Politécnico de Coimbra
Instituto Superior de Engenharia
Politécnico de Coimbra

Relatório

Conhecimento e Raciocínio

Trabalho Prático - Redes Neuronais

Trabalho realizado por:

Gonçalo Ramalho – 2019106561

Rafael Correia – 2019131435

Índice

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 10 caracteres gregos:

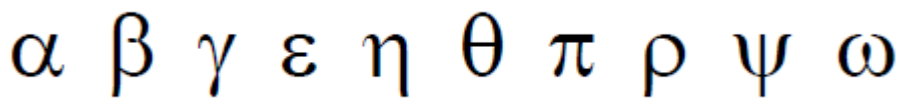

Figura 1: Caracteres gregos usados para classificação

Estes caracteres têm a seguinte designação (por ordem): Alfa, Beta, Gama, Épsilon, Eta, Teta, Pi, Ró, Psi e Ômega.

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 1.

b) Readaptar o código usado na alínea a) para agora treinar várias RN, com diferentes parametrizações, usando a pasta 2. Também devem ser gravadas as melhores RN obtidas nesta alínea.

c) Testar as melhores RN obtidas na alínea b) com as imagens da pasta 3. Treinar a RN usando as melhores parametrizações da alínea b) para a pasta 3, e testar a melhor RN obtida em cada pasta. Por fim, treinar a RN com todas as imagens fornecidas (pasta 1 + 2 + 3) 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 1 e usá-las para treinar uma RN sem grandes alterações de parâmetros.

Começando pela transformação das imagens, foi usado um ciclo simples para carregar cada imagem, depois o resize (redimensionar), 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 10 imagens carregadas. Devido ao tamanho enorme das imagens fornecidas, foi optado inicialmente por redimensionar as imagens em 90%.

Em relação à nossa matriz target, optámos por diferenciar cada letra pela posição do bit 1. Por exemplo, a letra alfa tem o bit na primeira posição, a letra beta tem o bit na segunda posição, etc. Tudo com a mesma ordem da figura 1.

```
target = [1 0 0 0 0 0 0 0 0 0;
          0 1 0 0 0 0 0 0 0 0;
          0 0 1 0 0 0 0 0 0 0;
          0 0 0 1 0 0 0 0 0 0;
          0 0 0 0 1 0 0 0 0 0;
          0 0 0 0 0 1 0 0 0 0;
          0 0 0 0 0 0 1 0 0 0;
          0 0 0 0 0 0 0 1 0 0;
          0 0 0 0 0 0 0 0 1 0;
          0 0 0 0 0 0 0 0 0 1];
```

Figura 2: Matriz target da pasta 1

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

- trainc
- traincgb
- trainrp
- traingdx
- trains

Todas estas funções de treino funcionaram com a nossa main_matrix, mas continuávamos a ter outro problema, o facto de não conseguirmos bons resultados na precisão total (percentagem de imagens corretamente classificadas) e também o tempo necessário para treinar era considerável

(não era propriamente rápido). Esta precisão variava entre 30 e 70%, o que do nosso ponto de vista não era satisfatório, pois o nosso objetivo era de obter uma RN com 100% de precisão total.

Após alguma pesquisa acerca do que poderia estar a acontecer para os resultados serem tão baixos e demorados, resolvemos focar não nas funções de treino, mas sim na transformação das imagens. Notamos que uma redução de 90% era ainda muito pesada para o treino de uma RN. Optámos então por reduzir ainda mais as imagens, usando uma redução de 96.5% (imresize(image,0.035);), ficando assim com uma imagem 106 por 106 pixéis.

Figura 3: Exemplo de uma imagem com 106 por 106 pixéis

Ao usar este redimensionamento, os resultados obtidos foram plenamente satisfatórios: não só o treino era rápido como os resultados da precisão total nunca eram abaixo de 80%. As funções de treino que obtivemos melhores resultados (100%) foram a **traincgb** *(Conjugate gradient backpropagation with Powell-Beale restarts*) e a **trains** (*Sequential order incremental training with learning functions*). Estas duas funções serão então usadas para futuros testes nas alíneas seguintes.

A partir daqui todas as alíneas seguintes seguem o mesmo padrão de redimensionamento (96.5%) e o mesmo padrão da matriz target (letras alfa têm o bit na primeira posição, as letras beta têm o bit na segunda posição, etc).

Testagem de diferentes arquiteturas de RN - alínea b)

Nesta alínea usamos agora as imagens da pasta 2, que contém 100 exemplos (10 para cada letra), é feito o treino de várias RN e verificado o desempenho usando vários parâmetros, tais como:

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

Nesta alínea como já não usamos todos exemplos para o treino, estes vão ser divididos em 3 conjuntos: treino, validação e teste. O primeiro conjunto é utilizado durante a aplicação do algoritmo de treino supervisionado. O segundo conjunto é útil para detetar quando deve terminar o treino (i.e., ajuda a detetar quando a rede começa a perder capacidade de generalização). O terceiro não é utilizado durante o processo de treino e serve normalmente para fazer comparações entre diferentes estratégias de classificação. Os dois mais importantes analisados aqui e nas seguintes alíneas continua a ser a precisão de treino e agora temos também a precisão de teste. O default da divisão dos exemplos é de 70%, 15% e 15%, respetivamente.

Em relação ao número e dimensão de camadas escondidas, é assumido por default 10 neurónios (1 camada escondida).

Quanto às funções de ativação, estas por default assumem-se sempre como tansig (camadas escondidas) e purelin (camada de saída).

Como explicado na alínea anterior, não será usada neste trabalho a função de treino default (trainlm), mas sim a função traincbg que foi das que obteve melhores resultados. Será então a nossa função de treino default.

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

1. Com a configuração default obtivemos uma precisão global de 47.8%. Esta configuração não conseguiu chegar aos 100% como na alínea a), logo concluímos que o número de imagens juntamente com uma matriz target maior influência bastantes os resultados, logo teriam de ser testados vários parâmetros de treino da RN para obter os 100%.
2. Em relação à alteração do número de camadas, fomos aumentando progressivamente tanto o número de camadas como o número de neurónios de cada camada. Obtivemos várias RN com precisão global a 100% usando 6 camadas escondidas com 50 neurónios cada.
3. Quanto à função de treino, testamos a trains e a traingdx. A primeira obteve resultados bastantes mais satisfatórios (92.8%) que a segunda (62.6%).

4. Acerca das funções de ativação, devido ao elevado número de funções existentes e de possíveis combinações, foram testadas de forma puramente aleatória. Os resultados foram bastante fracos (média de 20%).

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

Todos estes resultados estão descritos pormenorizadamente no ficheiro excel anexado a este relatório.

Por fim, gravamos as duas melhores redes obtidas nesta alínea. A melhor RN que guardamos apresenta uma precisão global e de teste a 100%. A topologia usada foram 6 camadas escondidas com 50 neurónios cada, com a função de treino traincgb, e função de activação e divisão de exemplos default:

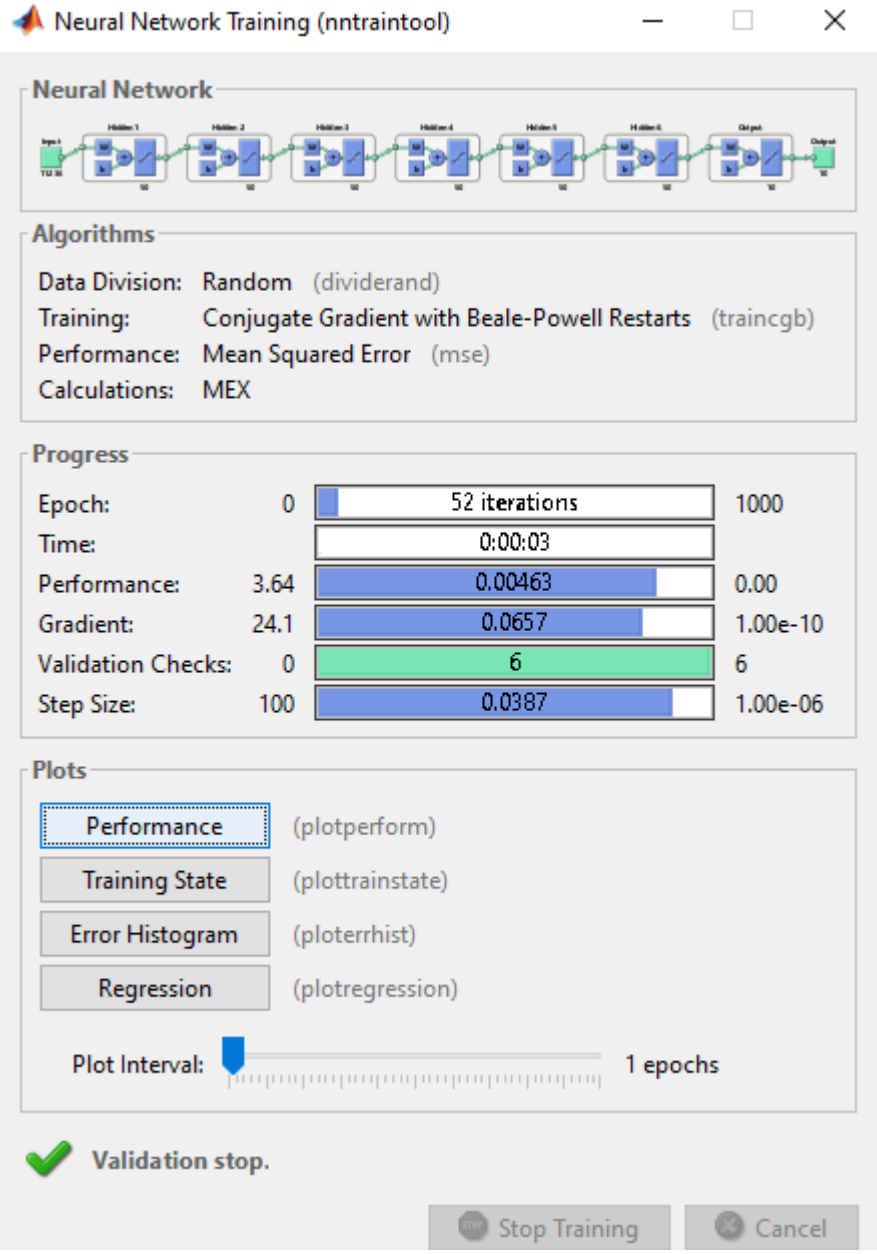

Figura 4: resultado do treino da melhor RN obtida na alínea b)

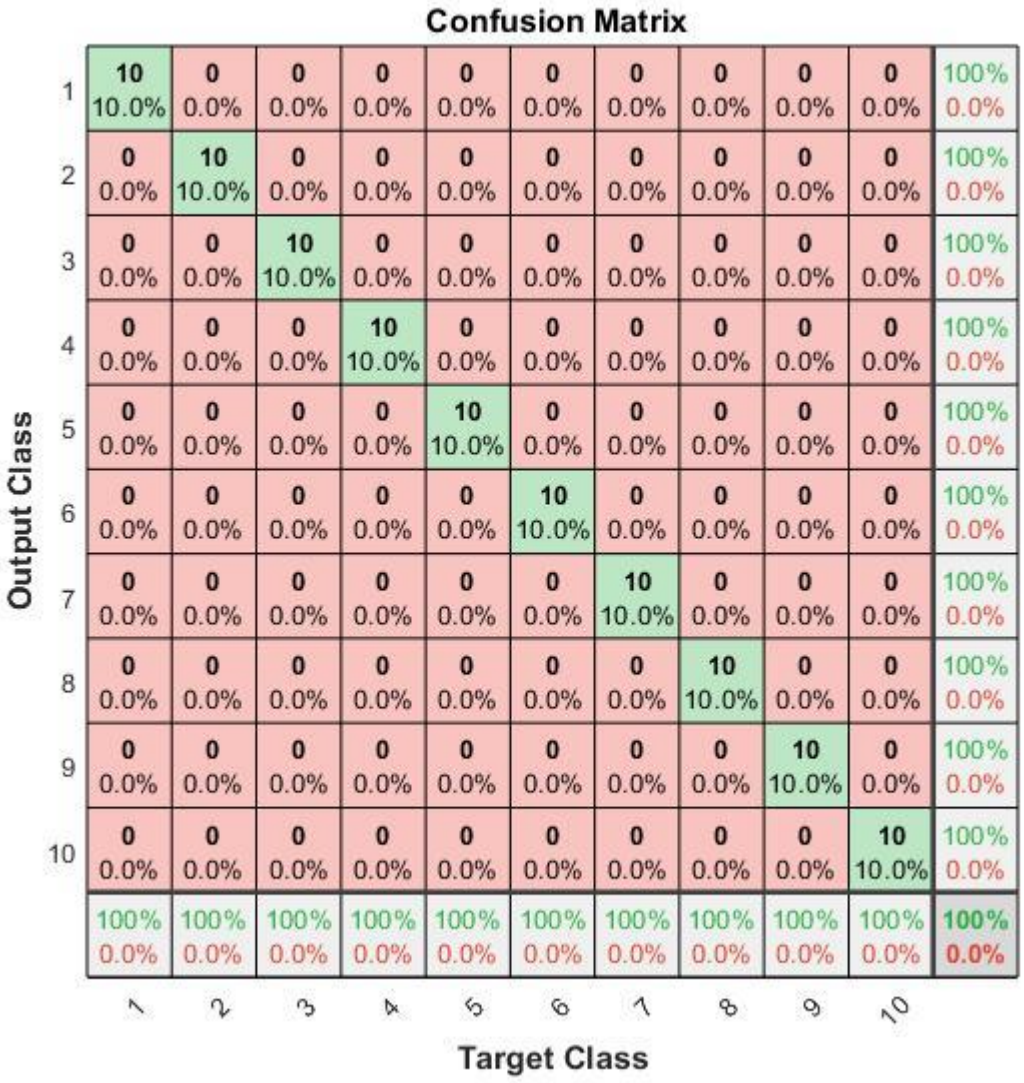

Figura 5: matriz de confusão da melhor RN obtida na alínea b)

