	ha	su
NetConv	ha	su

   We can find that the mistakes are usually caused by high shape similarity between two characters.

c. ① **Based on model NetFull**

	num_of_hid_nodes	final accuracy
NetFull	10	68
	100	84
	1000	85
	10000	83

As the number of hidden nodes increasing, the final accuracy improves as well, while when the number of hidden nodes increase to a certain extent, the final accuracy will drop.

② **Based on model NetConv**

Two convolutional layers use max pooling (screenshot in question3), the accuracy is around **93%**.

While without max_pool layer, the accuracy is around **91%**.

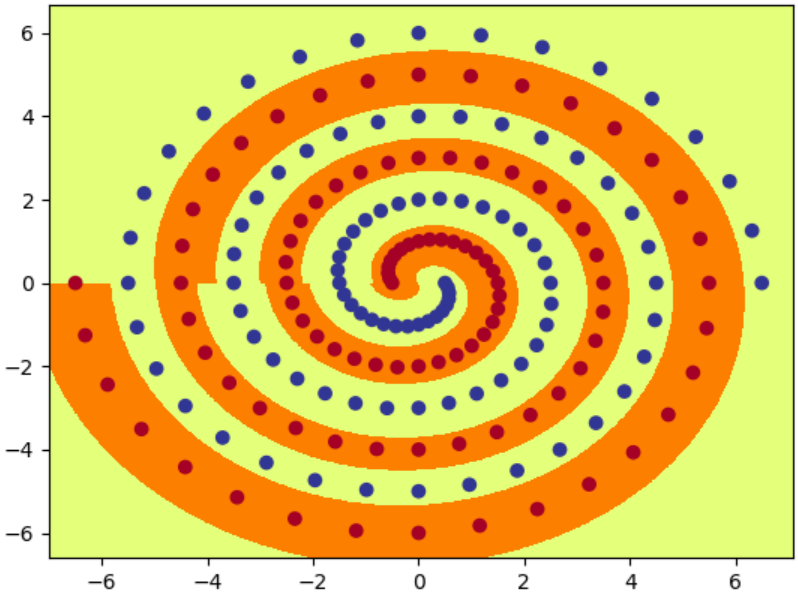
```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.019014
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.007524
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.010398
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.032908
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.014068
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.025620
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.009538
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.043428
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.009697
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.068963
<class 'numpy.ndarray'>
[[933.   4.   1.   1.  31.   4.   0.  18.   7.   1.]
 [  3. 898.  14.   1.  15.   0.  39.   2.  13.  15.]
 [ 11.  11. 877.  29.  13.   8.  15.  14.  13.   9.]
 [  1.   1.  20. 949.   3.   5.   6.   7.   2.   6.]
 [ 33.   9.   8.   6. 904.   2.  13.   6.  11.   8.]
 [  5.   9.  87.   7.   9. 840.  14.   4.  10.  15.]
 [  3.  10.  25.   3.  18.   1. 929.   6.   3.   2.]
 [ 18.   4.   7.   1.   7.   1.  11. 921.   5.  25.]
 [  5.  25.   7.   4.   8.   2.   8.  13. 921.   7.]
 [ 14.   4.  25.   2.  10.   1.   3.  12.  11. 918.]]

Test set: Average loss: 0.3987, Accuracy: 9090/10000 (91%)
```

Therefore, using max_pool layer can perform better in this case.

