Relatório

Trabalho Prático - Redes Neuronais

Licenciaturas em Engenharia Informática
Conhecimento e Raciocínio

Trabalho Realizado por:

->Tiago Quintas, 2019128044, a2019128044@isec.pt

->Francisco Reis, a2019143035@isec.pt

Índice

Introdução 4
Objetivos 5
Transformação de imagens e treino simples de RN –..... 6
alínea a) 6
Testagem de diferentes arquiteturas de RN - alínea b) 8
Testagem, seleção e comparação de RN - alínea c)..... 11
Leitura de imagens - alínea d) 19
GUI do Matlab - alínea e)..... 20
Conclusão..... 21

Introdução

Este relatório tem como base o trabalho prático da cadeira de Conhecimento e Raciocínio lecionada em 2020/2021, do segundo ano de Licenciatura em Engenharia Informática.
O trabalho em si consiste na implementação e teste de diferentes arquiteturas de redes neuronais (RN) feedforward para classificar corretamente caracteres gregos. As RN podem ser usadas para determinar relações e padrões entre entradas e saídas.

Neste caso as nossas entradas serão imagens, que serão posteriormente transformadas, e a nossa saída será uma matriz que nos indica qual o resultado esperado.

A classe feedforward foi a primeira e mais simples RN desenvolvida. Nesta rede, as informações movem-se apenas numa direção - para a frente - dos nós de entrada, através dos nós escondidos (se houver) para os nós de saída. Apesar de existirem outras classes mais adaptadas à leitura de imagens, por requisito do enunciado apenas iremos usar esta classe.

Todos os resultados aqui apresentados foram obtidos com recurso ao software Matlab versão R2020b, com os add-ons Deep Learning Toolbox e Image Processing Toolbox instalados.

Objetivos

O objetivo deste trabalho prático consiste na implementação e teste de diferentes arquiteturas de redes neuronais (RN) feedforward para classificar corretamente 6 figuras geométricas:

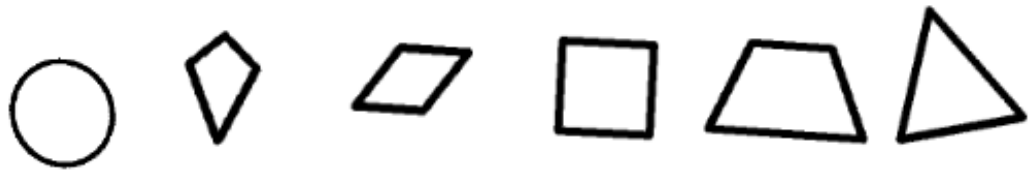


FIGURA 1: CARACTERES GREGOS USADOS PARA CLASSIFICAÇÃO

Estes caracteres têm a seguinte designação (por ordem): círculo, papagaio, paralelograma, quadrado, trapézio e triângulo.

É-nos fornecido 3 pastas com várias imagens destes caracteres, e de acordo as várias alíneas (enunciado em anexo) sabemos de que maneira as vamos usar para treinar diferentes RN. Estas alíneas consistem em:

a) Primeiro contacto e manipulação das imagens (transformação de imagens jpg para matrizes binárias) e treino de uma RN simples (topologia default e/ou usando diferentes funções de treino) usando imagens da pasta **start**.
b) Readaptar o código usado na alínea a) para agora treinar várias RN, com diferentes parametrizações, usando a pasta **train**. Também devem ser gravadas as melhores RN obtidas nesta alínea.
c) Testar as melhores RN obtidas na alínea b) com as imagens da pasta **test**. Treinar a RN usando as melhores parametrizações da alínea b) para a pasta **test**, e testar a melhor RN obtida em cada pasta. Por fim, treinar a RN com todas as imagens fornecidas (pasta **start** + **test** + **train**) e testar a melhor RN obtida para cada pasta.
d) Desenvolver uma aplicação consola simples para ler imagens desenhadas por nós para serem classificadas com recurso à melhor RN obtida na alínea c).
e) Desenvolver uma aplicação gráfica que use todas as tarefas exploradas pelas alíneas anteriores.

Neste relatório é explicado como foi resolvido cada alínea e discutido os seus resultados.
Estas alíneas estão todas por ordem e escritas nos títulos deste relatório para uma

maior facilidade de navegação dentro do mesmo. Cada RN, código, enunciado, imagens e outros recursos usados/obtidos estão anexados a este relatório.

Transformação de imagens e treino simples de RN – alínea a)

Para esta primeira alínea a meta é simples e direta – transformar as imagens da pasta **start** e usá-las para treinar uma RN sem grandes alterações de parâmetros.

Começando pela transformação das imagens, foi usado a função imageDatastore para carregar as imagens, o imbinarize (binarizar a imagem), e por fim a verticalização da matriz da imagem binarizada para esta ocupar só uma coluna e juntar à nossa main_matrix que no final irá conter as 5 imagens carregadas.

Em relação à nossa matriz target, uma matriz 6x30, optámos por diferenciar cada figura pela posição do bit 1. Por exemplo, a letra círculo tem o bit na primeira posição, a figura papagaio tem o bit na segunda posição, etc. Tudo com a mesma ordem da figura 1. Criada da seguinte maneira:

```
Saida desejada:
     1     1     1     1     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     1     1     1     1     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     1     1     1     1     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     1     1     1     1     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     1     1     1     1     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     1     1     1     1
```

Figura 2 e 3: Matriz target

Foram também usados todos os exemplos no treino (net.divideFcn='';).

Após este primeiro setup, foi iniciada a primeira tentativa de treino da RN, que foi um insucesso: a nossa main_matrix era excessivamente grande para o Matlab a processar usando a função de treino default (trainlm), originando um erro. Esta função apenas funcionavam com uma redução de 80% da resolução no entanto o tempo de treino era demasiado elevado e não conseguimos obter resultados.

