Instituto Superior de Engenharia

Politécnico de Coimbra

Departamento de Engenharia Informática e de Sistemas
Licenciatura em Engenharia Informática
Unidade Curricular de Conhecimento e Raciocínio
Ano Letivo 2021/2022

TP – Rede Neuronal

Miguel Neves – Nº 2020146521
Nuno Domingues – Nº 2020109910

05/2022

Índice

Índice de Figuras

1. Introdução

Esta neste relatório, uma apresentação e descrição sobre a implementação/criação da rede neuronal em *Matlab* para o tema proposto do trabalho prático.

O trabalho consiste, na criação de rede neuronal, para reconhecimento de formas geométricas, nomeadamente: círculo, deltoide (*kite*), paralelogramo, quadrado, trapézio e triângulo. Para isso foram usadas três pastas, fornecidas juntamente com o enunciado do trabalho, correspondentes a três *datasets* de imagens diferentes.

Está ainda presente neste relatório, uma breve explicação/comparação dos testes realizados. E ainda um livro de Excel, onde se pode observar mais detalhadamente todos os resultados obtidos.

Por fim temos uma descrição e guião sobre a aplicação gráfica desenvolvida em *MatLab*, que permite um interação pessoa-máquina de uma forma intuitiva.

2. Configurações iniciais

Começamos por fazer o upload das imagens da pasta "*start*", e logo foi percetível que a quantidade de informação das imagens era enorme, isto porque as imagens originalmente têm uma resolução de 224 x 224 pixéis, o que significa que só numa imagem teríamos que armazenar em memória 50176 pixéis de informação.

Sendo que a pasta "*start*" (pasta com menos imagens), contém ao todo 30 imagens de todas as formas geométricas, temos cerca 1505280 pixéis de informação para processar.

A solução foi fazer um redimensionamento das imagens para 25 x 25 pixéis, o que reduz a quantidade de informação a processar.

Nota: Todas as imagens que forem carregadas tem que obrigatoriamente obedecer ao tamanho original (224 x 224), senão irá haver inconsistência dos dados e não serão classificadas corretamente.

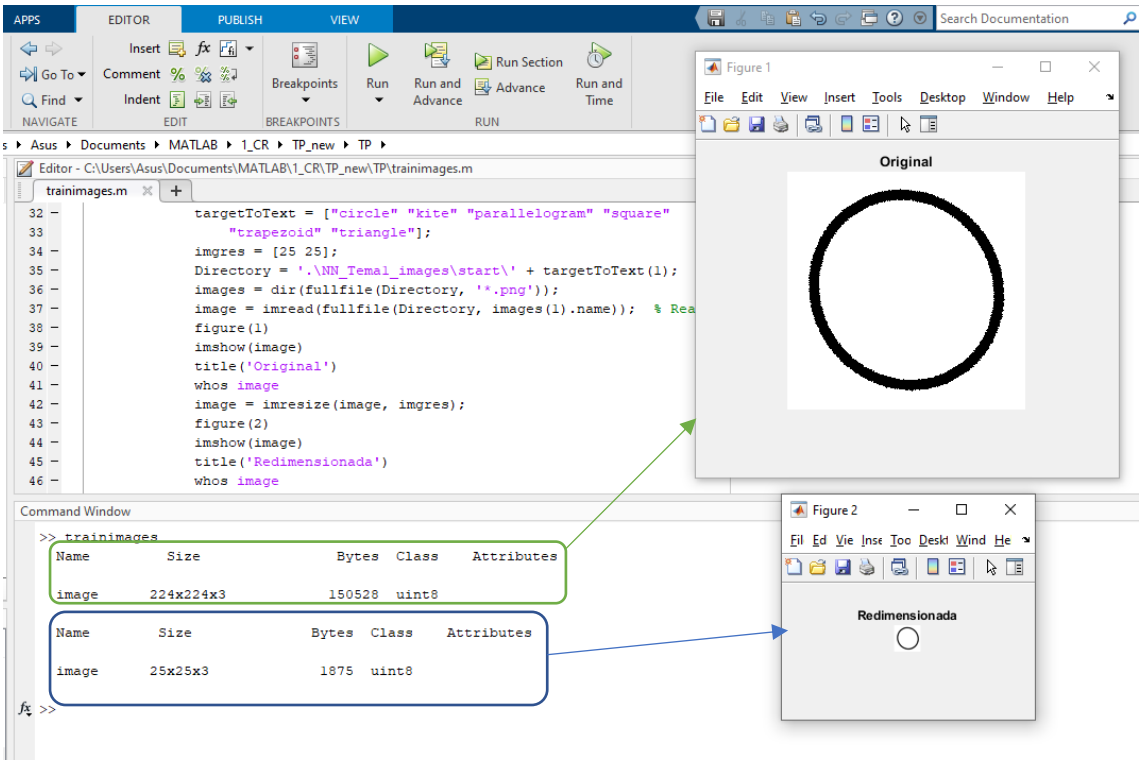


Figura 1 - Redimensionamento de imagem

3. Preparação do ambiente para treino

Existem três pastas diferentes: a "*start*","*test*" e "*train*", cada pasta contém imagens das 6 diferentes formas geométricas.