Part2

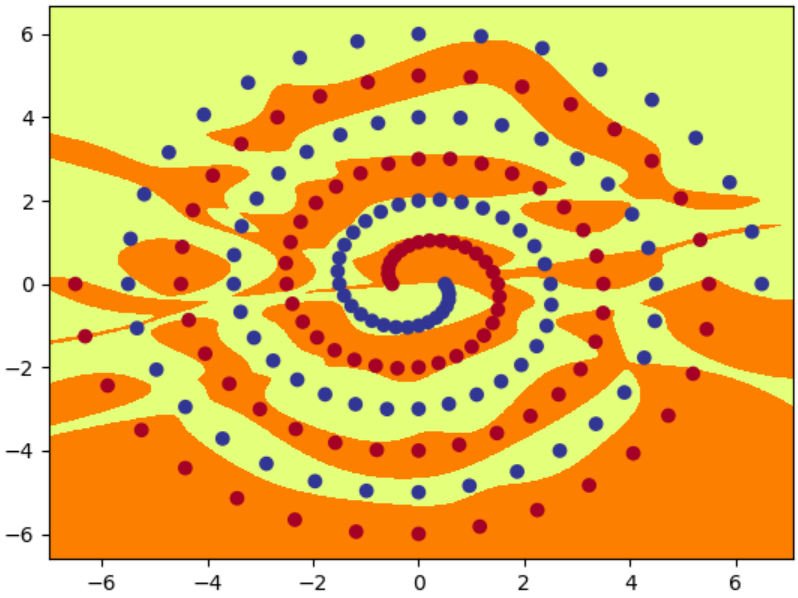
Question 1 && 2

The minimum number of hidden nodes required is **7**. Here is polar_out.png.



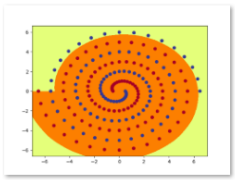
Question 3 && 4

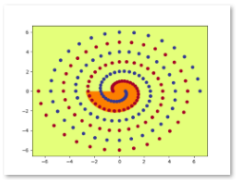
The number of **hidden nodes = 10** and **init_weight = 0.2**. Here is raw_out.png.

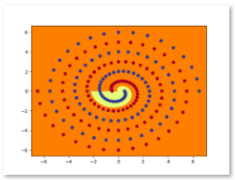


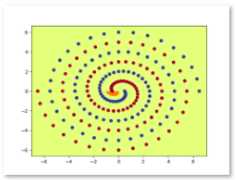
Question 5

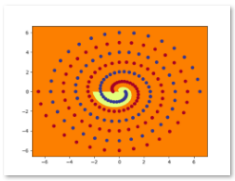
a. Here are plots of all the hidden nodes in PolarNet

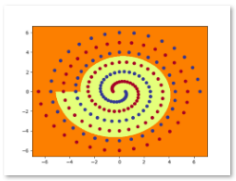

polar1_0

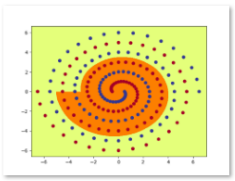

polar1_1


polar1_2

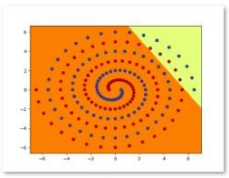

polar1_3

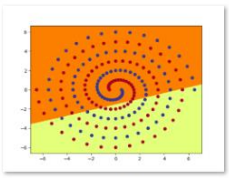

polar1_4

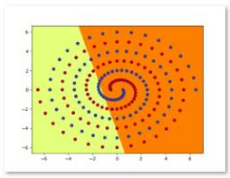

polar1_5

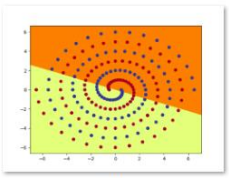

polar1_6

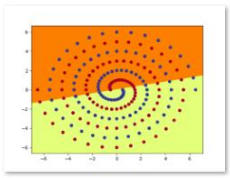
b. Here are plots of all the hidden nodes in both layers of RawNet