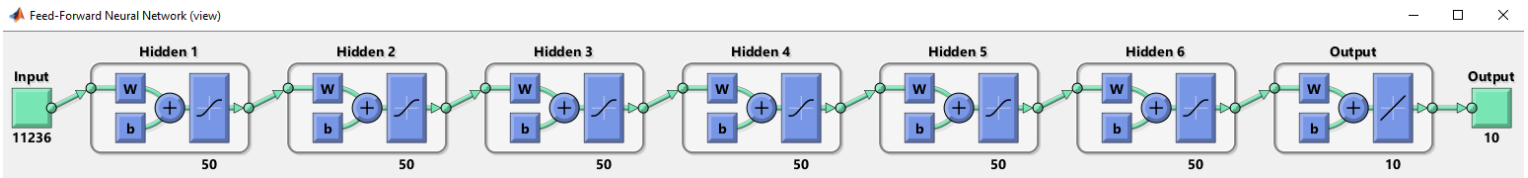

Figura 6: topologia da melhor RN obtida na alínea b)

Os gráficos de performance, estado de treino, histograma de erros e regressão desta RN estão também anexados a este relatório, juntamente com a rede guardada.

Guardamos também o que consideramos a segunda melhor RN obtida nesta alínea, com uma topologia diferente (4 camadas escondidas de 40 neurónios cada), mas com a mesma função de treino (traincgb), e restantes parâmetros em default. Esta RN apresenta uma precisão global de 97% e de 86% de teste. Optámos por guardar esta rede para efeitos de teste para a alínea c).

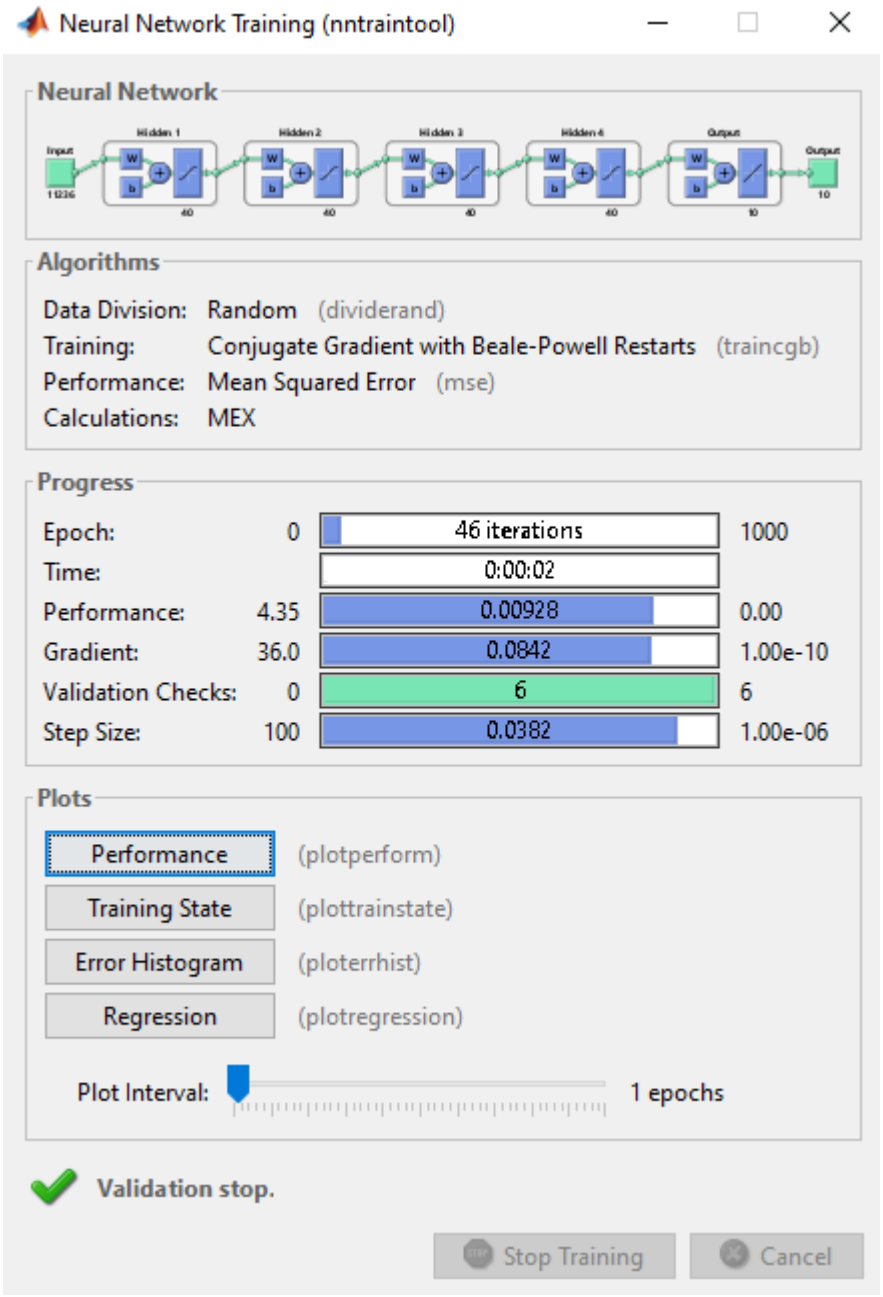

Figura 7: resultado do treino da segunda melhor RN obtida na alínea b)

Quanto à matriz de confusão, esta RN falhou na classificação de uma letra Épsilon, uma letra Pi, e uma letra Ró. Ou seja, conseguiu classificar 97 imagens corretamente em 100.

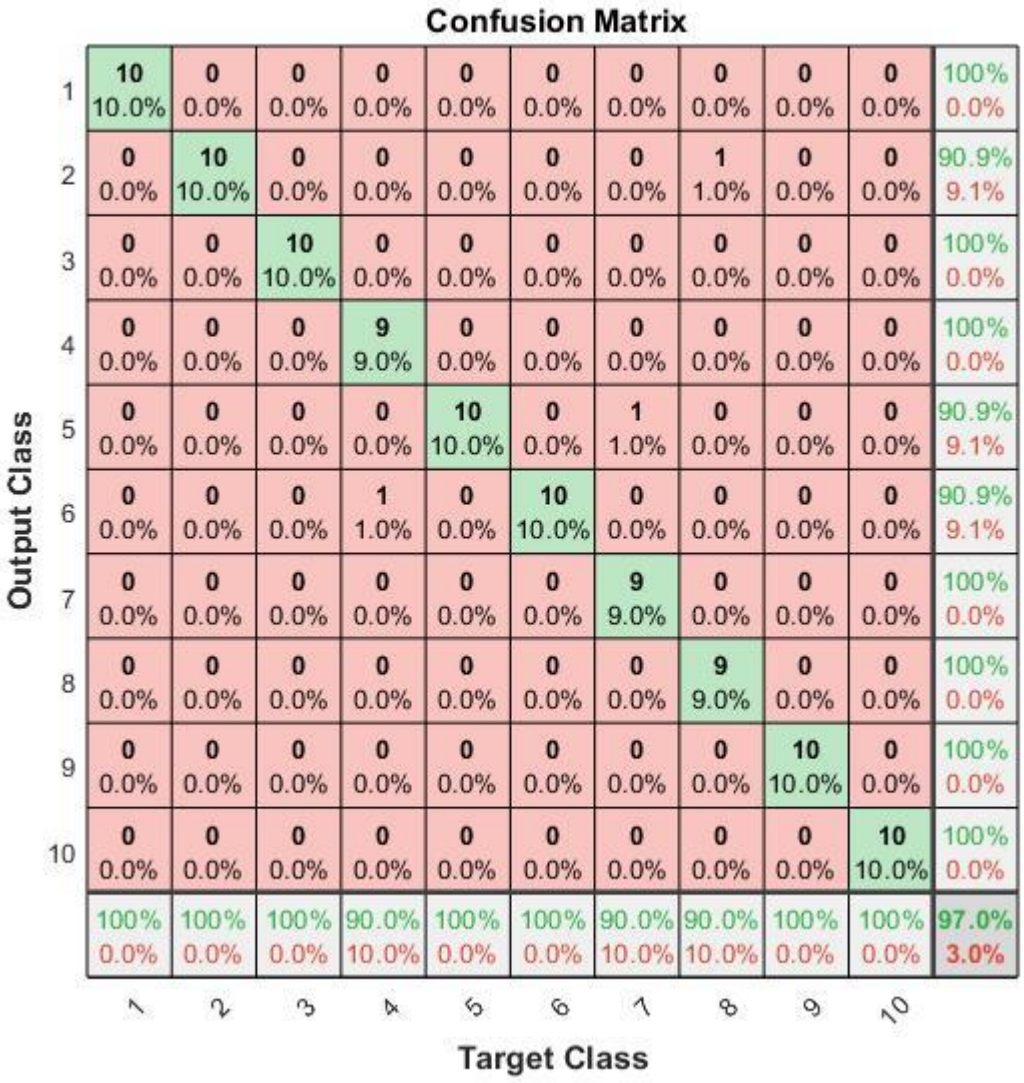

Figura 8: matriz de confusão da segunda melhor RN obtida na alínea b)

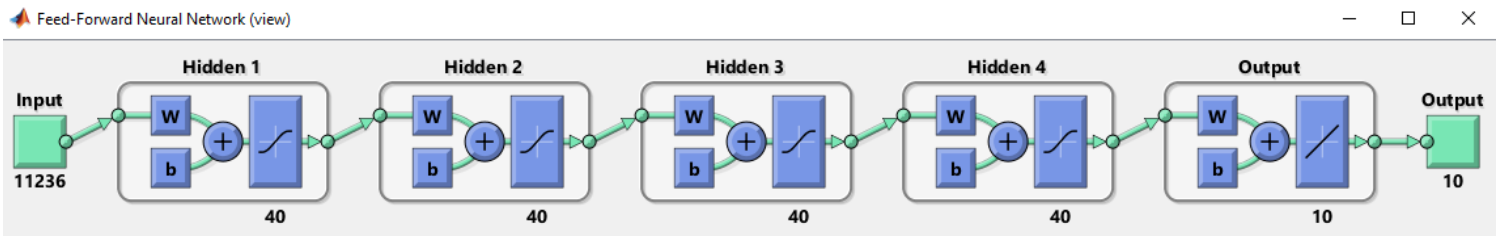

Figura 9: topologia da segunda melhor RN obtida na alínea b)

Os gráficos de performance, estado de treino, histograma de erros e regressão desta RN estão também anexados a este relatório, juntamente com a rede guardada.

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 3. 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 3.
2. Treinar a RN para os exemplos da pasta 3, e testar as melhores RN obtidas com a pasta 1, 2 e 3, separadamente.
3. Voltar a treinar a RN, mas desta vez para o conjunto de imagens das 3 pastas (150 exemplos), e testar as melhores RN para cada pasta, separadamente.