Foram então testadas outras funções de treino para esta alínea, tais como:

- `trainscg`
- `traincgf`
- `traincgp`
- `trainoss`

- `traingdx`

Todas estas funções de treino funcionaram com a nossa main_matrix com as imagens com a resolução reduzida a 50% (112x112 pixeis) sendo que reparamos que com as imagens com uma resolução maior tinham um tempo de treino mais elevado e a precisão total era significativamente mais baixa.

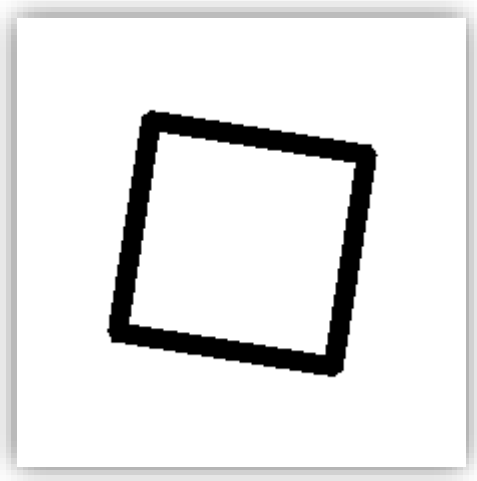


Figura 4: Imagem com resolução original (224x224 pixeis)

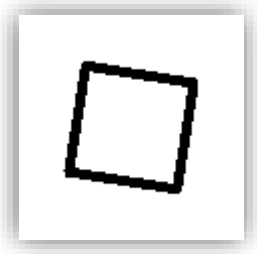


Figura 5: Imagem reduzida a 50% (112x112pixeis)

Optamos assim por usar esta resolução das imagens conseguindo assim obter uma precisão de 100% tal como era desejado, usando a função de treino `traincgp` Optamos assim por usar esta resolução das imagens conseguindo assim obter uma precisão de 100% tal como era desejado, usando a função de treino `traincgp` (Conjugate gradient backpropagation with Powell-Beale restarts).

As funções de treino com melhores resultados foram `trainscg` com uma precisão a variar entre 80% e 100% e a função `traincgp` com uma precisão a variar entre 90% e 100% sendo que foi a com melhores resultados e a que obteve uma precisão de 100% com mais frequência.

As funções de treino `traincgf` obteve resultados entre 45% e 80% com uma media de 60%, a função `traingdx` obteve resultados entre 10% e 23% com uma media de 14% e a função de treino `trainoss` foi a que obteve os piores resultados com uma precisão que variava entre 3% e 8%.

Testagem de diferentes arquiteturas de RN - alínea b)

Para a alínea b) o objetivo final é direto – transformar as imagens da pasta **train** e usá-las para treinar uma RN sem grandes alterações de parâmetros, tais como:

- Número e dimensão de camadas escondidas
- Função de treino
- Função de ativação
- Divisão dos exemplos pelo conjunto

As funções de ativação default assumem-se sempre como tansig (camadas escondidas) e radbasn (camada de saída).

Em relação ao número e dimensão de camadas escondidas, é assumido por default 50 neurónios (1 camada escondida)

Quanto à função de treino, definiu-se como default a função trainscg dado a obtenção de melhores resultados em relação às restantes.

Começamos então por treinar as várias RN começando por usar a configuração default, e gradualmente fomos alterando vários parâmetros. Obtivemos os seguintes resultados:

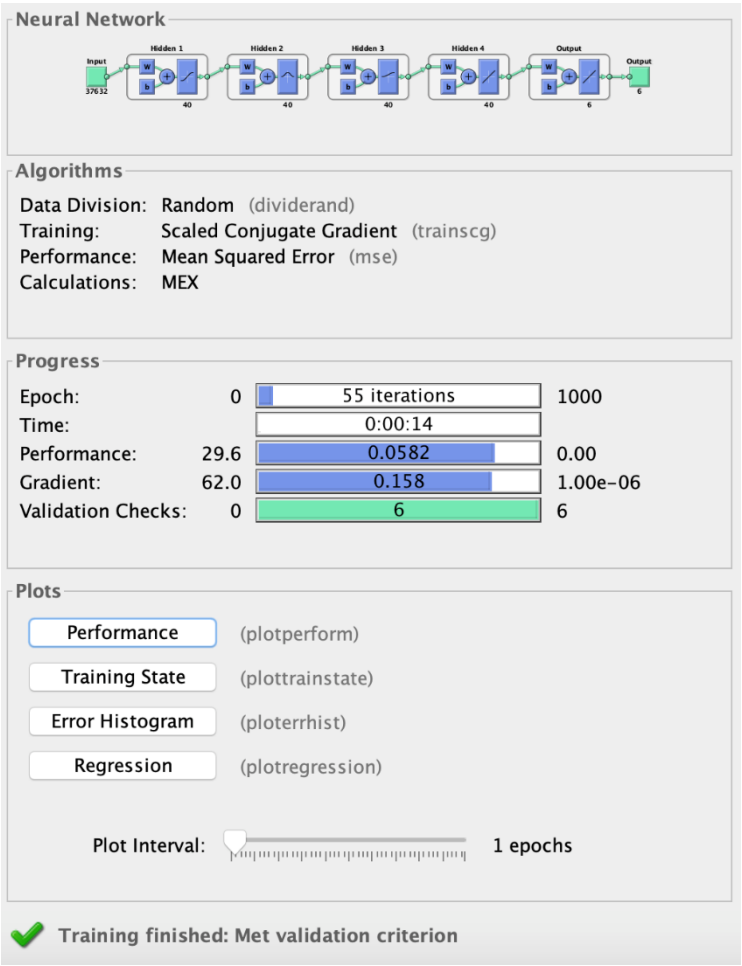
- Com a configuração default obtivemos uma precisão global de 91.60%. Esta configuração não conseguiu chegar aos 100% como na alínea a), logo concluímos que o número de imagens juntamente com uma matriz target maior influencia bastantes os resultados, logo teriam de ser testados vários parâmetros de treino da RN para obter os 100%.
- Em relação à alteração do número de camadas e a dimensão das camadas escondidas verificámos que à medida que se aumenta o número de camadas escondidas e de neurónios, a precisão global e de testes aumenta progressivamente também, concluindo que a mesma influencia os resultados finais.