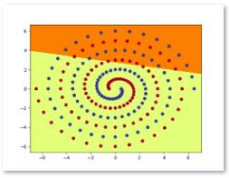

raw1_0

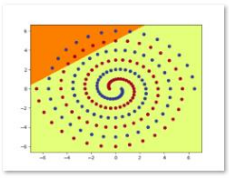

raw1_1

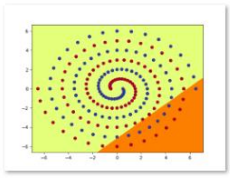

raw1_2

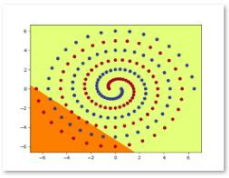

raw1_3

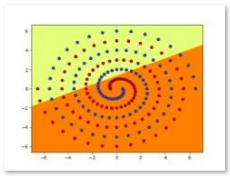

raw1_4

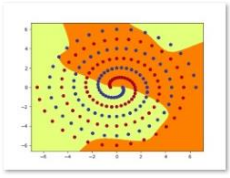

raw1_5

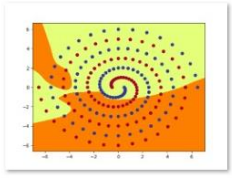

raw1_6

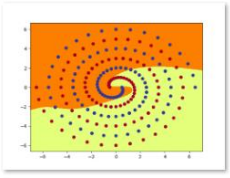

raw1_7

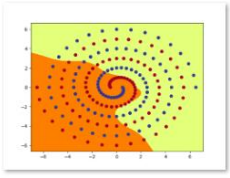

raw1_8

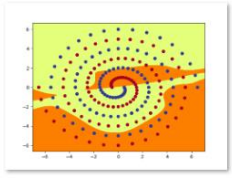

raw1_9

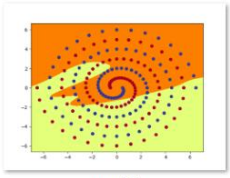

raw2_0

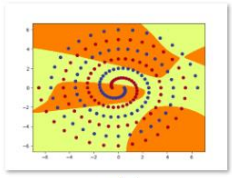

raw2_1

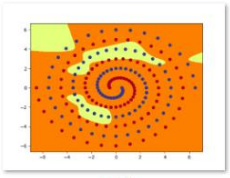

raw2_2

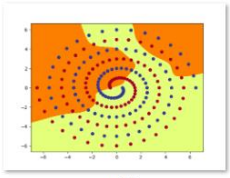

raw2_3

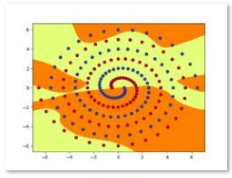

raw2_4


raw2_5


raw2_6


raw2_7


raw2_8


raw2_9

Question 6

a. Plots of all the hidden nodes of model PolarNet look much **smoother** than that of RawNet. Compared with PolarNet, RawNet has two fully connected hidden layers so that the images look **twisted**.

b. After testing the different values for initial weight size given in FAQ, I find that if initial weight is 0.001 or 0.01, the speed and success of learning for RawNet will be quite low. When the value comes to 0.1 or 0.2, Speed and success rate can improvement Significantly. However, if initial weight is 0.3, the rate is lower than before (0.1 and 0.2).

c. Set --hid 10 --init 0.2 as a standard, test 5 times for each model

① changing batch size from 97 to 194

PolarNet

batch_size=97 (Left)

```
ep:12300 loss: 0.0194 acc: 98.97
ep:12400 loss: 0.0199 acc: 98.97
ep:12500 loss: 0.0195 acc: 98.97
ep:12600 loss: 0.0193 acc: 98.97
ep:12700 loss: 0.0192 acc: 98.97
ep:12800 loss: 0.0192 acc: 98.97
ep:12900 loss: 0.0191 acc: 98.97
ep:13000 loss: 0.0190 acc: 98.97
ep:13100 loss: 0.0189 acc: 98.97
ep:13200 loss: 0.0188 acc: 98.97
ep:13300 loss: 0.0187 acc: 98.97
ep:13400 loss: 0.0186 acc: 98.97
ep:13500 loss: 0.0185 acc: 98.97
ep:13600 loss: 0.0185 acc: 98.97
ep:13700 loss: 0.0184 acc: 98.97
ep:13800 loss: 0.0183 acc: 98.97
ep:13900 loss: 0.0183 acc: 98.97
ep:14000 loss: 0.0181 acc: 100.00
```