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

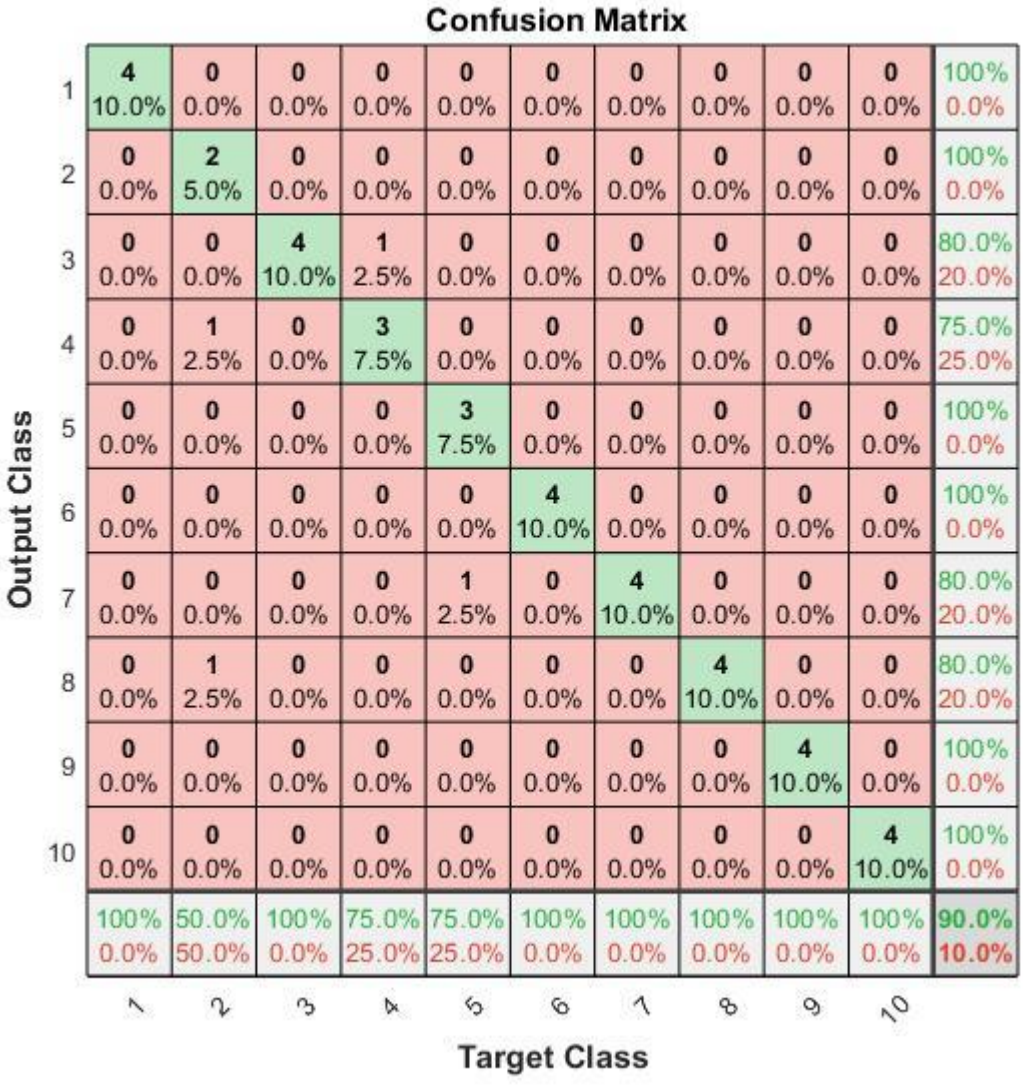

Figura 10: matriz de confusão da melhor RN obtida na alínea b), aplicada à pasta 3

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 100% de precisão global quando treinada para as imagens da pasta 2, o que nos diz que algumas das imagens providenciadas pela pasta 3 estão desenhadas de maneira diferente, em especial a letra Beta, que neste caso a nossa RN só acertou 2 em 4.

Testamos também na nossa segunda melhor RN da alínea b), e o resultado foi este:

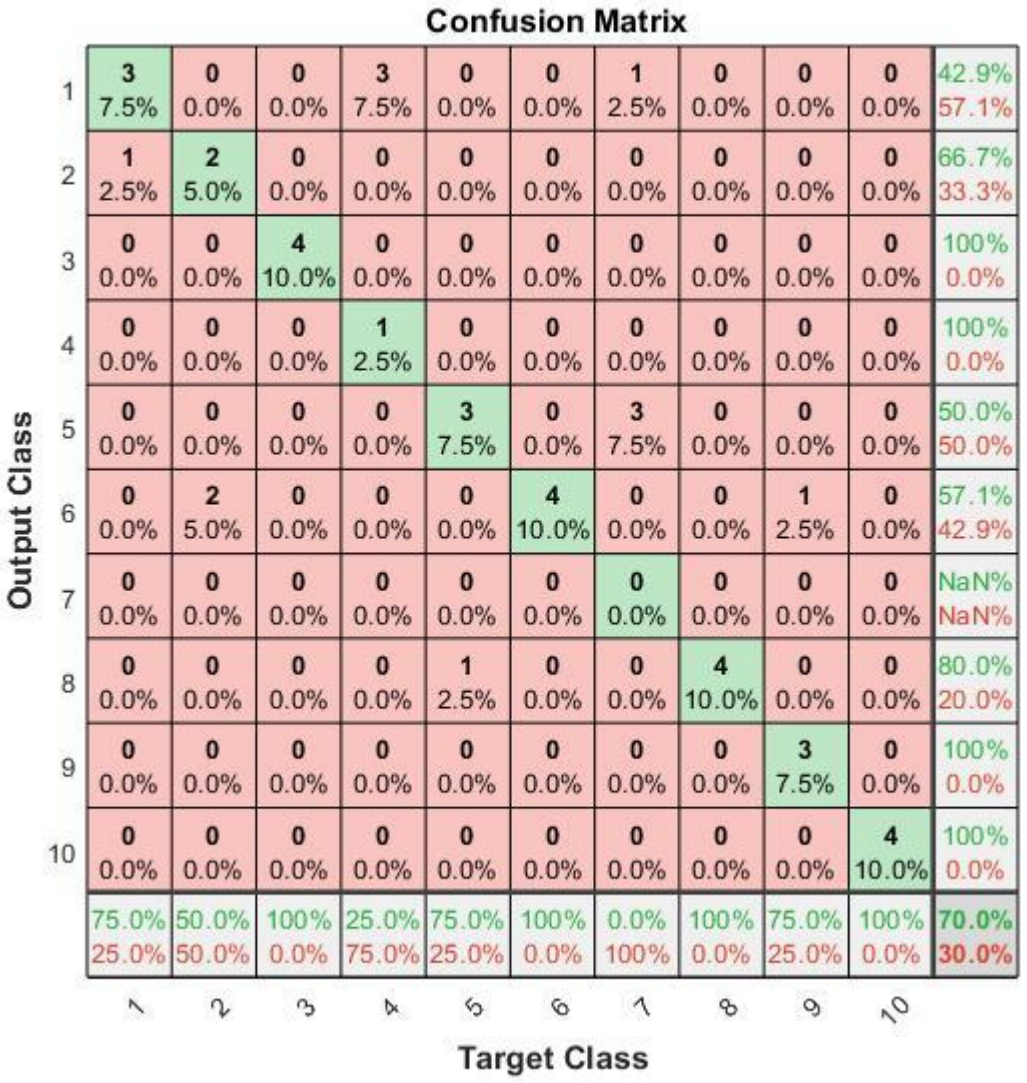

Figura 11: matriz de confusão da segunda melhor RN obtida na alínea b), aplicada à pasta 3

Como seria de esperar, esta rede obteve resultados inferiores em relação à primeira RN testada. De notar que:

- Novamente na letra Beta, houve um decréscimo de 50% de precisão.
- Na letra Épsilon, só conseguiu acertar uma imagem. Relembro que na matriz de confusão desta rede apresentada na alínea b), falhou na classificação de uma imagem desta letra.

- Na letra Pi, falhou todos os exemplos. Nos exemplos da pasta 2 também falhou na classificação de uma imagem desta letra.
- Apesar de ter falhado na alínea b) na classificação de um Ró, esta RN quando usada na pasta 3 conseguiu classificar todos os Ró com sucesso.

Daqui concluímos principalmente que há letras nesta pasta 3 desenhadas de forma significativamente diferente quando comparada com a pasta 2, especialmente a letra Beta, pois em ambas as RN existiram falhas na classificação desta letra.

Agora em relação ao **ponto 2** (treinar a rede para os exemplos da pasta 3 e testagem), voltamos a pegar nos mesmos parâmetros das nossas duas melhores RN, ou seja, voltamos a treinar com os seguintes parâmetros:

- 6 camadas escondidas com 50 neurónios cada, função de treino traincgb, restantes parâmetros default:

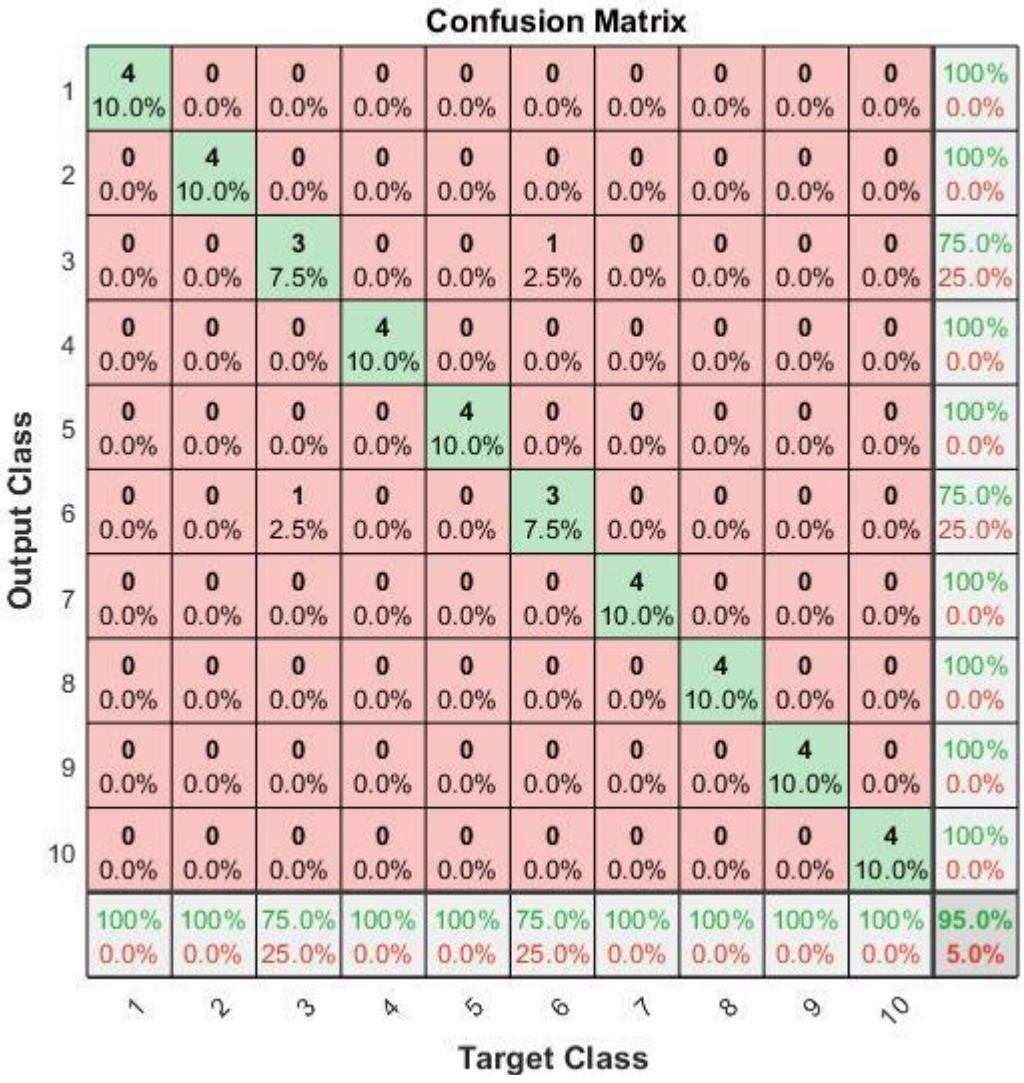

Figura 12: matriz de confusão da melhor RN obtida com exemplos da pasta 3

Em que obtemos uma precisão global de 95% e uma precisão de teste a 66%, e ainda:

- 4 camadas escondidas com 40 neurónios cada, função de treino traincgb, restantes parâmetros default:

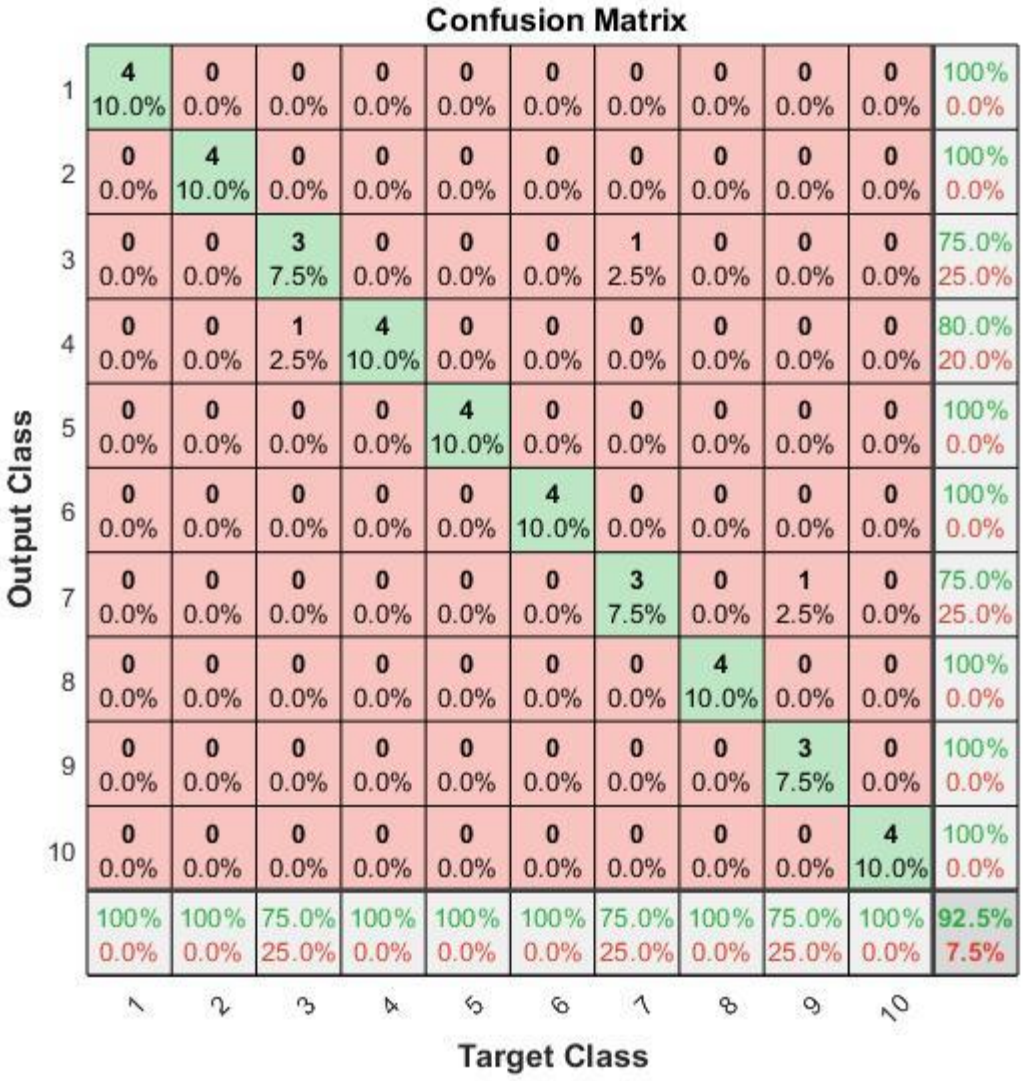

Figura 13: matriz de confusão da segunda melhor RN obtida com exemplos da pasta 3

Nesta RN obtivemos uma precisão global de 92% e uma precisão de teste a 83%. 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.

Procedemos então à testagem destas redes para a pasta 1, 2 e 3, separadamente. Os resultados obtidos foram estes:

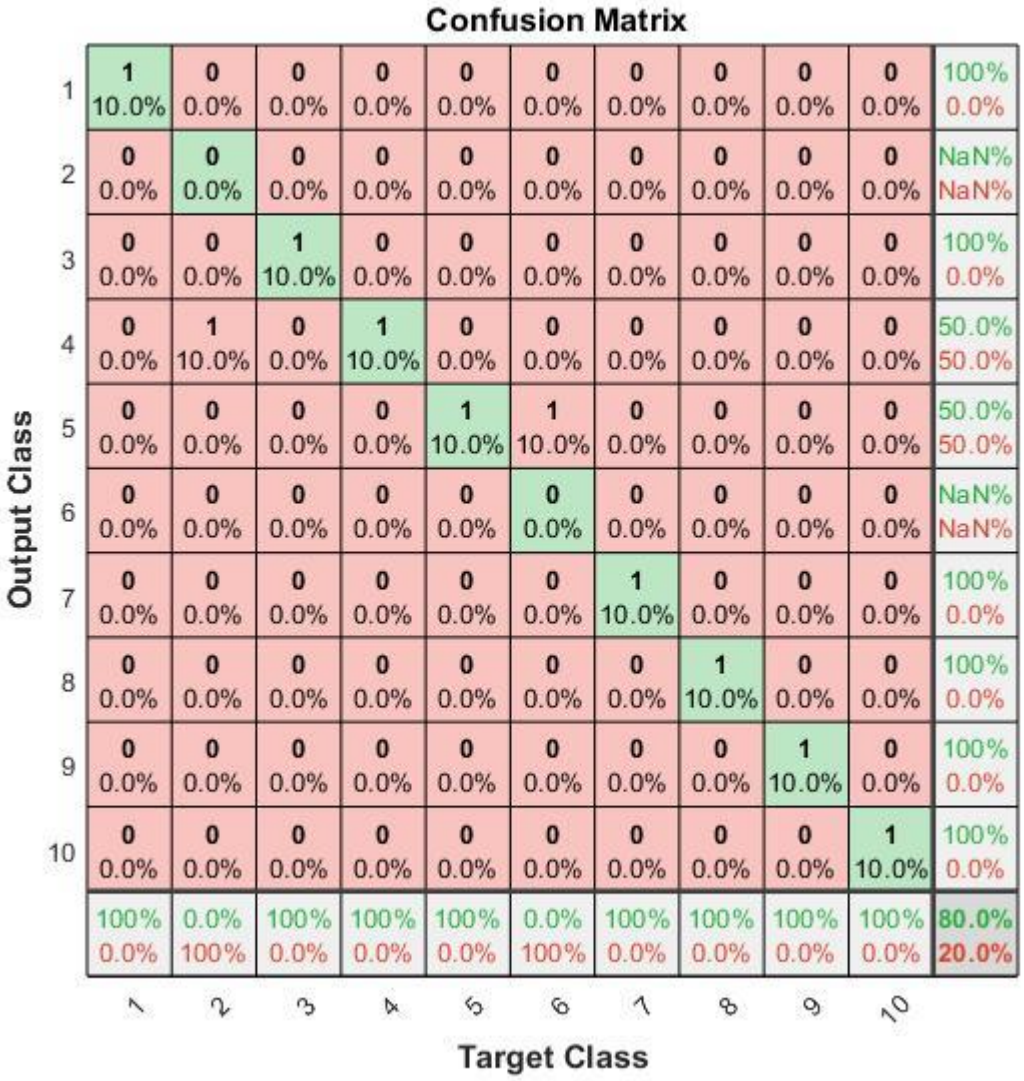

Figura 14: matriz de confusão da melhor RN obtida na alínea c) ponto 2 com exemplos da pasta 1

Obtivemos uma precisão global de 80%, ou seja, 8 em cada 10 imagens foram classificadas corretamente.

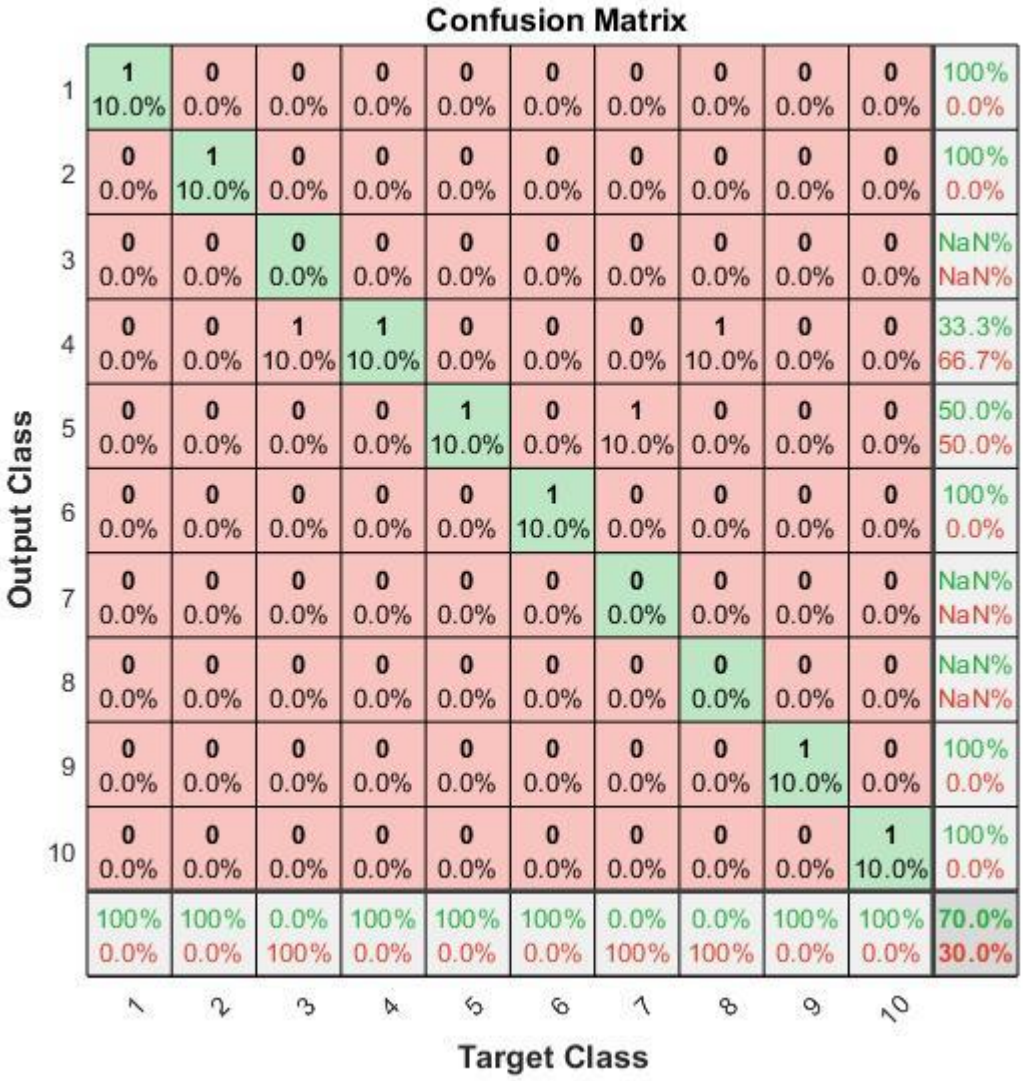

Figura 15: matriz de confusão da segunda melhor RN obtida na alínea c) ponto 2 com exemplos da pasta 1

Já aqui, obtivemos uma precisão global de 70%, mas curiosamente conseguiu classificar imagens que a nossa melhor RN não consegui classificar.