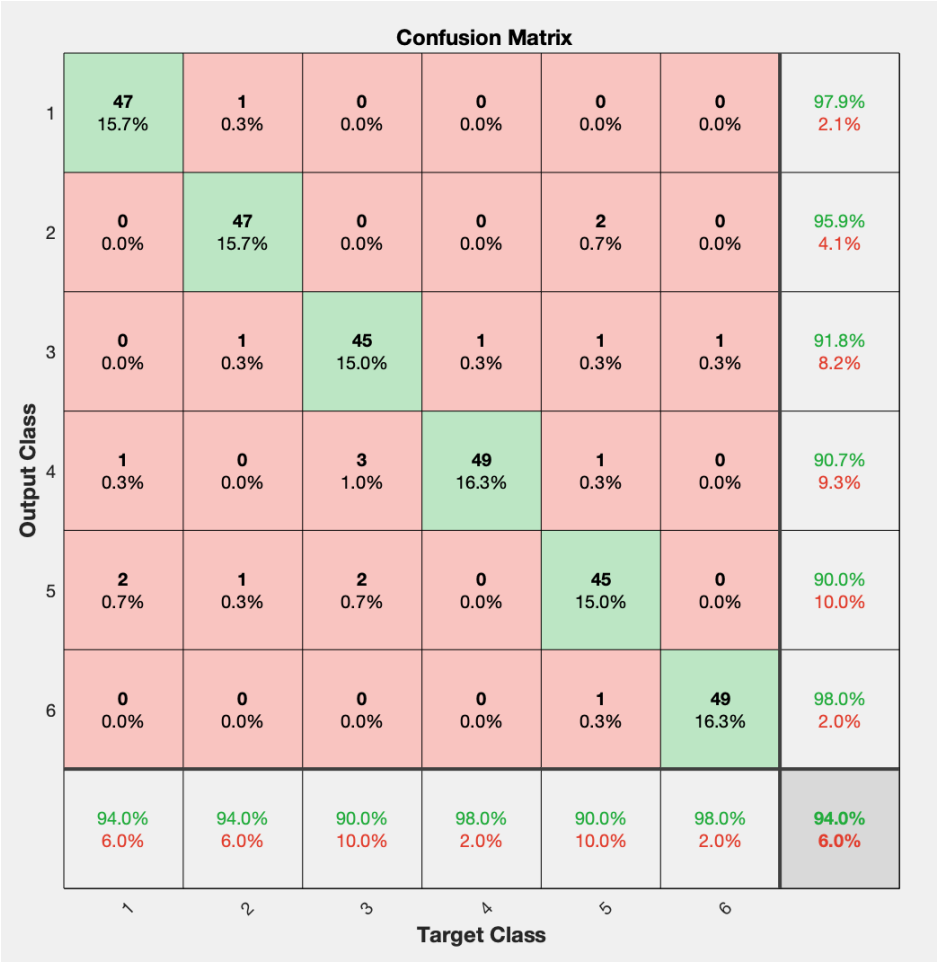
- A alteração às funções de treino, deu-nos a perceber que existem funções que produzem melhores resultados em termos de precisão global e de teste como é o caso das funções traingdx (89.80%) e traincgb(88.40%). No sentido inverso verifica-se que por exemplo a função traingdm (72.60%) foi a que produziu resultados menos positivos em termos de precisão ao longo dos testes.

- As funções de ativação através da combinação de forma aleatória, revelou que a alteração das mesmas provoca um grande impacto na precisão da rede neuronal, como foram exemplos os casos da combinação (netinv, radbasn) que produziram resultados de 17.20% ou a junção das funções de ativação (netinv, purelin) que teve uma precisão de 17.40%.

- Nas divisões dos conjuntos, obtivemos maiores precisões de teste quando diminuímos o tamanho do conjunto usado para teste.

Por fim, gravámos a melhor rede obtida nesta alínea, com uma precisão global de 94% e de 91% em relação aos testes. A topologia usada foram 4 camadas escondidas com 40 neurónios cada,função de treino trainscg, funções de ativação tansig, radbasn, logsig, purelin, purelin e divisão de exemplos default.




Confusion Matrix
Output Class
Target Class
1
2
3
4
5
6
47 15.7%
1 0.3%
0 0.0%
0 0.0%
0 0.0%
0 0.0%
97.9% 2.1%
0 0.0%
47 15.7%
0 0.0%
0 0.0%
2 0.7%
0 0.0%
95.9% 4.1%
0 0.0%
1 0.3%
45 15.0%
1 0.3%
1 0.3%
1 0.3%
91.8% 8.2%
1 0.3%
0 0.0%
3 1.0%
49 16.3%
1 0.3%
0 0.0%
90.7% 9.3%
2 0.7%
1 0.3%
2 0.7%
0 0.0%
45 15.0%
0 0.0%
90.0% 10.0%
0 0.0%
0 0.0%
0 0.0%
0 0.0%
1 0.3%
49 16.3%
98.0% 2.0%
94.0% 6.0%
94.0% 6.0%
90.0% 10.0%
98.0% 2.0%
90.0% 10.0%
98.0% 2.0%
94.0% 6.0%

Testagem, seleção e comparação de RN - alínea c)

Esta alínea vamos proceder à testagem de RN, à seleção das melhores RN e comparar resultados usando exemplos da pasta test. Estes tópicos foram divididos da seguinte forma:

1. Verificar se a classificação das duas melhores RN obtidas na alínea b) é a correta quando aplicada à pasta **test**.

2. Treinar a RN para os exemplos da pasta **test**, e testar as melhores RN obtidas com a pasta **start**, **test**, **train** separadamente.