batch_size=194 (Right)

```
ep: 7900 loss: 0.0365 acc: 98.97
ep: 8000 loss: 0.0364 acc: 98.97
ep: 8100 loss: 0.0362 acc: 98.97
ep: 8200 loss: 0.0362 acc: 98.97
ep: 8300 loss: 0.0359 acc: 98.97
ep: 8400 loss: 0.0358 acc: 98.97
ep: 8500 loss: 0.0357 acc: 98.97
ep: 8600 loss: 0.0356 acc: 98.97
ep: 8700 loss: 0.0354 acc: 98.97
ep: 8800 loss: 0.0354 acc: 98.97
ep: 8900 loss: 0.0353 acc: 98.97
ep: 9000 loss: 0.0353 acc: 98.97
ep: 9100 loss: 0.0350 acc: 98.97
ep: 9200 loss: 0.0349 acc: 98.97
ep: 9300 loss: 0.0350 acc: 98.97
ep: 9400 loss: 0.0347 acc: 98.97
ep: 9500 loss: 0.0347 acc: 98.97
ep: 9600 loss: 0.0345 acc: 98.97
ep: 9700 loss: 0.0346 acc: 99.48
ep: 9800 loss: 0.0344 acc: 98.97
ep: 9900 loss: 0.0346 acc: 100.00
```

RawNet

batch_size=97 (Left)

```
ep: 6000 loss: 0.0452 acc: 96.39
ep: 6100 loss: 0.0449 acc: 96.91
ep: 6200 loss: 0.0434 acc: 96.39
ep: 6300 loss: 0.0410 acc: 97.42
ep: 6400 loss: 0.0387 acc: 97.42
ep: 6500 loss: 0.0369 acc: 97.42
ep: 6600 loss: 0.0355 acc: 97.42
ep: 6700 loss: 0.0345 acc: 97.94
ep: 6800 loss: 0.0337 acc: 97.94
ep: 6900 loss: 0.0378 acc: 97.42
ep: 7000 loss: 0.0325 acc: 97.94
ep: 7100 loss: 0.0321 acc: 97.94
ep: 7200 loss: 0.0318 acc: 97.94
ep: 7300 loss: 0.0315 acc: 97.94
ep: 7400 loss: 0.0309 acc: 97.94
ep: 7500 loss: 0.0302 acc: 97.94
ep: 7600 loss: 0.0293 acc: 98.45
ep: 7700 loss: 0.0284 acc: 99.48
ep: 7800 loss: 0.0274 acc: 100.00
```

batch_size=194 (Right)

```
ep:  100 loss: 0.6735 acc: 61.86
ep:  200 loss: 0.6592 acc: 61.86
ep:  300 loss: 0.6477 acc: 51.55
ep:  400 loss: 0.6360 acc: 58.25
ep:  500 loss: 0.6281 acc: 59.79
ep:  600 loss: 0.6064 acc: 62.37
ep:  700 loss: 0.5635 acc: 64.95
ep:  800 loss: 0.5149 acc: 65.98
ep:  900 loss: 0.4489 acc: 77.84
ep: 1000 loss: 0.3698 acc: 77.84
ep: 1100 loss: 0.2974 acc: 84.54
ep: 1200 loss: 0.2120 acc: 91.24
ep: 1300 loss: 0.1645 acc: 93.30
ep: 1400 loss: 0.1404 acc: 93.81
ep: 1500 loss: 0.1233 acc: 95.88
ep: 1600 loss: 0.1072 acc: 97.42
ep: 1700 loss: 0.0897 acc: 98.45
ep: 1800 loss: 0.0702 acc: 98.97
ep: 1900 loss: 0.0570 acc: 99.48
ep: 2000 loss: 0.0491 acc: 98.97
ep: 2100 loss: 0.0432 acc: 100.00
```

In this case, changing batch size from 97 to 194 can improve the calculate speed.

② changing tanh to relu

PolarNet

Tanh (Left)

```
ep:10000 loss: 0.0155 acc: 96.91
ep:10100 loss: 0.0157 acc: 96.91
ep:10200 loss: 0.0160 acc: 96.91
ep:10300 loss: 0.0163 acc: 97.42
ep:10400 loss: 0.0168 acc: 97.94
ep:10500 loss: 0.0173 acc: 97.94
ep:10600 loss: 0.0177 acc: 97.94
ep:10700 loss: 0.0180 acc: 98.45
ep:10800 loss: 0.0188 acc: 98.97
ep:10900 loss: 0.0191 acc: 98.97
ep:11000 loss: 0.0190 acc: 99.48
ep:11100 loss: 0.0189 acc: 99.48
ep:11200 loss: 0.0184 acc: 99.48
ep:11300 loss: 0.0181 acc: 98.97
ep:11400 loss: 0.0191 acc: 98.97
ep:11500 loss: 0.0196 acc: 99.48
ep:11600 loss: 0.0202 acc: 99.48
ep:11700 loss: 0.0210 acc: 99.48
ep:11800 loss: 0.0216 acc: 99.48
ep:11900 loss: 0.0331 acc: 96.91
ep:12000 loss: 0.0200 acc: 100.00
```