Na pasta "*start*", existem 5 imagens para cada forma geométrica, na pasta "*test*", existem 10 imagens para cada forma geométrica e na pasta "*train*" ", existem 50 imagens para cada forma geométrica.

Exemplo da pasta "*start*":

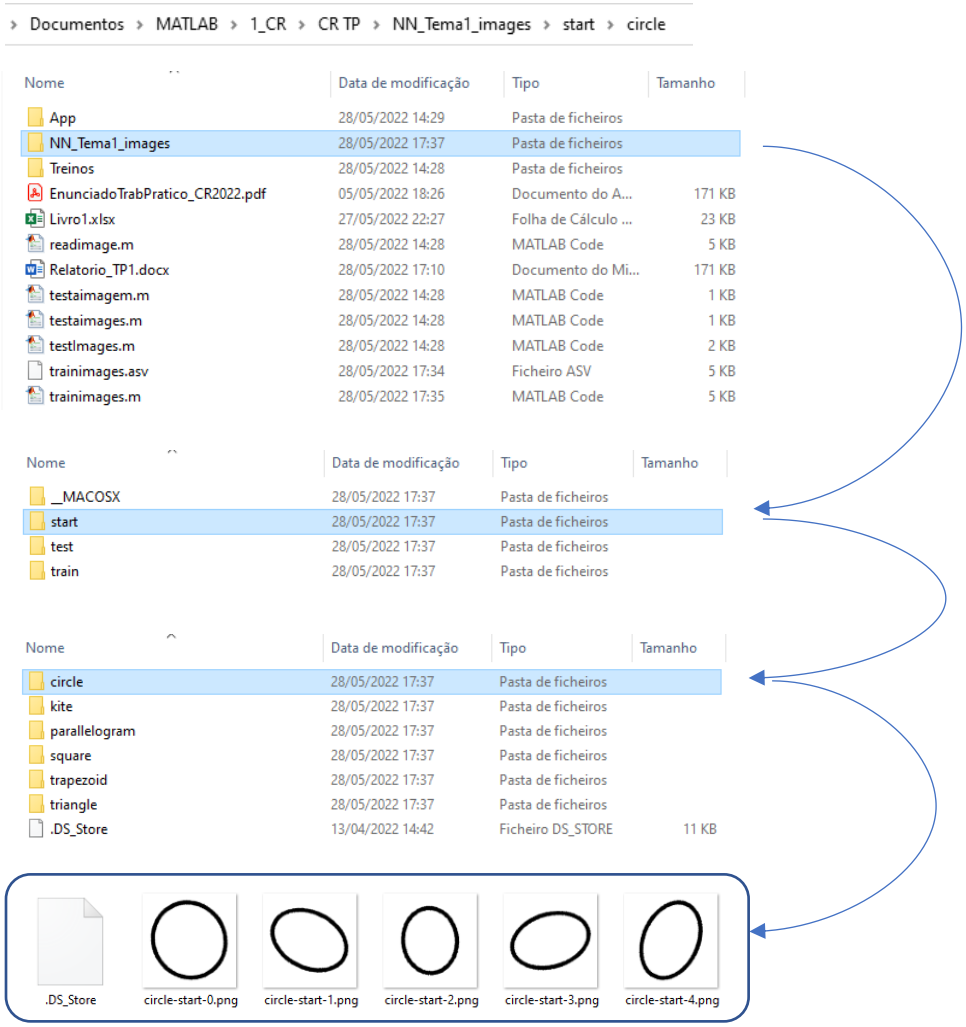


Figura 2 - Diretoria das imagens

3.1. Preparação das imagens e os targets

Vamos dar o exemplo para a pasta start e a lógica é igual para as outras pastas. Para iniciar iremos percorrer cada pasta das formas geométricas e redimensionar as imagens para 25 x 25 pixéis.

Inicializar também uma matriz de targets, que vai indicar a imagem à forma geométrica correspondente.