3. Voltar a treinar a RN, mas desta vez para o conjunto de imagens das 3 pastas, pasta **start**, **test**, **train** (390 exemplos) e testar as melhores RN para cada pasta, separadamente.

Começando pelo **ponto 1**, testamos as nossas duas melhores redes descritas na alínea b), chegamos aos seguintes resultados:

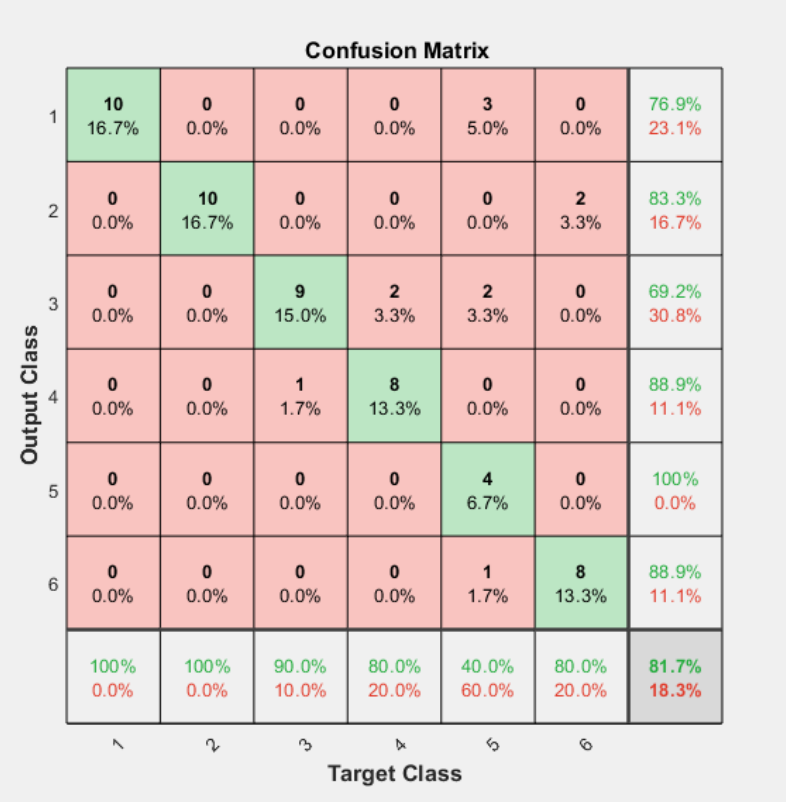


FIGURA ?: MATRIZ DE CONFUSÃO DA MELHOR RN OBTIDA NA ALÍNEA B), APLICADA À PASTA **TEST**

Como podemos observar, houve um decréscimo na precisão global da nossa melhor RN obtida na alínea b) em cerca de 10%. De recordar que esta rede obteve 91.6% de precisão global quando treinada para as imagens da pasta **train**, o que nos diz que algumas das imagens providenciadas pela pasta **test** estão desenhadas de maneira diferente.

- As figuras circle e kite obtiveram uma precisão de 100%
- A figura parallelogram obteve uma precisão de 90%
- As figuras square e triangle obtiveram uma precisão de 80%.
- a figura trapezoid, obteve uma precisão de 40%

Em especial a figura trapezoid, que neste caso a nossa RN só acertou 4 em 10 (precisão de 40%).

Agora em relação ao **ponto 2** (treinar a rede para os exemplos da pasta **test**), voltamos a pegar nos mesmos parâmetros das nossas duas melhores RN, ou seja, voltamos a treinar com os seguintes parâmetros:

- 1 camada escondida com 50 neurónios, função de treino trainscg com uma e funções de ativação tansig e radbasn divisão dos exemplos dividerand = {0.7, 0.15, 0.15}**.**

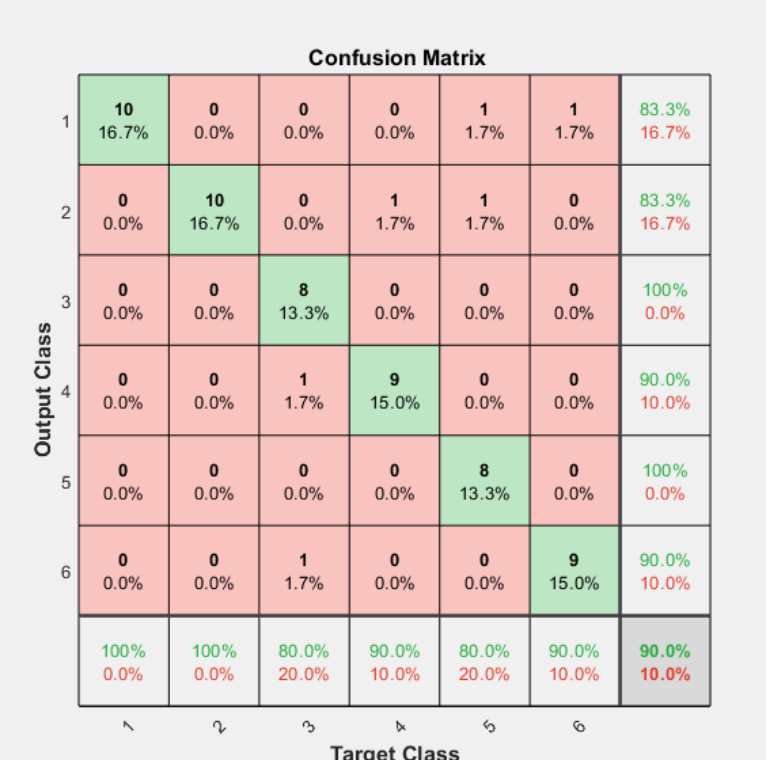


FIGURA ?: MATRIZ DE CONFUSÃO DA MELHOR RN OBTIDA COM EXEMPLOS DA PASTA **TEST**

Em que obtemos uma precisão global de 90% e uma precisão de teste a 66.7%.