ReLU (Right)

```
ep:27300 loss: 0.1320 acc: 83.51
ep:27400 loss: 0.1316 acc: 80.93
ep:27500 loss: 0.1316 acc: 83.51
ep:27600 loss: 0.1340 acc: 83.51
ep:27700 loss: 0.1320 acc: 82.47
ep:27800 loss: 0.1320 acc: 80.93
ep:27900 loss: 0.1329 acc: 81.44
ep:28000 loss: 0.1299 acc: 80.93
ep:28100 loss: 0.1293 acc: 81.44
ep:28200 loss: 0.1336 acc: 82.99
ep:28300 loss: 0.1320 acc: 82.47
ep:28400 loss: 0.1289 acc: 81.44
ep:28500 loss: 0.1325 acc: 80.93
ep:28600 loss: 0.1332 acc: 82.99
ep:28700 loss: 0.1326 acc: 82.99
ep:28800 loss: 0.1327 acc: 80.93
ep:28900 loss: 0.1290 acc: 81.44
```

RawNet

Tanh (Left)

```
ep: 1800 loss: 0.2117 acc: 93.30
ep: 1900 loss: 0.1913 acc: 94.33
ep: 2000 loss: 0.1626 acc: 96.39
ep: 2100 loss: 0.1458 acc: 96.91
ep: 2200 loss: 0.1348 acc: 96.91
ep: 2300 loss: 0.1264 acc: 96.91
ep: 2400 loss: 0.1197 acc: 96.91
ep: 2500 loss: 0.1143 acc: 96.91
ep: 2600 loss: 0.1095 acc: 96.91
ep: 2700 loss: 0.1031 acc: 97.42
ep: 2800 loss: 0.0958 acc: 97.42
ep: 2900 loss: 0.0857 acc: 97.42
ep: 3000 loss: 0.0780 acc: 97.42
ep: 3100 loss: 0.0728 acc: 97.42
ep: 3200 loss: 0.0630 acc: 98.45
ep: 3300 loss: 0.0595 acc: 98.97
ep: 3400 loss: 0.0552 acc: 98.97
ep: 3500 loss: 0.0655 acc: 92.27
ep: 3600 loss: 0.0483 acc: 99.48
ep: 3700 loss: 0.0451 acc: 99.48
ep: 3800 loss: 0.0414 acc: 99.48
ep: 3900 loss: 0.0364 acc: 99.48
ep: 4000 loss: 0.0313 acc: 100.00
```