Basicamente o que acontece, pode ser demonstrado pela figura abaixo:

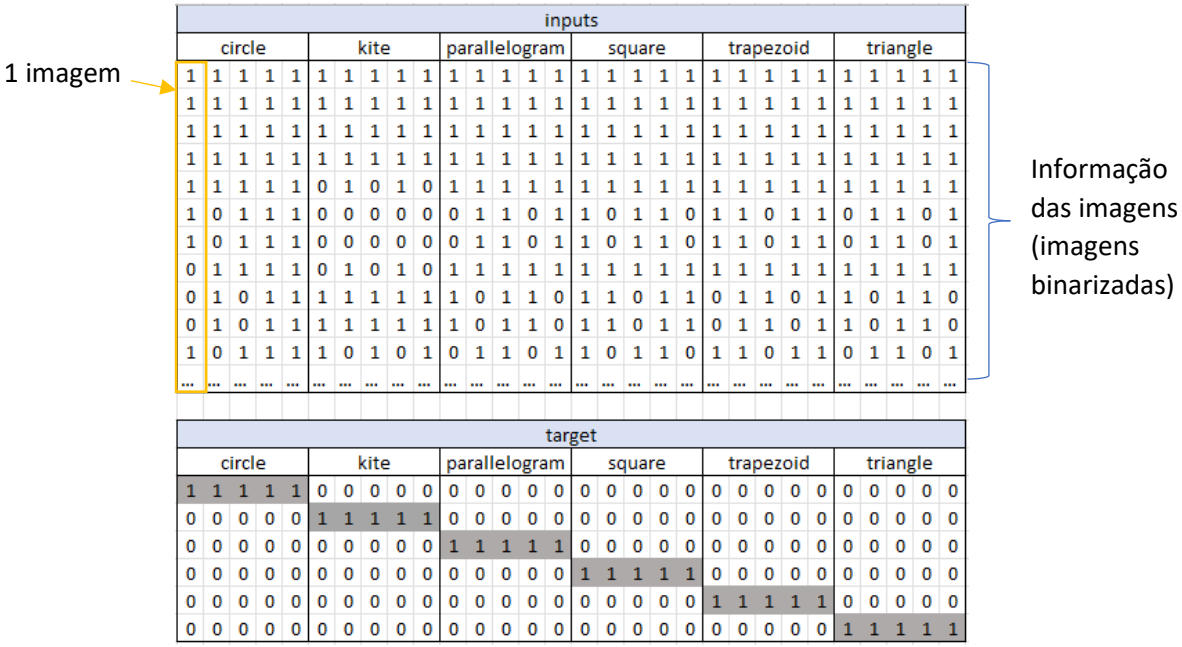


Figura 3 - Ligação entre os inputs e os targets

Para pré-visualizar o código clique aqui!, correspondente ao *trainimages.m*

Nota: O código é igual para todas as pastas, simplesmente é necessário mudar o número targets e a diretoria.

3.2. Preparação da arquitetura da rede neuronal

Depois dos targets e os inputs, só é necessário criar a rede neuronal. Isso consegue-se com a função feedforwardnet().

Depois da criação da rede neuronal, definimos combinações, alterando os parâmetros da rede neuronal, tal como: funções de ativação, funções de saída, função de treino e a divisão dos exemplos.

```
net = feedforwardnet([10 10]); %Criação da rede neuronal
net.layers{1:end-1}.transferFcn = 'logsig'; %função ativação
net.layers{end}.transferFcn = 'tansig'; %função de saida
net.trainFcn = 'trainlm'; %função de treino
net.divideFcn = 'dividerand'; %função de divisão
%parametros da função de divisão
net.divideParam.trainRatio = 0.7;
net.divideParam.valRatio = 0.15;
net.divideParam.testRatio = 0.15;
```

Figura 4 - Configuração da rede neuronal

De seguida depois da rede neuronal estar configurada da forma pretendida, a rede é treinada com a função train(net, inputs, targets), e depois simulada com a função sim(net, inputs).

```
[net,tr] = train(net, inputs, target); %treina a rede
view(net); %Visualizar a rede
out = sim(net, inputs); %simula a rede

plotconfusion(target, out) % Matriz de confusao
```

Figura 5 - Treino e simulação da rede

No fim da simulação, podemos medir a precisão da rede neuronal, se o valor máximo do target e da simulação estão na mesma linha, se estiverem é porque está correto.

```
%Calcula e mostra a percentagem de classificacoes corretas no total dos exemplos
r=0;
for i=1:size(out,2)           % Para cada classificacao
    [a b] = max(out(:,i));    %b guarda a linha onde encontrou valor mais alto da saida obtida
    [c d] = max(target(:,i)); %d guarda a linha onde encontrou valor mais alto da saida desejada
    if b == d                 % se estao na mesma linha, a classificacao foi correta (incrementa 1)
        r = r+1;
    end
end

accuracy = r/size(out,2)*100;
fprintf('epoch: %d Precisao total %f\n', epoch, accuracy);
accuracy_overall_stack = [accuracy_overall_stack accuracy];
```