Procedemos então à testagem destas redes para a pasta **start**, **test**, **train** separadamente. Os resultados obtidos foram estes:

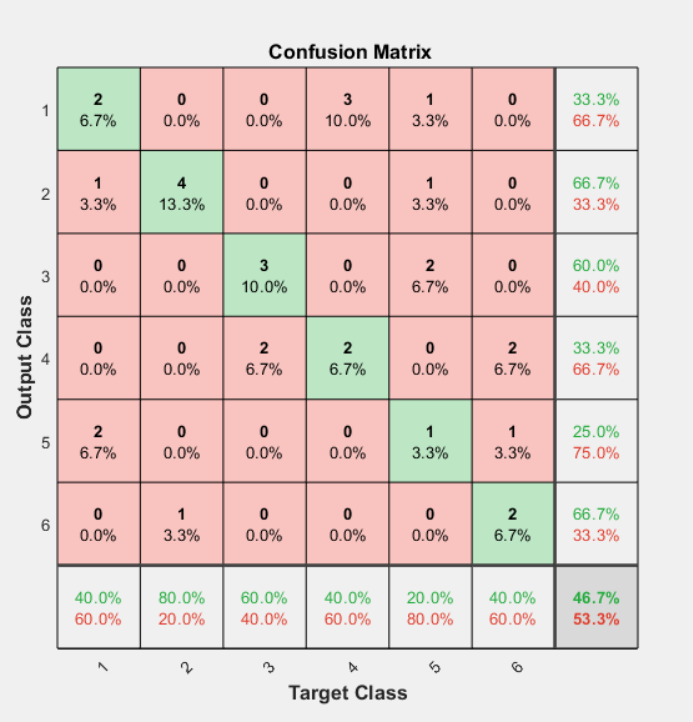


FIGURA ?: MATRIZ DE CONFUSÃO DA MELHOR RN OBTIDA NA ALÍNEA C) PONTO 2 COM EXEMPLOS DA PASTA **START**

Obtivemos uma precisão global de 46%, ou seja, 4.6 em cada 10 imagens foram classificadas corretamente.

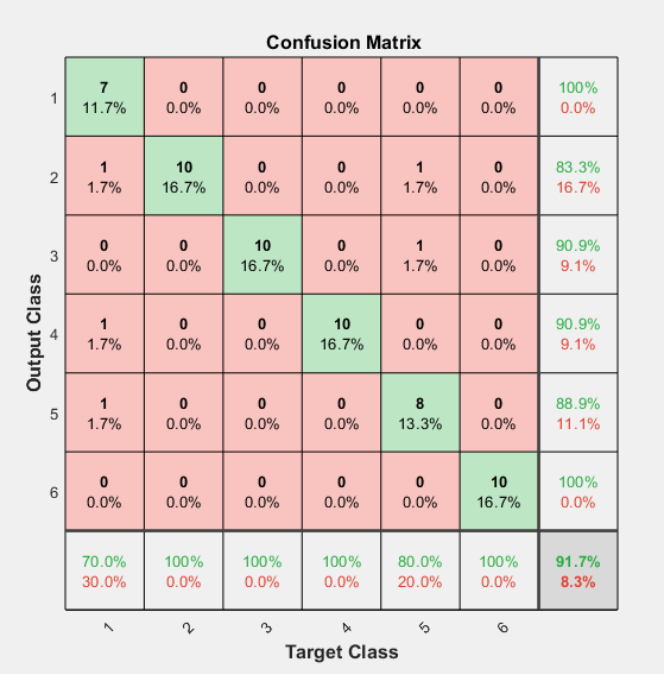


FIGURA ?: MATRIZ DE CONFUSÃO DA MELHOR RN OBTIDA NA ALÍNEA C) PONTO 2 COM EXEMPLOS DA PASTA **TEST**

Em que obtemos uma precisão global de 90% e uma precisão de teste a 66.7%.