Agora para a pasta 2, foram obtidos os seguintes resultados:

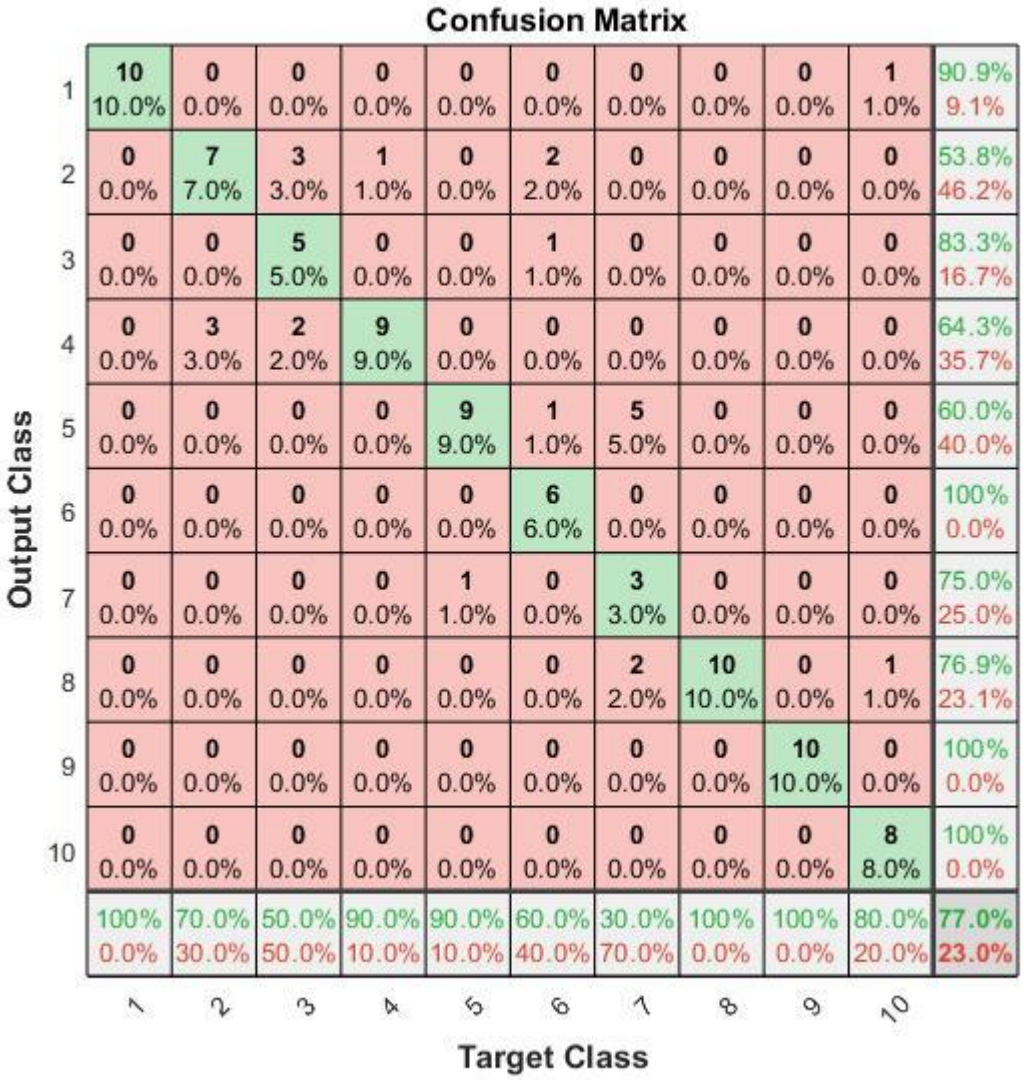

Figura 16: matriz de confusão da melhor RN obtida na alínea c) ponto 2 com exemplos da pasta 2

Mesmo quando aumentamos o número de exemplos a RN mantêm-se com valores semelhantes às da pasta 1.

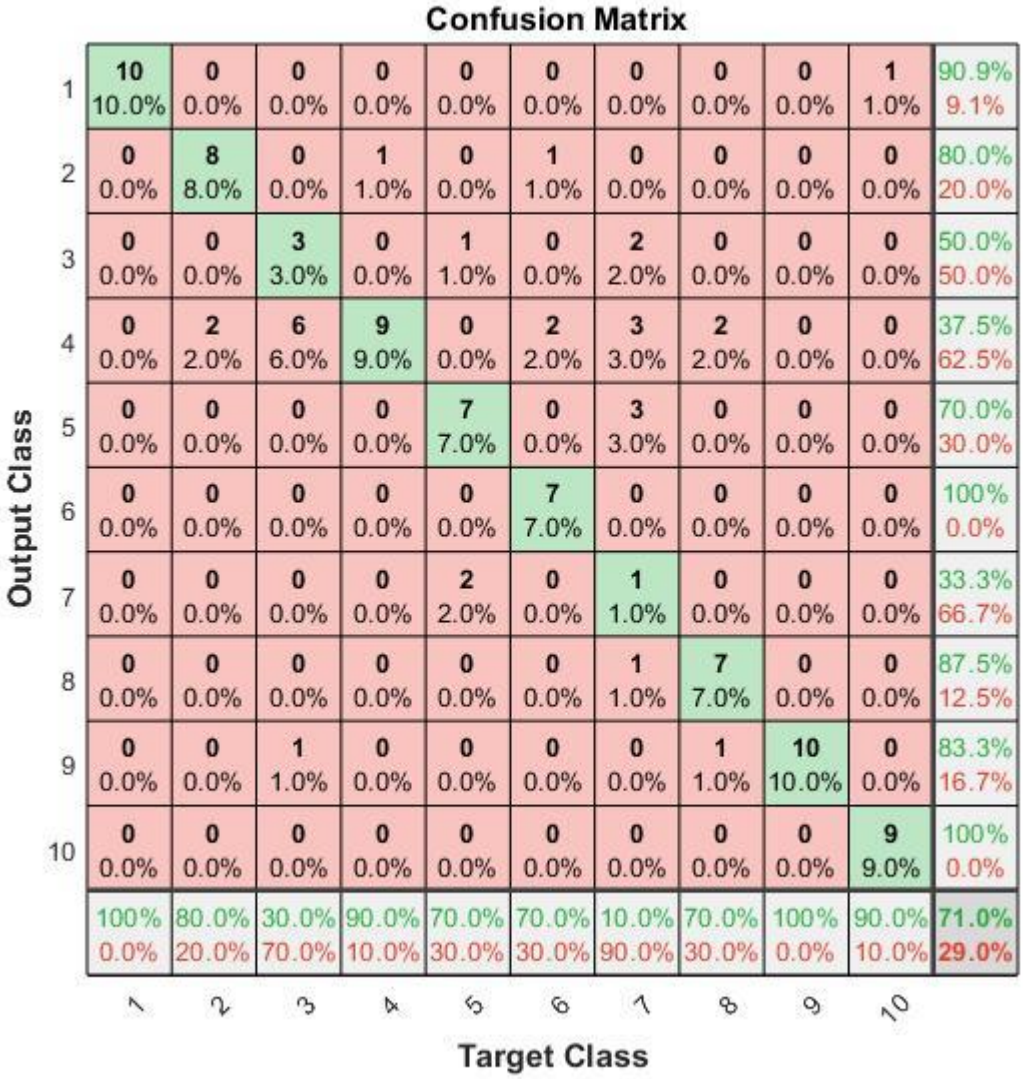

Figura 17: matriz de confusão da segunda melhor RN obtida na alínea c) ponto 2 com exemplos da pasta 2

Como podemos observar, as nossas RN tiveram mais dificuldade em classificar a letra Pi da pasta 2.

Em relação à testagem para a pasta 3, os resultados da matriz de confusão são iguais à da figura 11 e 12.

Concluímos que cada RN tem resultados melhores quando treinada para determinada pasta, quando testada para outra há tendência para existir um decréscimo (pequeno) da precisão global. Quanto maior for o número de imagens a analisar, maior o decréscimo.

Por fim no **ponto 3** vamos pegar em todas as imagens fornecidas das três pastas (150 imagens) e voltar a testar em separado para as mesmas. Novamente, como no ponto 2, vamos treinar duas redes com os mesmos parâmetros (6 camadas escondidas com 50 neurónios cada,

função de treino traincgb, restantes parâmetros default; 4 camadas escondidas com 40 neurónios cada, função de treino traincgb, restantes parâmetros default). Obtivemos a primeira com 98% de precisão global e 95% de precisão de teste, e a segunda com 98% de precisão global 91% e de precisão de teste. Como seria de esperar, devido ao elevado número de imagens, o número de iterações é maior em relação às anteriores:

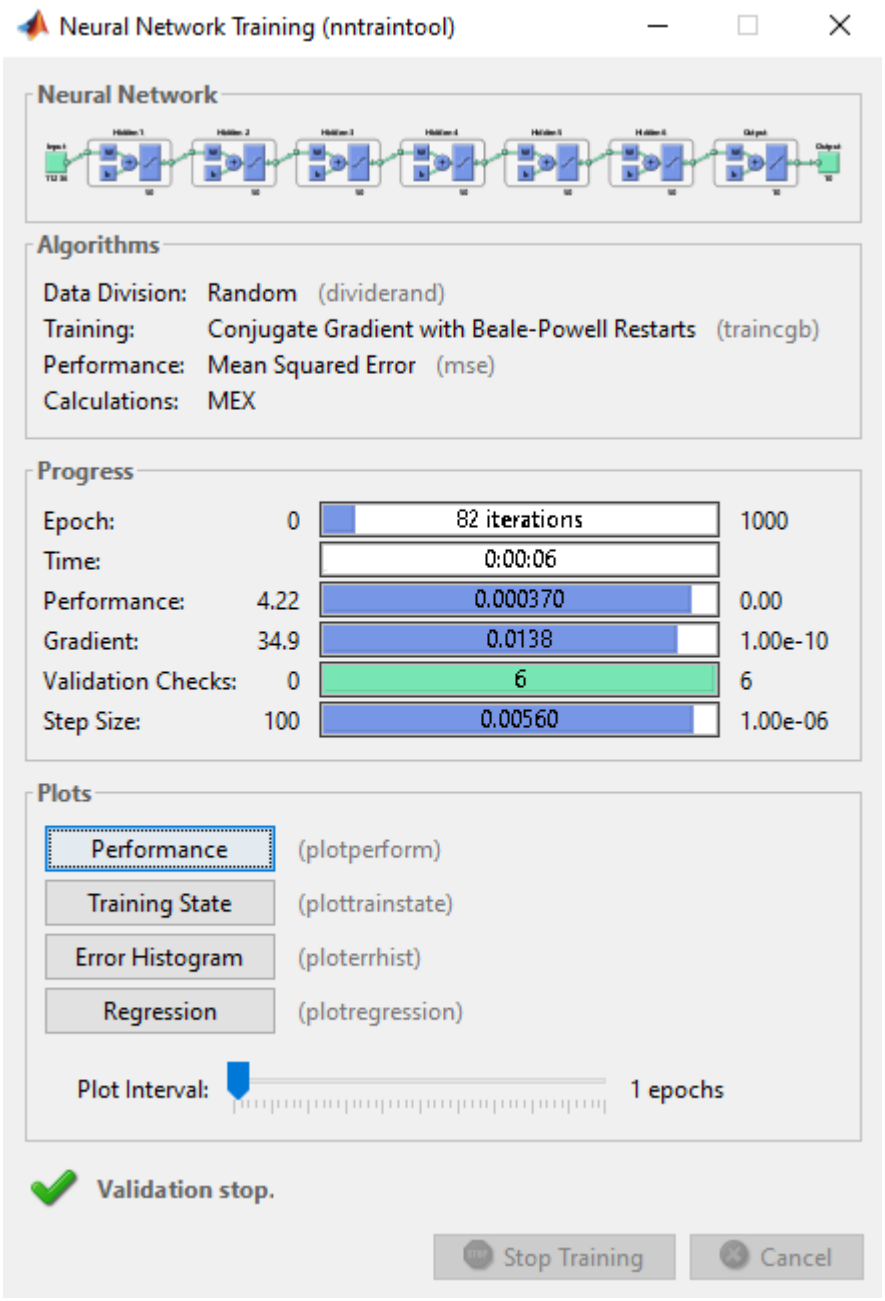

Figura 18: resultado do treino da melhor RN obtida na alínea c) ponto 3

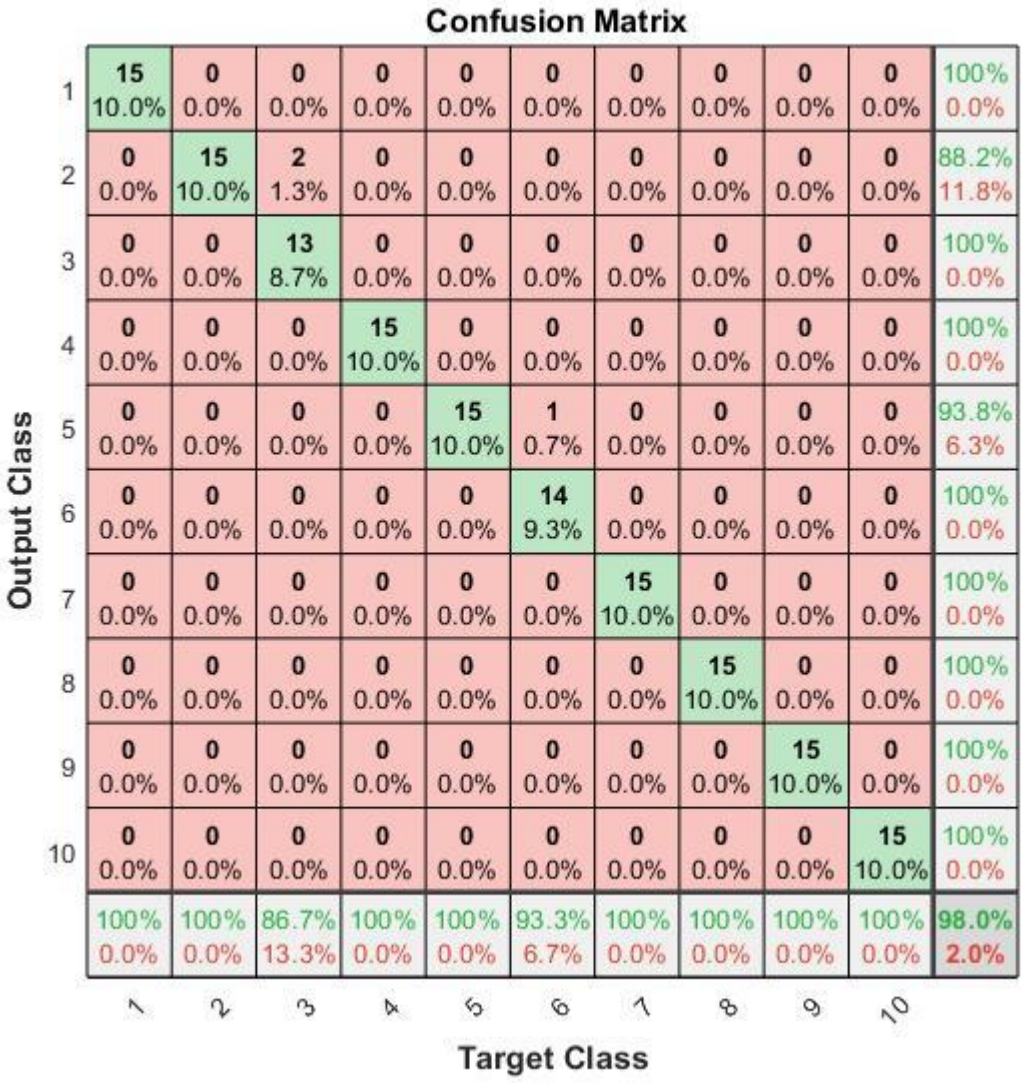

Figura 19: matriz de confusão da melhor RN obtida na alínea c) ponto 3

Como podemos observar, esta RN falhou na classificação de duas letras Gama e uma letra Teta.