Figura 6 - Classificação de precisão total dos exemplos

```
TTargets = target(:, tr.testInd);
out = sim(net, TInput);
%Calcula e mostra a percentagem de classificacoes corretas no conjunto de teste
r=0;
for i=1:size(tr.testInd,2)      % Para cada classificacao
    [a b] = max(out(:,i));      % b guarda a linha onde encontrou valor mais alto da saida obtida
    [c d] = max(TTargets(:,i)); % d guarda a linha onde encontrou valor mais alto da saida desejada
    if b == d                   % se estao na mesma linha, a classificacao foi correta (incrementa 1)
        r = r+1;
    end

end
accuracy = r/size(tr.testInd,2)*100;
fprintf('epoch: %d Precisao teste %f\n\n', epoch, accuracy);

accuracy_teste_stack = [accuracy_teste_stack accuracy];
```

Figura 7 - Classificação de precisão de teste

4. Análise de resultados

Para todos os *datasets,* o número de epochs (número de repetições) é 10.

Começando pela pasta "*start*", fizemos 12 combinações inerentes os parâmetros da rede neuronal.

Os resultados dos vários testes podem ser observados no ficheiro Excel (folha 1) junto com este documento ou clicando aqui!

- Verificou-se que a melhor média dos 10 Epochs para a precisão total foi:

  Neurónios: 2 camadas com 10 neurónios cada;
  Função de ativação (camada escondida): *logsig*;
  Função de saída: *tansig*;
  Função de treino: *trainlm*;
  Divisão dos exemplos: *dividerand* = {0.7; 0.15; 0.15}
  Precisão: 76.6%

- Verificou-se que a melhor média dos 10 Epochs para a precisão teste foi:

  Neurónios: 2 camadas com 10 neurónios cada;
  Função de ativação (camada escondida): *logsig*;
  Função de saída: *purelin*;
  Função de treino: *traingd*;
  Divisão dos exemplos: *dividerand* = {0.7; 0.15; 0.15}
  Precisão: 40%

A função de treino *trainbfg*, demora imenso tempo a treinar a rede, não sendo viável, visto que os resultados não são melhores comparando com as funções de treino: *trainlm* e *traingd*.

Na pasta “*train*”, fizemos 16 combinações ligados aos parâmetros da rede neuronal.

Notou-se um melhoramento nos resultados, devido ao *dataset* ser mais complexo, no entanto não acabamos o treino de quatro combinações por demorarem muito, e com muito tempo queremos dizer mais de 7 horas à espera e sem resultados, pelo poder computacional torna-se inviável. Curiosamente trata-se da função de treino *trainbfg*.

Os resultados dos vários testes podem ser observados no ficheiro Excel (folha 1) junto com este documento ou clicando aqui!

- Verificou-se que a melhor média dos 10 Epochs para a precisão total foi:

  Neurónios: 2 camadas com 10 neurónios cada;
  Função de ativação (camada escondida): *logsig*;
  Função de saída: *tansig*;
  Função de treino: *trainlm*;
  Divisão dos exemplos: *dividerand* = {0.7; 0.15; 0.15}
  Precisão: 89.5%

- Verificou-se que a melhor média dos 10 Epochs para a precisão teste foi:

  Neurónios: 2 camadas com 10 neurónios cada;
  Função de ativação (camada escondida): *logsig*;
  Função de saída: *tansig*;
  Função de treino: *trainlm*;
  Divisão dos exemplos: *dividerand* = {0.7; 0.15; 0.15}
  Precisão: 69.11%