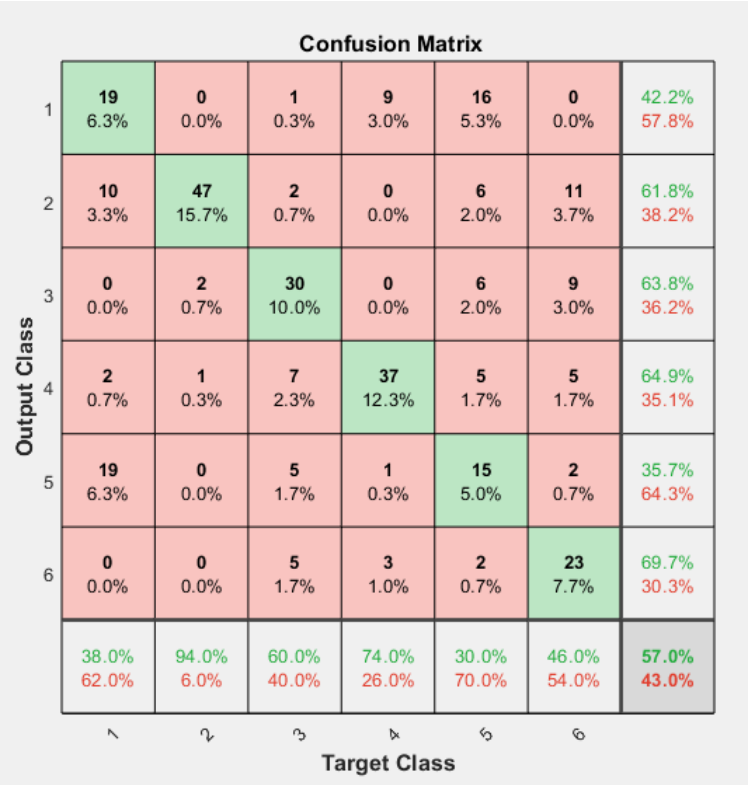


FIGURA ?: MATRIZ DE CONFUSÃO DA MELHOR RN OBTIDA NA ALÍNEA C) PONTO 2 COM EXEMPLOS DA PASTA **TRAIN**

Obtivemos uma precisão global de 57%, ou seja, 5.7 em cada 10 imagens foram classificadas corretamente.

Concluímos que cada RN tem resultados melhores quando treinada para determinada pasta, quando testada para outra há tendência para existir um decréscimo (entre 40% e 50%) da precisão global. Sendo que a pasta start é a que apresenta uma pior precisão global de 46%. Em relação à testagem para a pasta 3, os resultados da matriz de confusão são iguais aos esperados, ou seja 91%.

Por fim no ponto 3 vamos pegar em todas as imagens fornecidas das três pastas (390 imagens) e voltar a testar em separado para as mesmas. Novamente, como no ponto 2, vamos treinar duas redes com os mesmos parâmetros :

- 1 camada escondida com 50 neurónios, função de treino trainscg com uma e funções de ativação tansig e radbasn divisão dos exemplos dividerand = {0.7, 0.15, 0.15}.

Obtivemos a primeira com 83% de precisão global e 64% de precisão de teste, e a segunda com 98% de precisão global 91% e de precisão de teste.

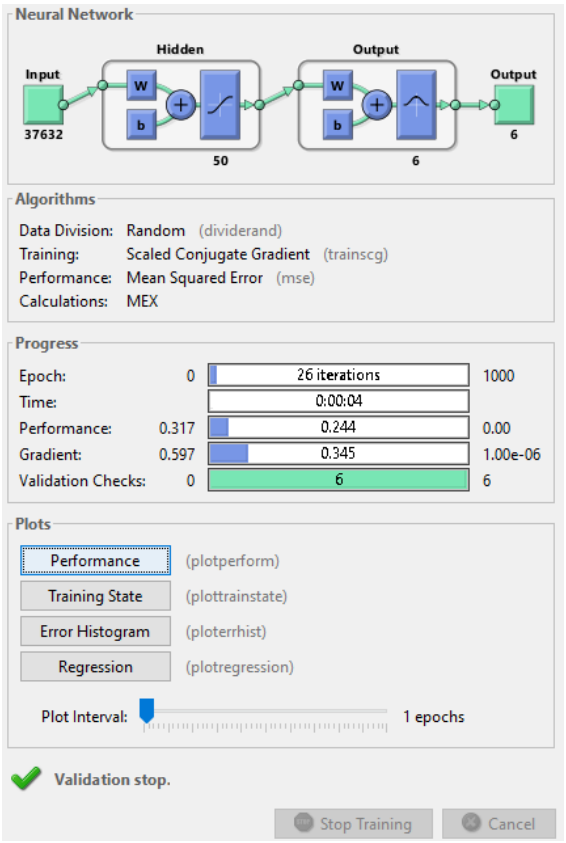


FIGURA ?: RESULTADO DO TREINO DA MELHOR RN OBTIDA NA ALÍNEA C) PONTO 3

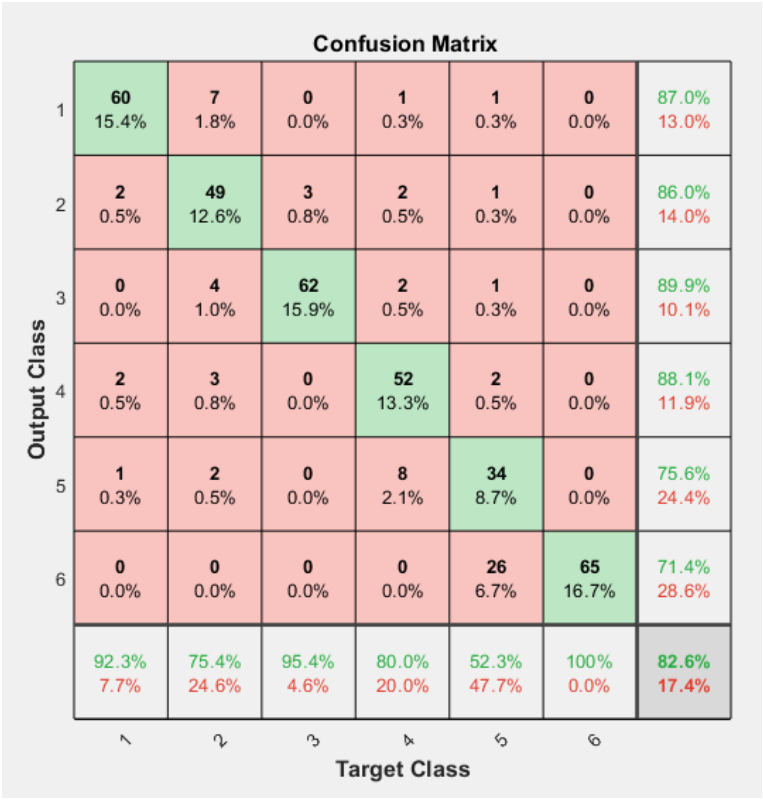


FIGURA ?: MATRIZ DE CONFUSÃO DA MELHOR RN OBTIDA NA ALÍNEA C) PONTO 3

Como podemos observar, esta RN falhou na classificação de várias figuras e a precisão global e de teste não era tão elevada como desejado.