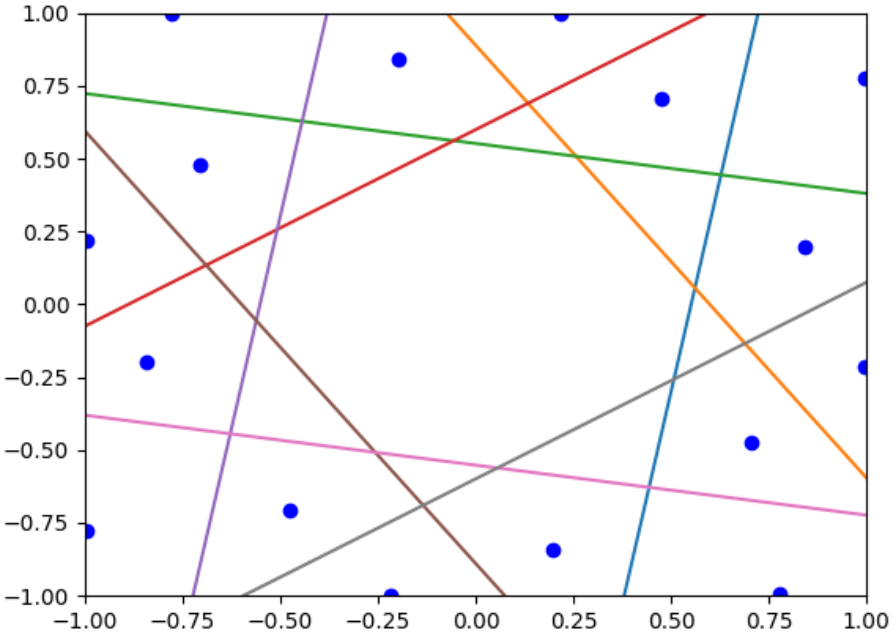
ReLU (Right)

```
ep:56900 loss: 0.2844 acc: 78.35
ep:57000 loss: 0.2811 acc: 77.84
ep:57100 loss: 0.2821 acc: 79.38
ep:57200 loss: 0.2835 acc: 77.84
ep:57300 loss: 0.2804 acc: 80.41
ep:57400 loss: 0.2829 acc: 77.32
ep:57500 loss: 0.2846 acc: 77.84
ep:57600 loss: 0.2821 acc: 77.32
ep:57700 loss: 0.2814 acc: 77.84
ep:57800 loss: 0.2805 acc: 77.84
ep:57900 loss: 0.2847 acc: 77.84
ep:58000 loss: 0.2833 acc: 77.84
ep:58100 loss: 0.2812 acc: 77.84
ep:58200 loss: 0.2812 acc: 78.87
ep:58300 loss: 0.2802 acc: 77.32
ep:58400 loss: 0.2803 acc: 77.32
ep:58500 loss: 0.2835 acc: 77.84
ep:58600 loss: 0.2832 acc: 77.84
ep:58700 loss: 0.2816 acc: 78.35
ep:58800 loss: 0.2802 acc: 79.90
ep:58900 loss: 0.2814 acc: 80.41
ep:59000 loss: 0.2796 acc: 78.35
ep:59100 loss: 0.2831 acc: 79.38
ep:59200 loss: 0.2804 acc: 78.35
```

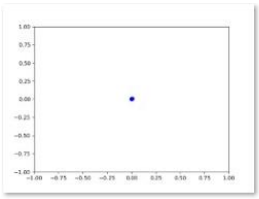
It seems that ReLU function is not suitable for both PolarNet and RawNet. Compared with ReLU, Tanh is much more efficient.