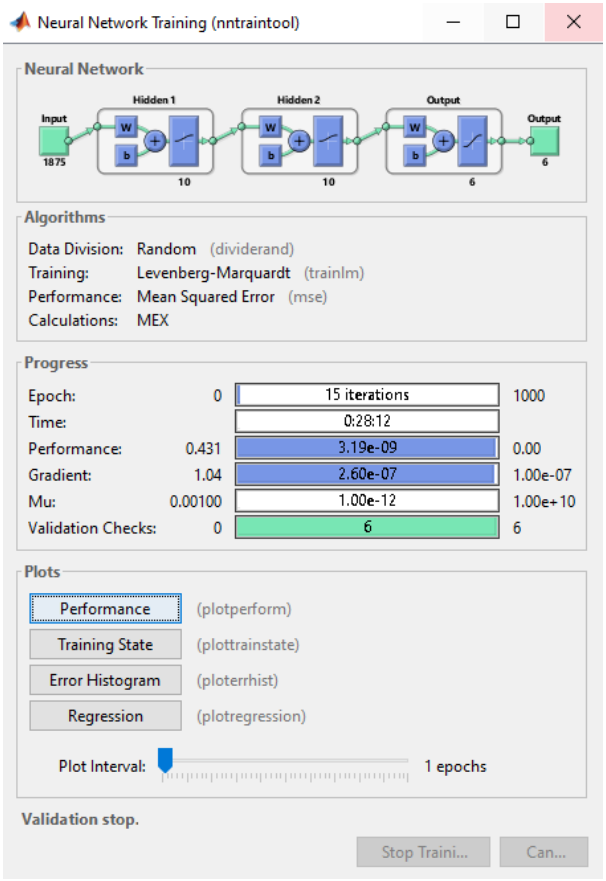


Figura 8 - Melhor rede neuronal

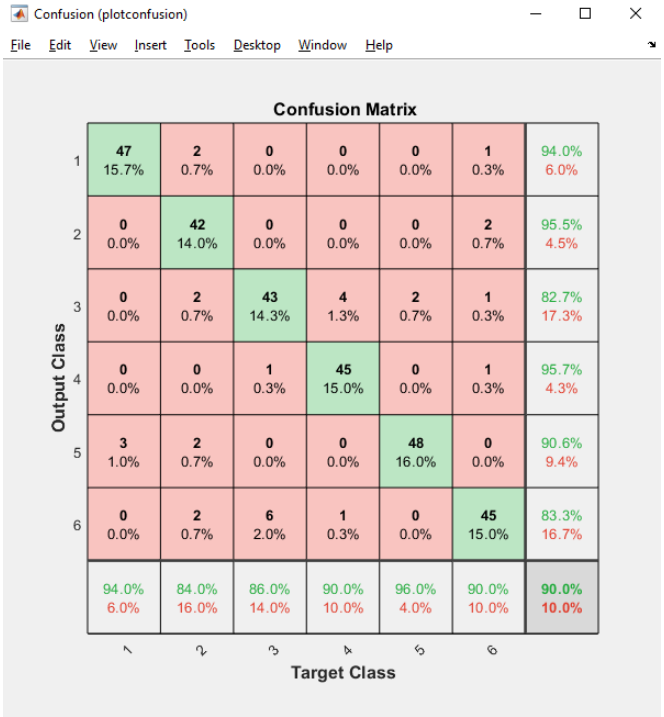


Figura 9 - Matriz confusão

4.1. *Dataset* da pasta "*test*"

Escolhendo a rede com melhores resultados da pasta "*train*", (*Train10.mat*), testámos a pasta "*test*". Os resultados foram relativamente bons. Sendo que a rede conseguiu detetar todos os círculos e quadrados. Os triângulos foram os que tiveram menos performance, acertando em 50% em relação a todos.

Para fazer este teste simplesmente foi só necessário fazer o carregamento da rede neuronal e depois simular com o *dataset* da pasta "*test*".

Para pré-visualizar o código do (testImages.m) clique aqui!

Com o treinamento da pasta "*test*", os resultados para a pasta "*test*" foram relativamente melhores, isto porque estamos a testar com o mesmo *dataset,* com que foi testado. Para as pastas "*start*" e "*train*", os resultados foram um pouco pior.

Por fim, tivemos melhores resultados, na rede neuronal, onde juntamos todas as imagens. Faz sentido porque estamos a aumentar o nosso *dataset*. E por isso os resultados já são consistentes quando tentamos testar em qualquer *dataset*.

Os resultados dos vários testes podem ser observados no ficheiro Excel (folha 2) junto com este documento ou clicando aqui!

5. Desenho de alguma formas geométricas

No programa paint desenhámos manualmente 5 formas geométricas, que foram testadas com a melhor rede neuronal conseguida anteriormente.

Os resultados foram bons, tendo uma precisão de 80%, ou seja, a rede consegui identificar 4/5 formas.