As figuras com mais erros são o trapezoid novamente tal como foi observado no ponto 1 com uma precisão apenas de 56%, o kite com uma precisão de 75%.

Com estes resultados podemos concluir que nas o treino e teste na rede nas várias pastas a figura trapazoid tende a ter uma precisão mais baixa que o esperado.

Volto a relembrar que os gráficos de performance, estado de treino, histograma de erros e regressão destas RN estão também anexados a este relatório, juntamente com as redes guardadas.

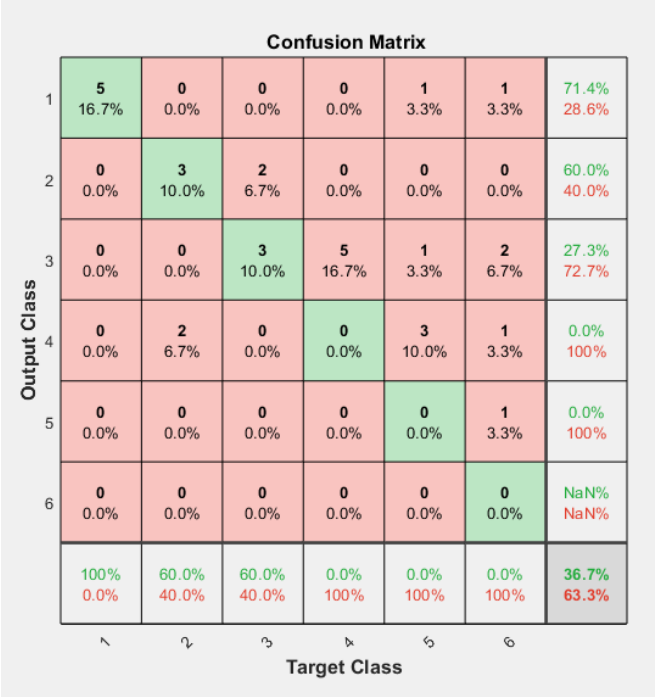


FIGURA ?: MATRIZ DE CONFUSÃO DA MELHOR RN OBTIDA NA ALÍNEA C) PONTO 3 COM EXEMPLOS DA PASTA **START**

Esta RN com a pasta **start** apenas obteve uma precisão de 36.7 sendo que as figuras square trapezoid e triangle nunca foram reconhecidas pela rede.

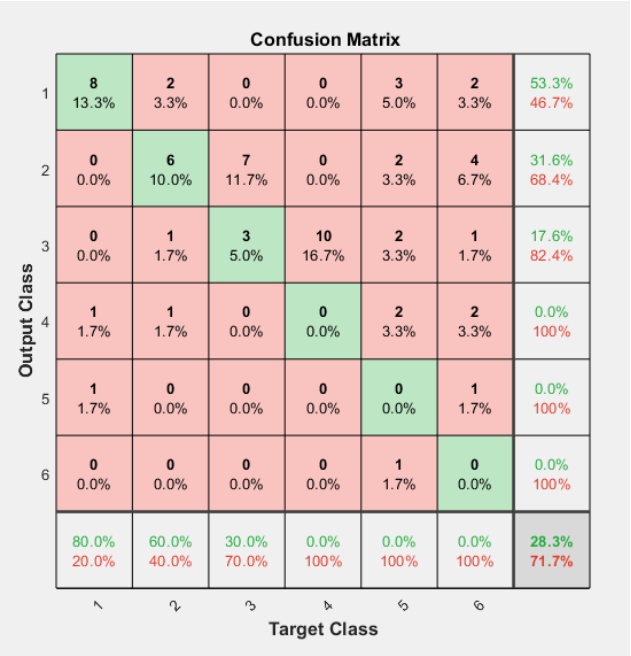


FIGURA ?: MATRIZ DE CONFUSÃO DA MELHOR RN OBTIDA NA ALÍNEA C) PONTO 3 COM EXEMPLOS DA PASTA **TEST**

Esta RN com a pasta **test** apenas obteve uma precisão de 28 sendo que as figuras square trapezoid e triangle nunca foram reconhecidas pela rede.

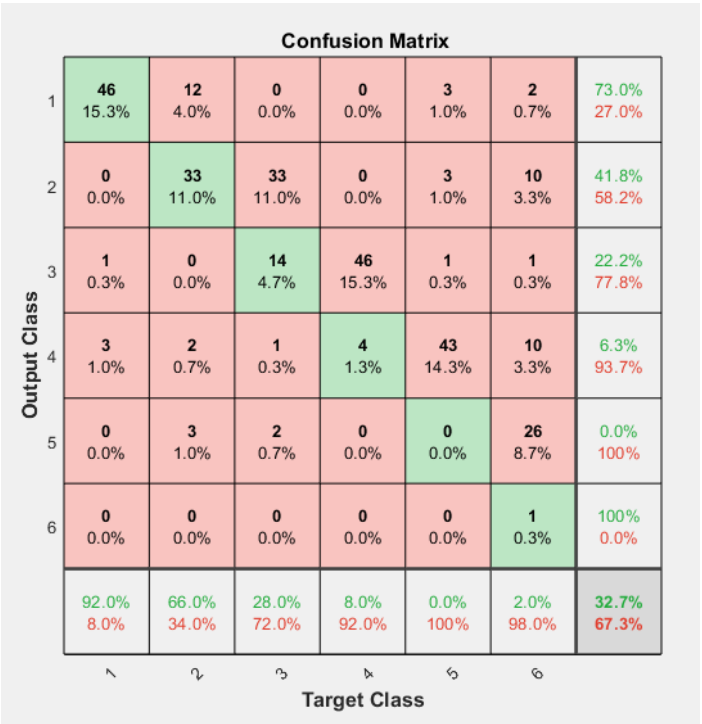


FIGURA ?: MATRIZ DE CONFUSÃO DA MELHOR RN OBTIDA NA ALÍNEA C) PONTO 3 COM EXEMPLOS DA PASTA **TRAIN**

Esta RN com a pasta **train** apenas obteve uma precisão de 33% sendo que a figura trapezoid nunca foi reconhecidas pela rede, a figura triangle apenas foi reconhecida uma vez e a figura square apenas 4.