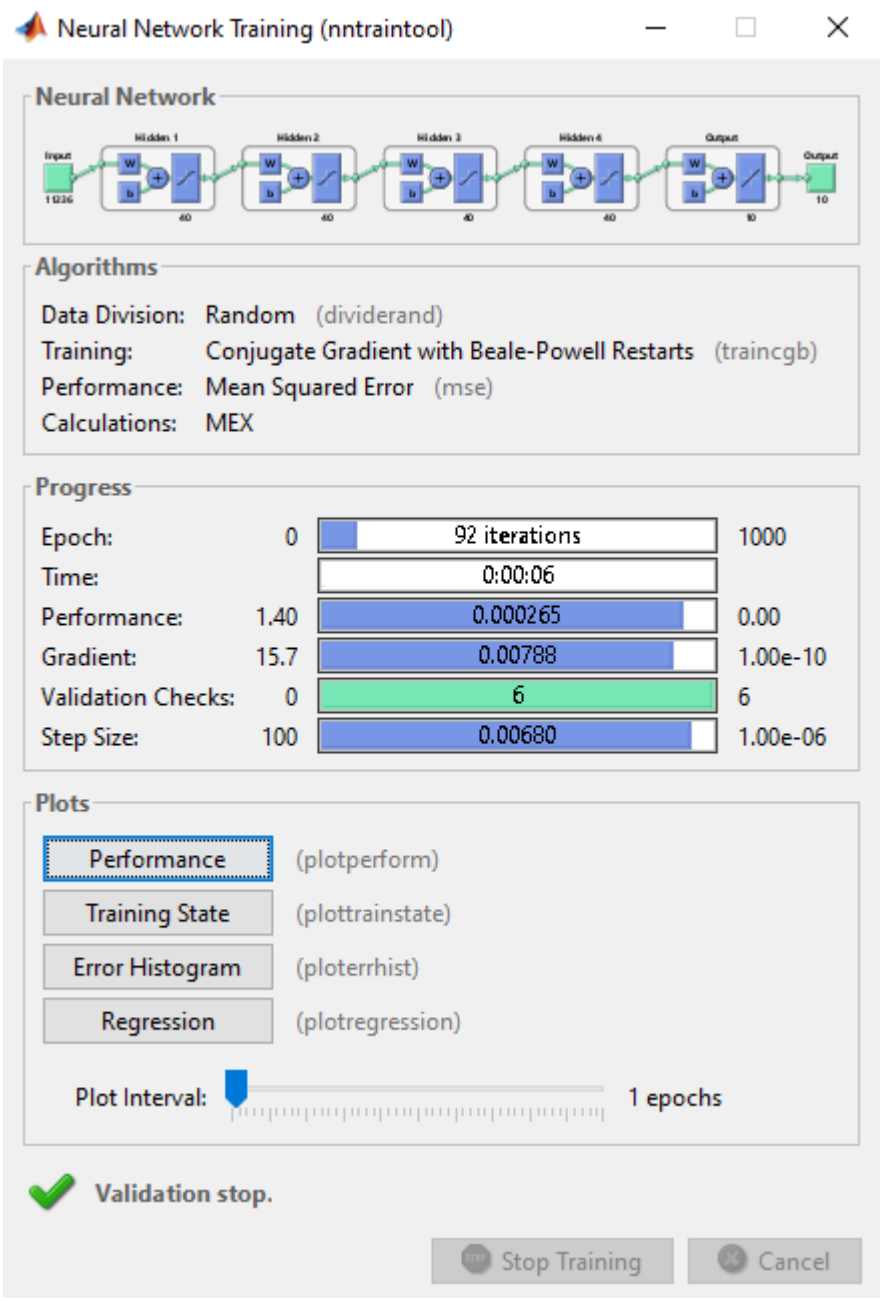

Figura 20: resultado do treino da segunda melhor RN obtida na alínea c) ponto 3

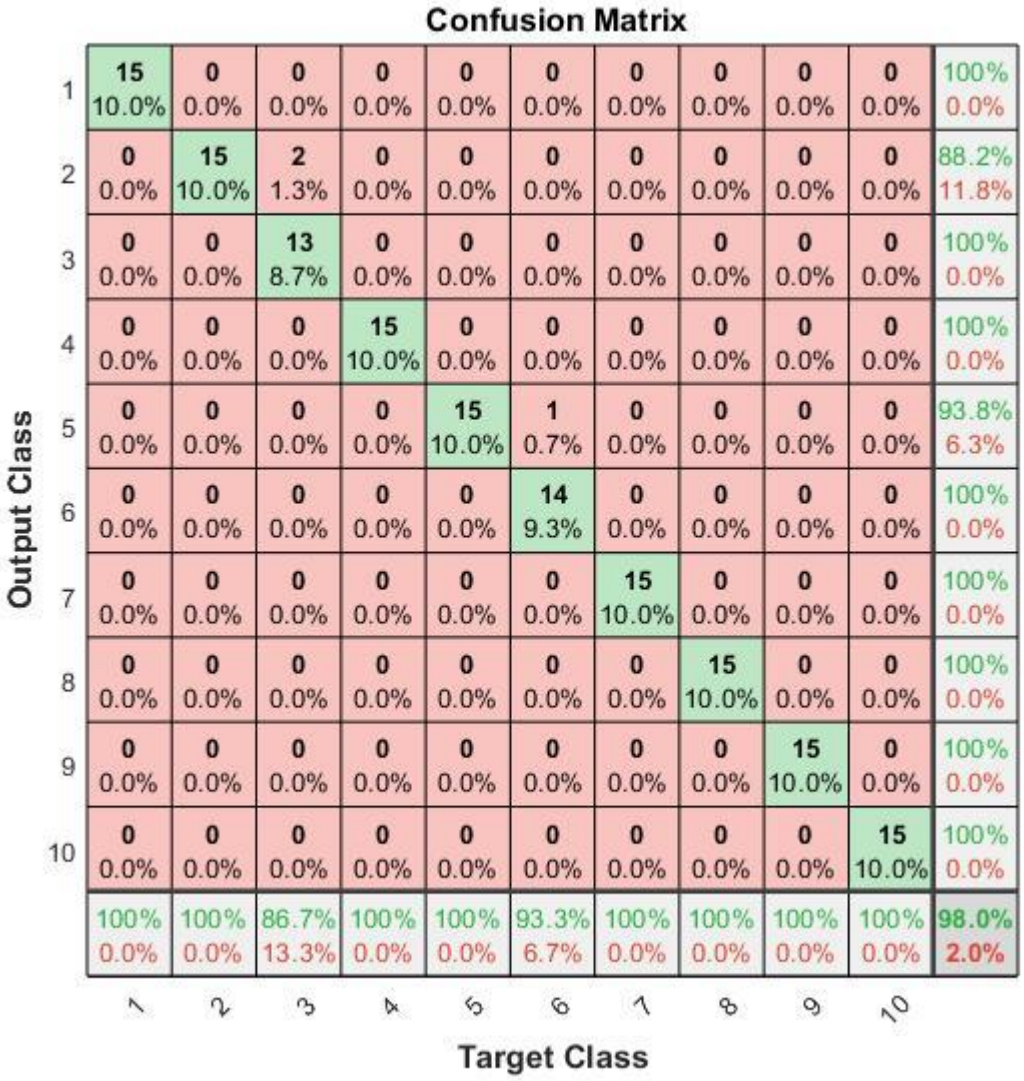

Figura 21: matriz de confusão da segunda melhor RN obtida na alínea c) ponto 3

Esta rede que obteve a mesma precisão global que a primeira, curiosamente também falhou exatamente na mesma classificação de letras da primeira.

Quanto à testagem com as três pastas em separado, os resultados foram estes (volto a frisar que os gráficos mais detalhados estão anexados a este relatório):

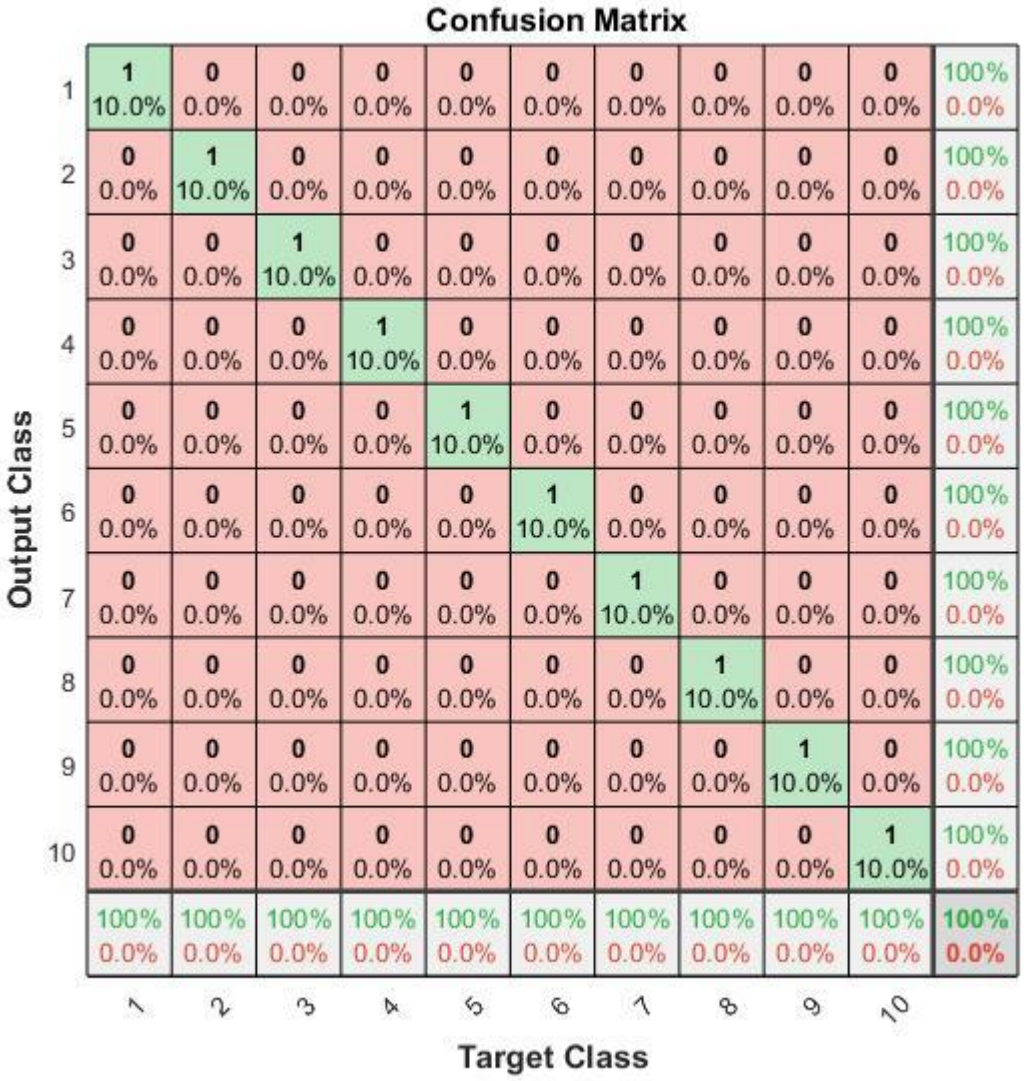

Figura 22: matriz de confusão da melhor RN obtida na alínea c) ponto 3 com exemplos da pasta 1

Esta RN com a pasta 1 obteve uma classificação perfeita.