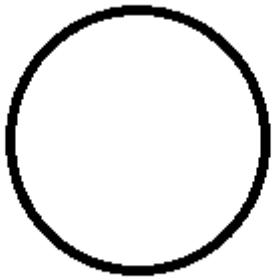


Figura 10 – Círculo

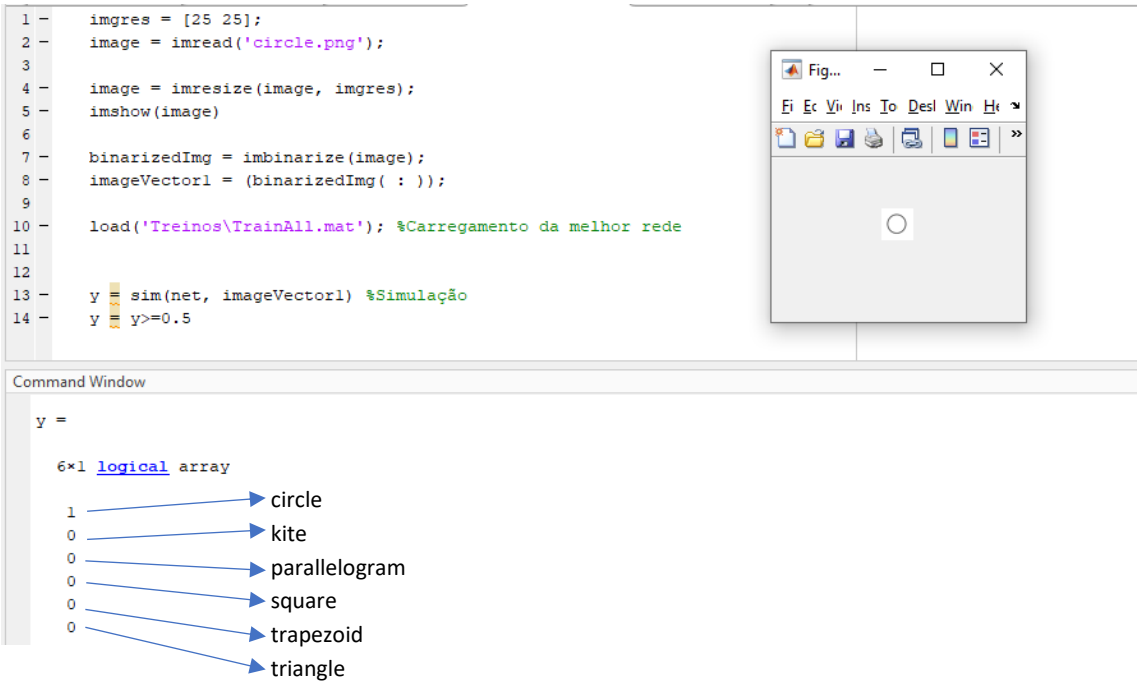
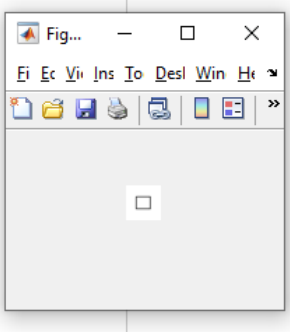


Figura 11 - Resultado do círculo

Figura 12 – Quadrado

```
1 -    imgres = [25 25];
2 -    image = imread('square.png');
3
4 -    image = imresize(image, imgres);
5 -    imshow(image)
6
7 -    binarizedImg = imbinarize(image);
8 -    imageVectorl = (binarizedImg( : ));
9
10 -   load('Treinos\TrainAll.mat'); %Carregamento da melhor rede
11
12
13 -   y = sim(net, imageVectorl) %Simulação
14 -   y = y>=0.5
```



Command Window

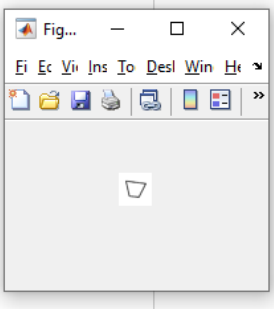
```
y =

  6×1 logical array

   0
   0
   0
   1
   0
   0
```

0 → circle
0 → kite
0 → parallelogram
1 → square
0 → trapezoid
0 → triangle

Figura 13 - Resultado do Quadrado

Figura 14 – Trapézio

```
1 -    imgres = [25 25];
2 -    image = imread('trapezoid.png');
3
4 -    image = imresize(image, imgres);
5 -    imshow(image)
6
7 -    binarizedImg = imbinarize(image);
8 -    imageVector1 = (binarizedImg( : ));
9
10 -   load('Treinos\TrainAll.mat'); %Carregamento da melhor rede
11
12
13 -   y = sim(net, imageVector1) %Simulação
14 -   y = y>=0.5
```



Command Window

```
y =

  6×1 logical array

   0
   0
   0
   0
   1
   0
```

circle
kite
parallelogram
square
trapezoid
triangle

Figura 15 - Resultado do trapézio

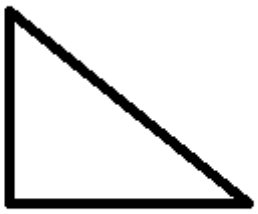
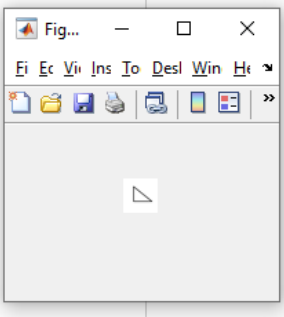


Figura 16 – Triângulo

```
1 -    imgres = [25 25];
2 -    image = imread('triangle.png');
3
4 -    image = imresize(image, imgres);
5 -    imshow(image)
6
7 -    binarizedImg = imbinarize(image);
8 -    imageVector1 = (binarizedImg( : ));
9
10 -   load('Treinos\TrainAll.mat'); %Carregamento da melhor rede
11
12
13 -   y = sim(net, imageVector1) %Simulação
14 -   y = y>=0.5
```