Podemos então concluir que a precisão de teste que obtivemos no ponto 3 não coincide com os resultados obtidos nas diversas pastas sendo que ao testar nas pastas individuais a precisão de teste diminui cerca de metade da original algo bastante significativo.

Leitura de imagens - alínea d)

Para esta alínea, foi desenvolvido um pequeno programa para ler um ficheiro do tipo imagem, com uma figura desenhada e usar a melhor RN obtida na alínea c). Neste caso vamos usar a RN com a seguinte configuração:

- 1 camada escondida com 50 neurónios, função de treino trainscg com uma e funções de ativação tansig e radbasn divisão dos exemplos dividerand = {0.7, 0.15, 0.15}**.**

Vamos pegar numa imagem desenhada à mão no paint:

Todas estas imagens foram reconhecidas exceto o trapazoid sendo que foi uma figura que nunca conseguimos com que fosse reconhecida.

Reparamos que se as figuras de modo a serem reconhecidas tem de ser desenhadas de forma bastante semelhante ao dataset, e tem de se encontrar numa posição bastante idêntica ao mesmo.

GUI do Matlab - alínea e)

Nesta última alínea, é-nos proposto desenvolver uma GUI (graphical user interface) em

Matlab que permita ao utilizador fazer as seguintes tarefas:

• Configurar a topologia da rede neuronal.

• Escolher funções de treino / ativação.

• Treinar a rede neuronal.

• Gravar uma rede neuronal previamente treinada.

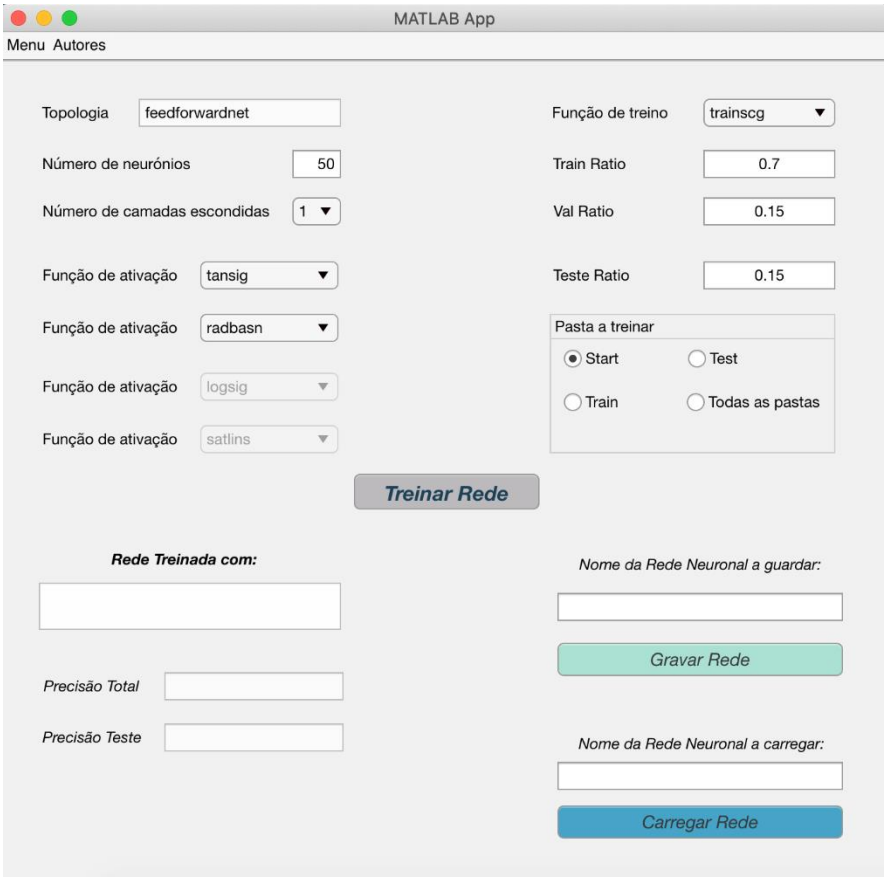
• Carregar uma rede neuronal previamente treinada e aplicá-la a um dataset.

• Desenhar uma nova figura, ou carregar um ficheiro de imagem onde esta já se encontre desenhada. Aplicar um a rede neuronal para classificar a figura desenhada.

• Visualizar os resultados da classificação.

• Geração/gravação de ficheiros de resultados se achar relevante e necessário.

A GUI foi então construída com base nestes pontos, e o resultado final é este:



Conclusão

Com este trabalho prático foi possível pôr em prática a matéria lecionada sobre Redes Neuronais (RN) pela cadeira de Conhecimento e Raciocínio. Implementamos com sucesso a todas as alíneas requeridas pelo enunciado.

Em relação ao trabalho em si, concluímos que quanto maior for o número de exemplos usados para treinar uma rede, melhores resultados se obtêm na classificação de figuras geométricas, mas aumentamos o tempo de computação. Também concluímos que as performances das RN aumentam quanto maior for o número de camadas e o número de neurónios.

Este trabalho foi bastante útil para aumentar os nossos conhecimentos de RN e de aprender a manipular estas redes através do software Matlab.