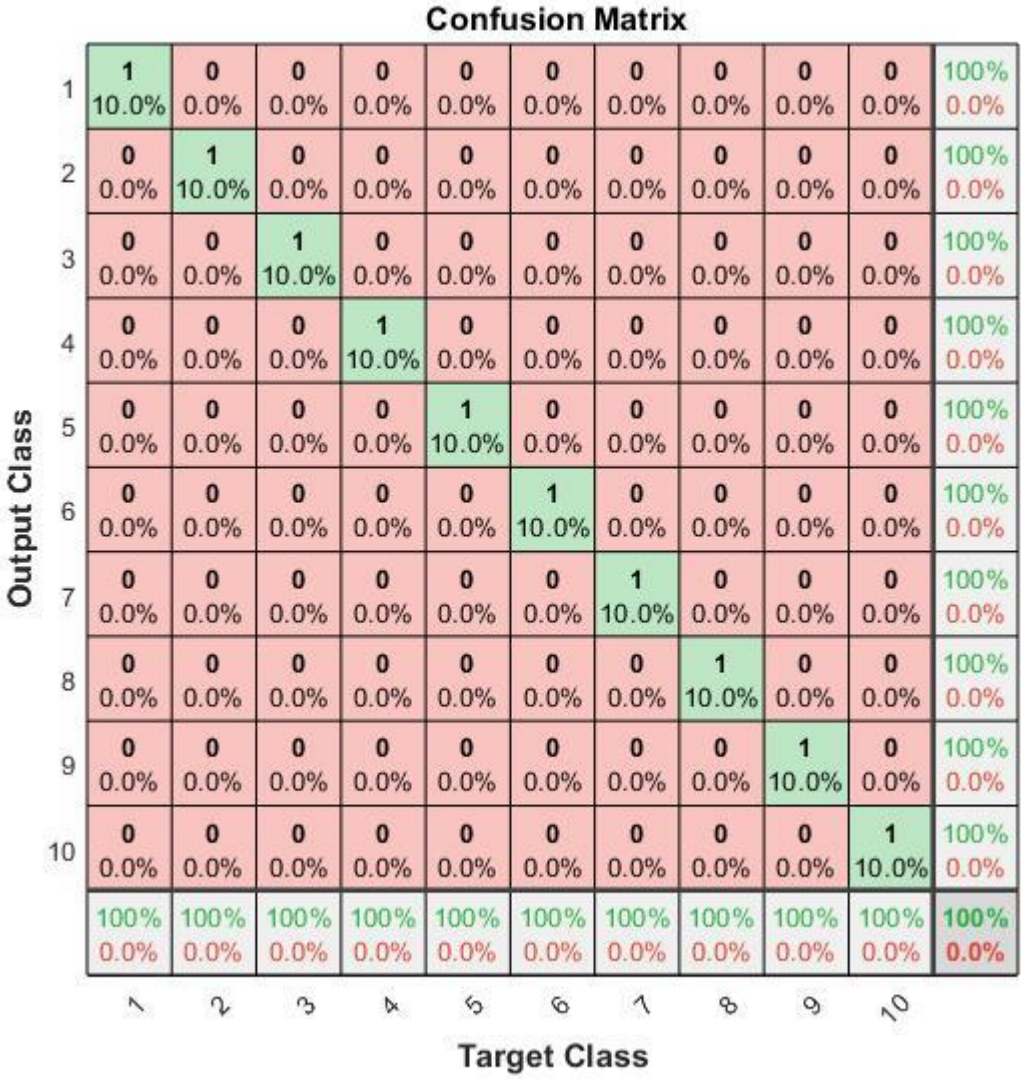

Figura 23: matriz de confusão da segunda melhor RN obtida na alínea c) ponto 3 com exemplos da pasta 1

A segunda melhor RN também obteve um resultado impecável com a pasta 1. Vamos agora verificar como se comportam para a pasta 2:

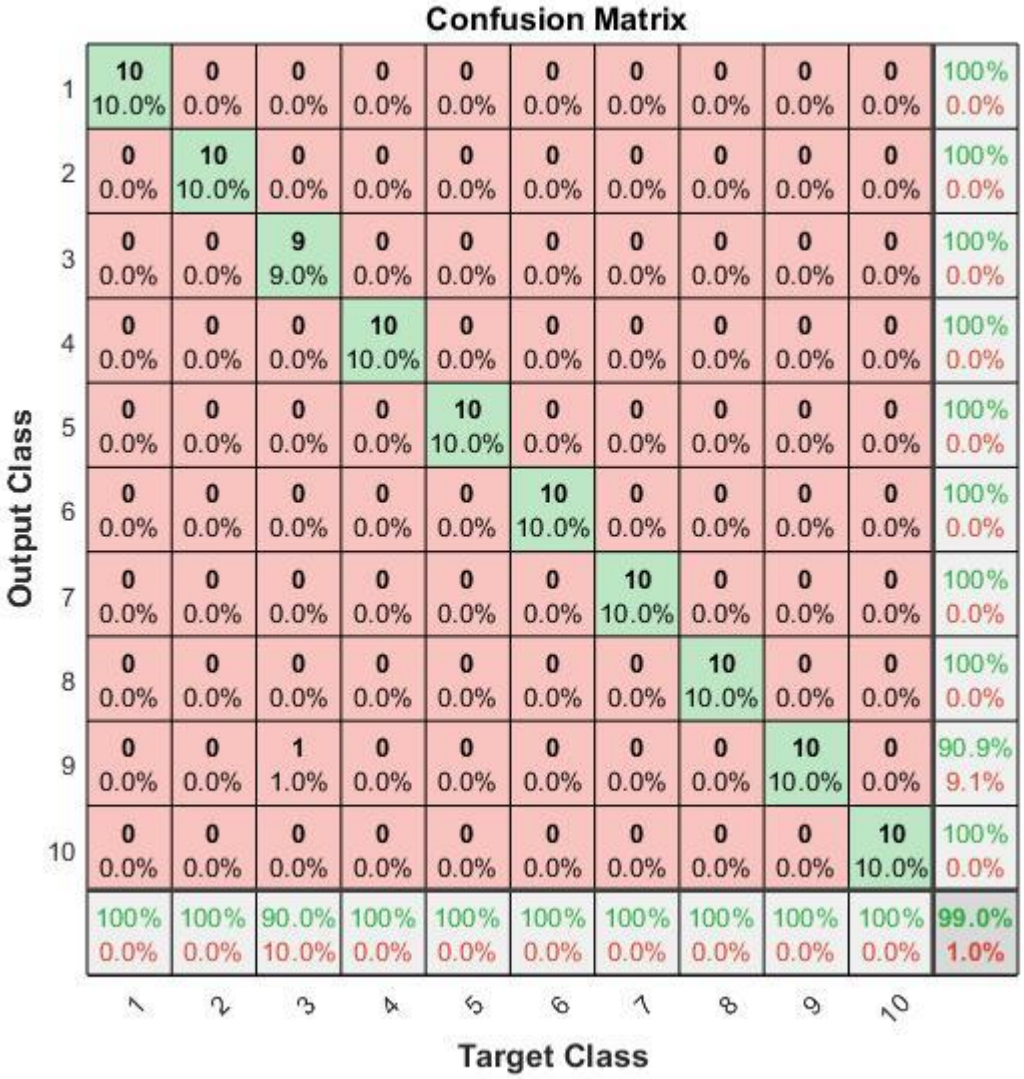

Figura 24: matriz de confusão da melhor RN obtida na alínea c) ponto 3 com exemplos da pasta 2

Nesta pasta houve apenas uma imagem mal classificada – a letra Gama.

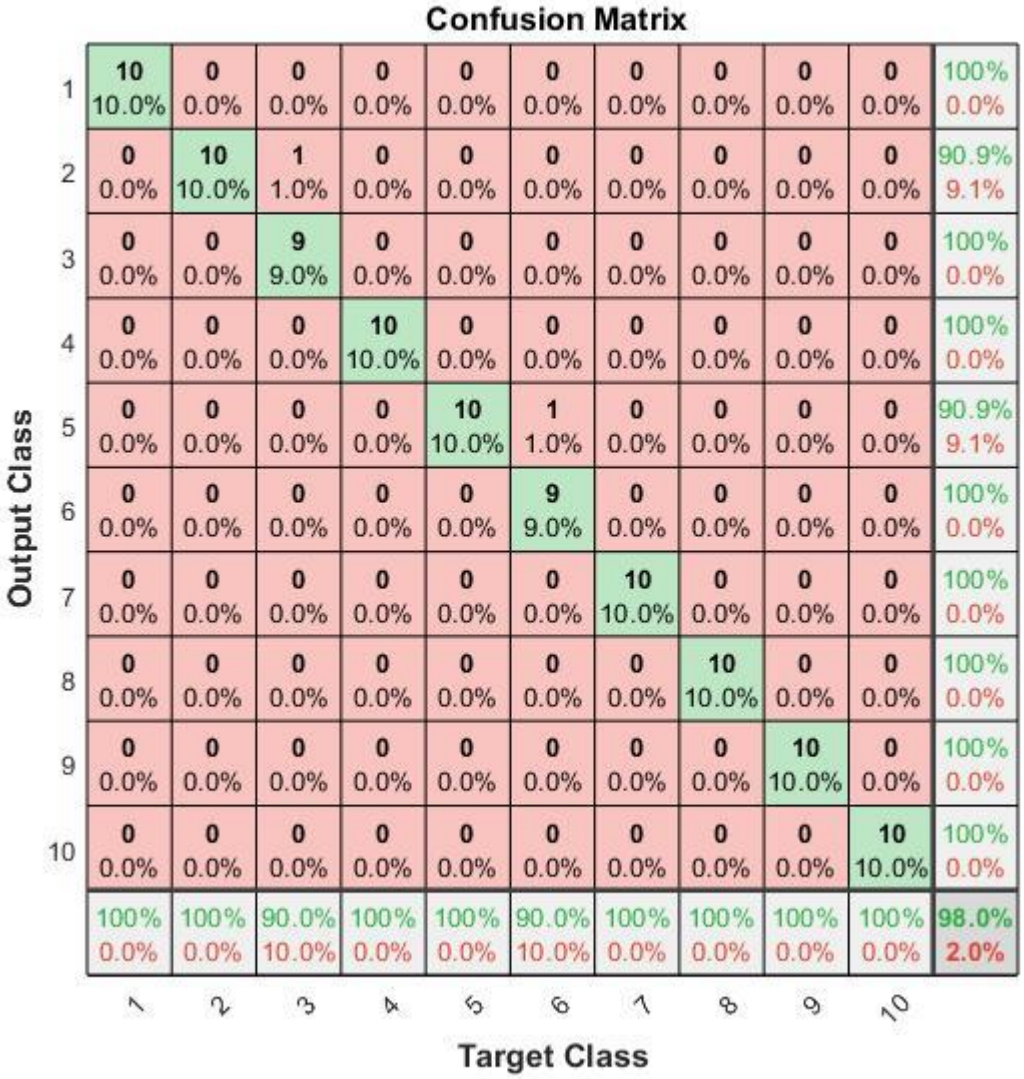

Figura 25: matriz de confusão da segunda melhor RN obtida na alínea c) ponto 3 com exemplos da pasta 2

Novamente, houve uma falha na classificação de uma letra Gama. Também falhou na classificação de uma letra Teta. Ainda assim, podemos considerar que as nossas duas redes apresentam precisões globais quase perfeitas. Vamos agora verificar se mantêm essa consistência para exemplos da pasta 3:

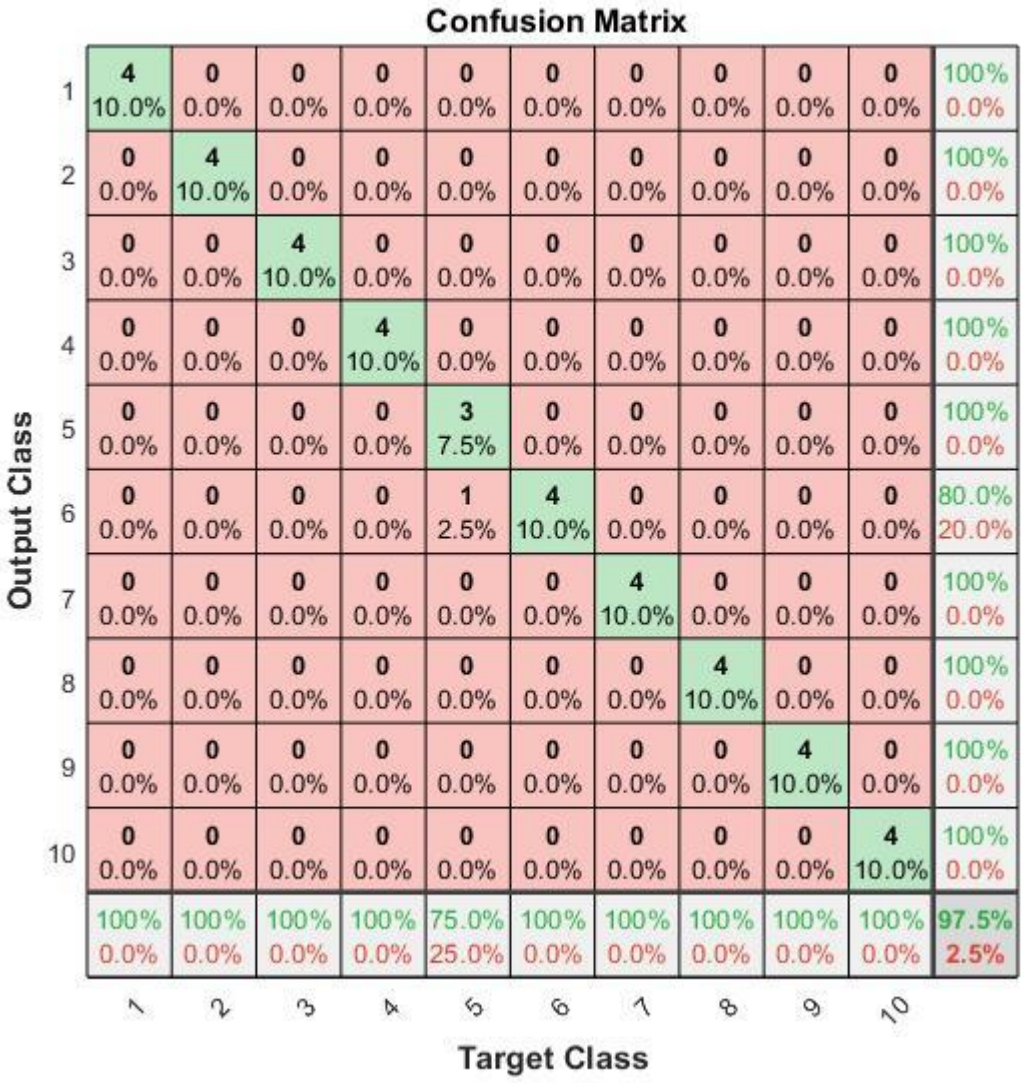

Figura 26: matriz de confusão da melhor RN obtida na alínea c) ponto 3 com exemplos da pasta 3

Com uma precisão global de 97.5%, apenas falhou na classificação da letra Eta.

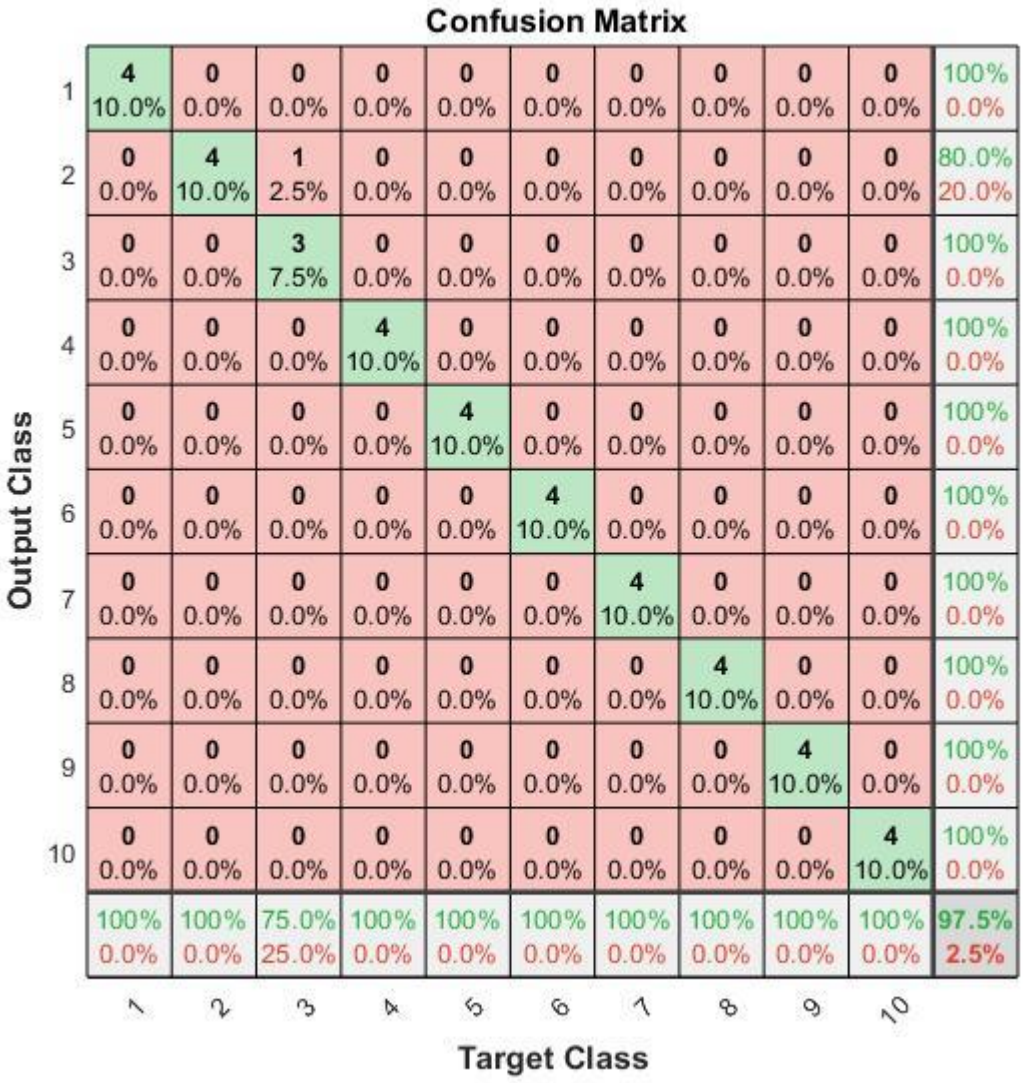

Figura 27: matriz de confusão da segunda melhor RN obtida na alínea c) ponto 3 com exemplos da pasta 3

Esta RN obteve uma precisão global igual à anterior, falhando apenas na classificação da letra Gama.

Podemos concluir que quando uma RN é treinada com todos os exemplos disponíveis, ao ser testada para conjuntos individuais (10, 100 ou 150 imagens), obtém resultados significativamente superiores do que quando treinada para um número restrito de exemplos e testada com outros que não fizeram parte do treino da RN.

Leitura de imagens - alínea d)

Para esta alínea, foi desenvolvido um pequeno programa para ler um ficheiro do tipo imagem, com uma letra desenhada e usar a melhor RN obtida na alínea c). Neste caso vamos usar a RN com a seguinte configuração: 6 camadas escondidas com 50 neurónios cada, função de treino traincgb, restantes parâmetros default (do ponto 3). De notar que incluímos uma nova função (rgb2gray(image);) para transformar a imagem de 24 bits para 8, tal como as imagens das pastas fornecidas.

Vamos pegar numa imagem desenhada à mão no paint:

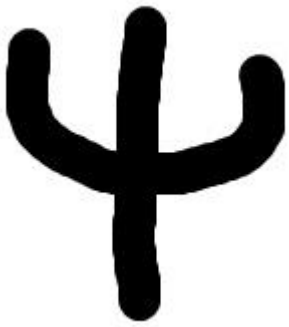

Figura 28: letra Psi desenhada à mão no paint

E ao correr o algoritmo obtemos:

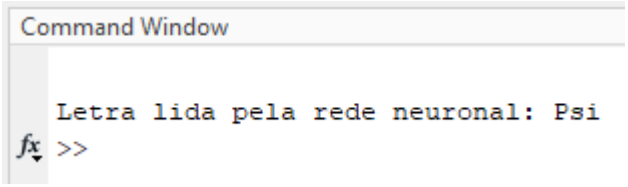

Figura 29: output da consola

Testamos com todas as letras e a RN acerta sempre exceto quando a letra é desenhada com um pincel demasiado fino. Quanto mais grosso melhores são os resultados.

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 letra, ou carregar um ficheiro de imagem onde esta já se encontre desenhada. Aplicar uma rede neuronal para classificar a letra 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:

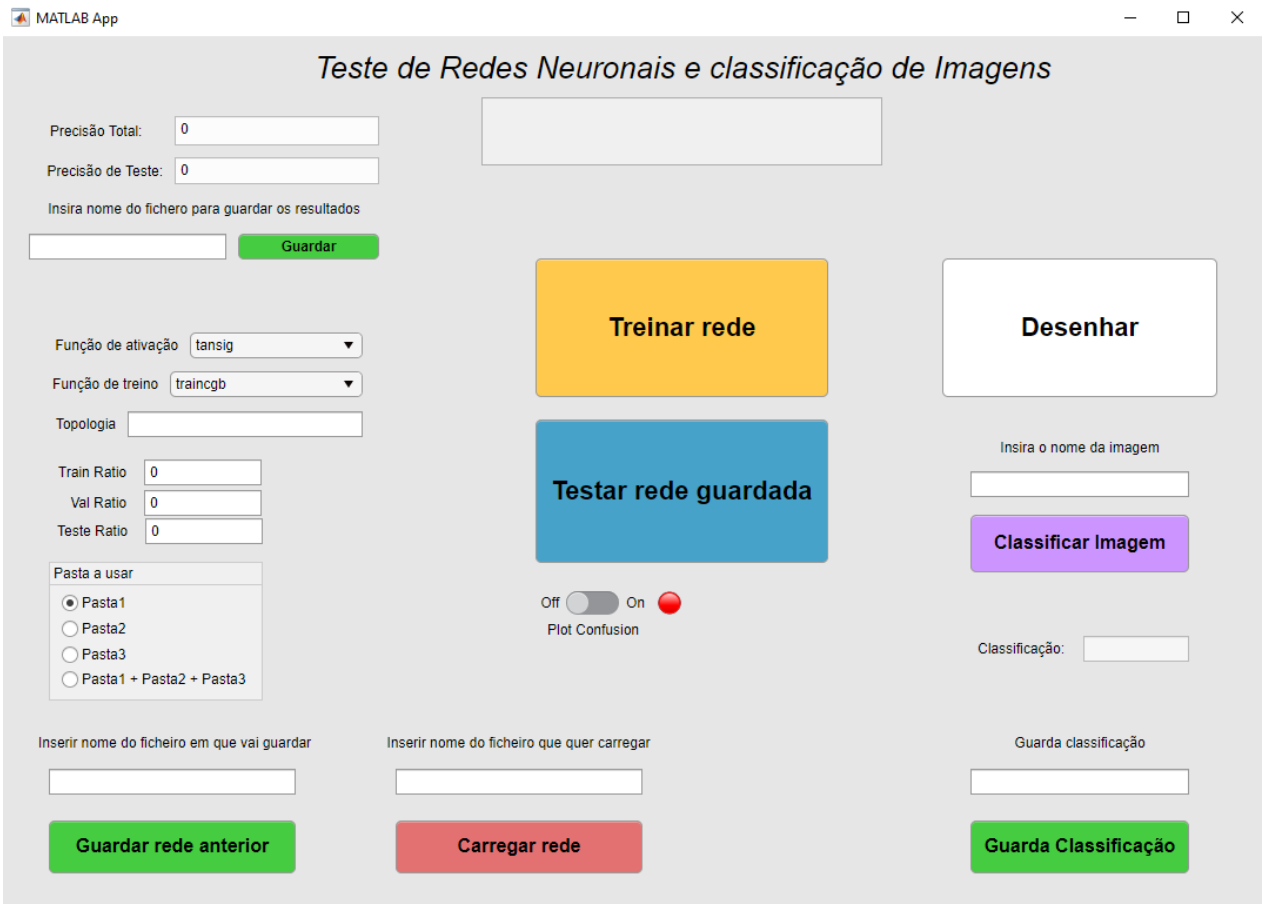

Figura 30: GUI implementada

Todos os pontos requeridos foram implementados com sucesso.

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 letras, 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.

Anexos

Enunciado

Excel alínea b)

Código alínea a)

Código alínea b)

Código alínea c)

Código alínea c) (todas as pastas)

Código alínea d)

GUI

RN alínea b)

RN alínea c)