Command Window

```
y =

  6×1 logical array

   0
   0
   0
   0
   0
   1
```

circle
kite
parallelogram
square
trapezoid
triangle

Figura 17 - Resultado do triângulo

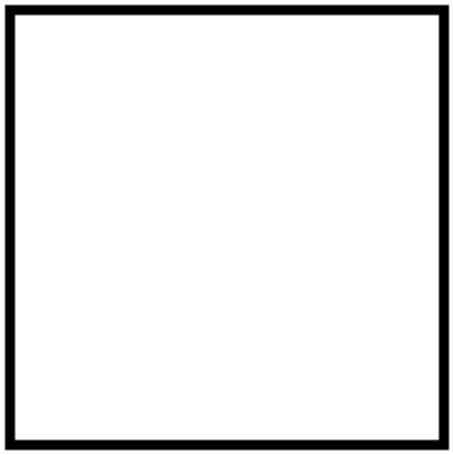
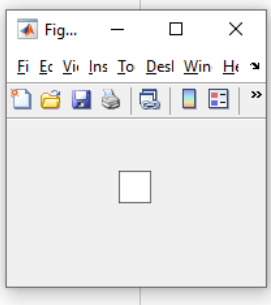


Figura 18 - Quadrado grande

```
imgres = [25 25];
image = imread('largeSquare.png');

image = imresize(image, imgres);
imshow(image)

binarizedImg = imbinarize(image);
imageVector1 = (binarizedImg( : ));

load('Treinos\TrainAll.mat'); %Carregamento da melhor rede


y = sim(net, imageVector1) %Simulação
y = y>=0.5
```



Command Window

```
y =

  6×1 logical array

   0
   0
   0
   0
   0
   1
```

circle
kite
parallelogram
square
trapezoid
triangle

Figura 19 - Resultado do quadrado grande

6. Aplicação gráfica

A aplicação gráfica, permite de uma forma fácil e intuitiva:

-Classificar imagens dentro das seis formas geométricas;

-Configurar a topologia da rede neuronal, escolher funções de treino/ativação;

-Treinar a rede neuronal;

-Gravar uma rede neuronal que foi treinada;

-Permite o carregamento uma rede neuronal a aplicar a imagens;

-Visualização de resultados.

6.1. Guião

6.1.1. Separador Classifica

O separador Classifica, como o próprio nome indica, permite classificar uma imagem mediante as 6 formas geométricas disponíveis.

É dada opção ao utilizador de poder classificar a imagem com uma rede predefinida (melhor rede obtida nos testes anteriores), ou se quiser pode carregar uma rede neuronal externa para classificar a imagem.