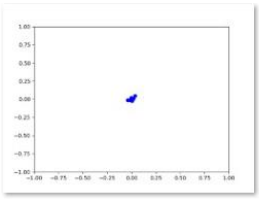
Part3

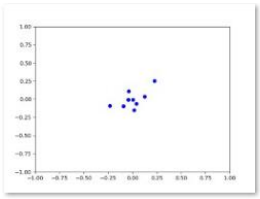
Question 1

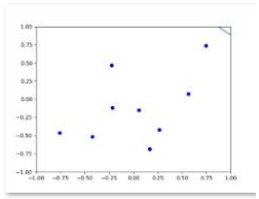


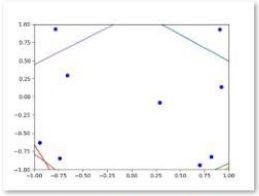
Question 2

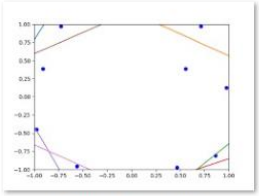

Epochs 50 loss = 0.3501

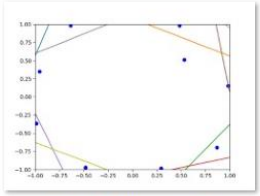

Epochs 100 loss = 0.3488

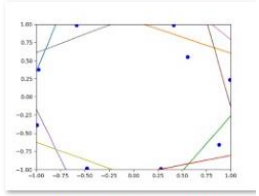

Epochs 150 loss = 0.3459

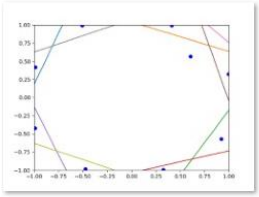

Epochs 200 loss = 0.3010

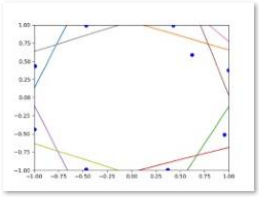

Epochs 300 loss = 0.2040

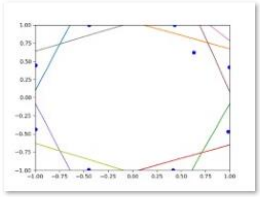

Epochs 500 loss = 0.1686

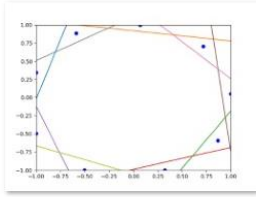

Epochs 700 loss = 0.1477


Epochs 1000 loss = 0.1292


Epochs 1500 loss = 0.1092

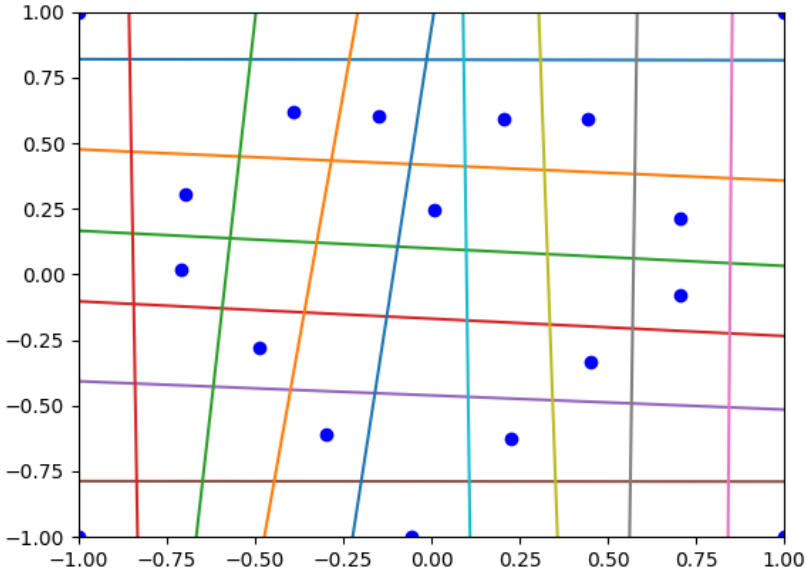

Epochs 2000 loss = 0.0953


Epochs 3000 loss = 0.0776


Epochs 17230 loss = 0.0200

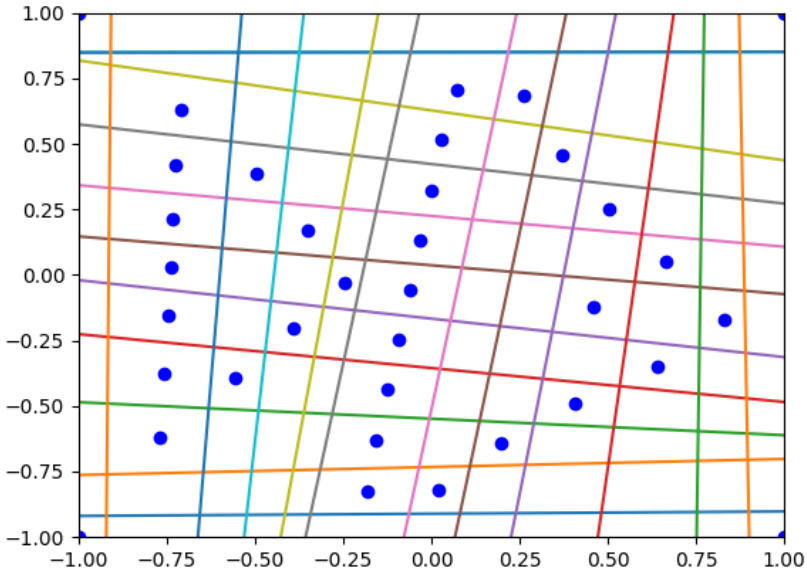
At first, the initial points gather together. As the training progresses, these dots start to spread out. When comes to Epochs 200, output boundaries appear. We can find that lines gradually divide all points so that each area has at most one dot.

Question 3



Question 4

Target1 is a fish



Target2 is a butterfly