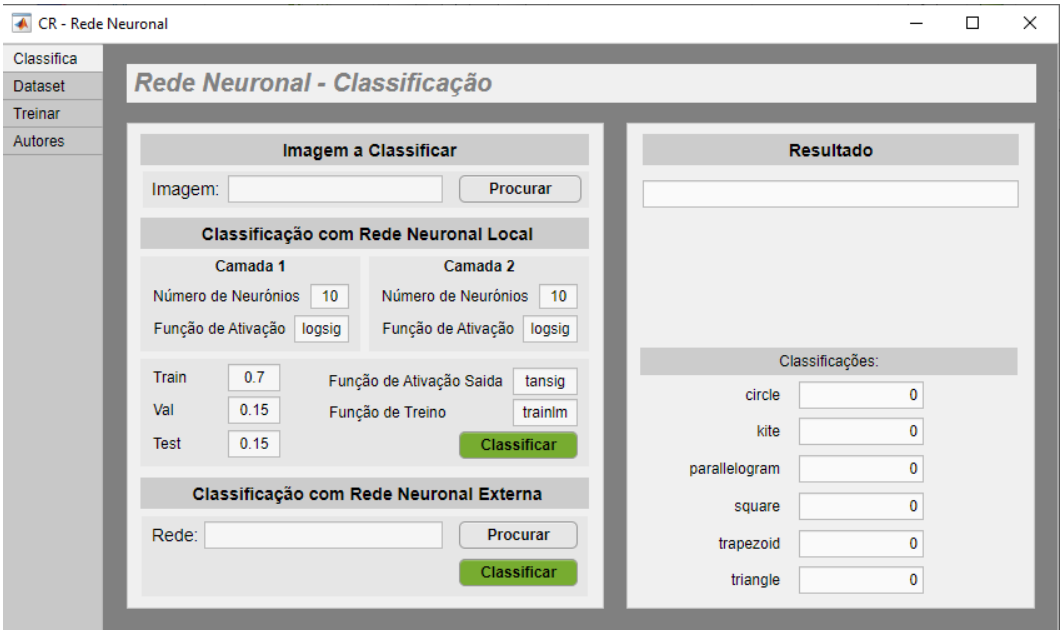


Figura 20 - Layout do separador Classifica

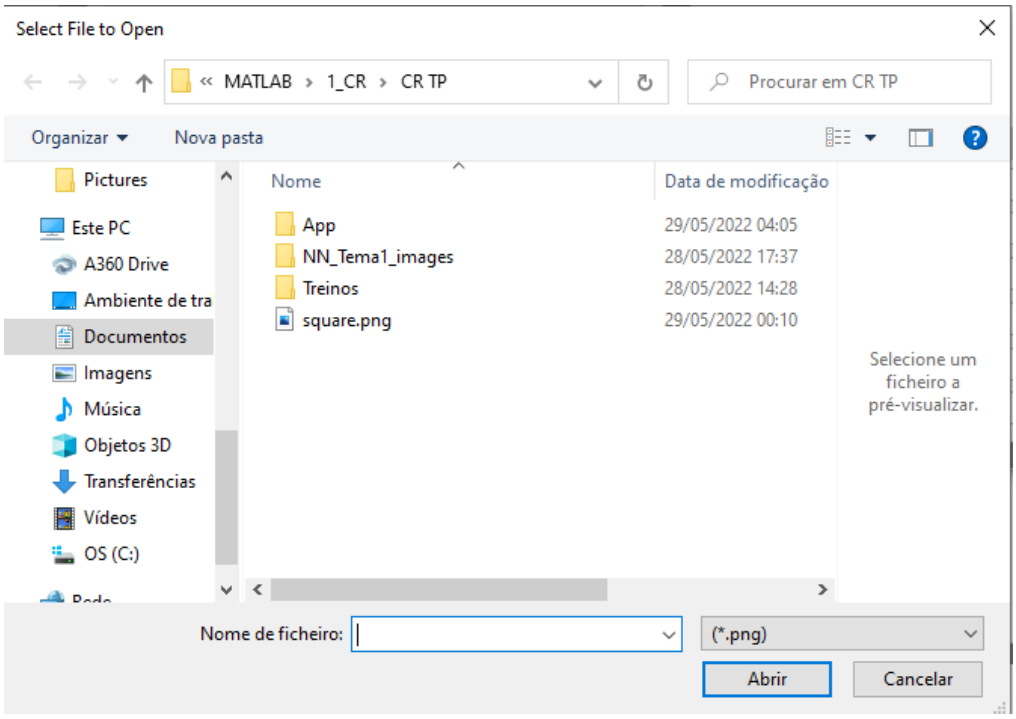


Figura 21 - Escolha de imagem

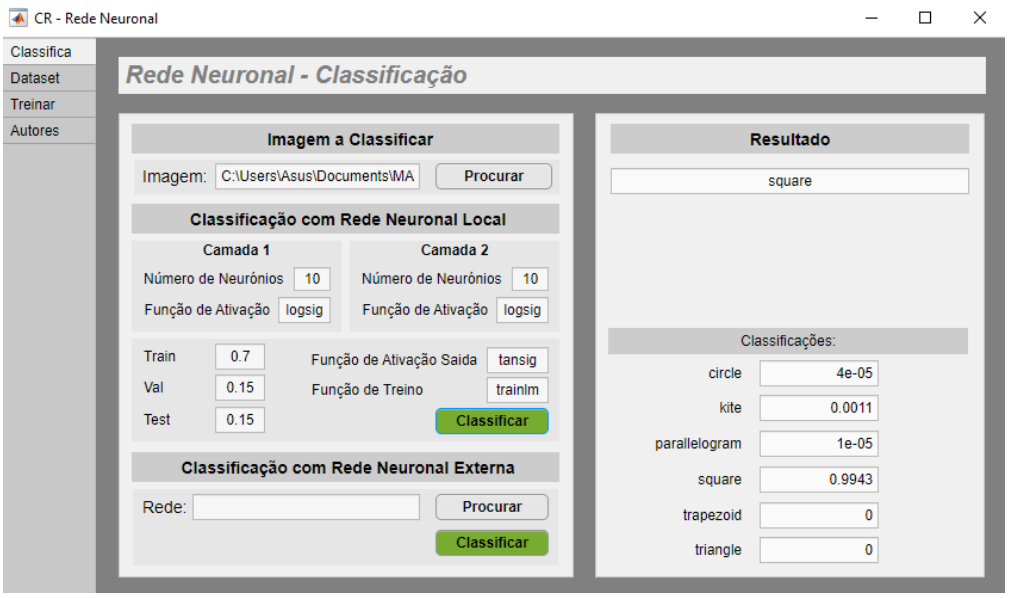


Figura 22 - Resultado com rede neuronal local

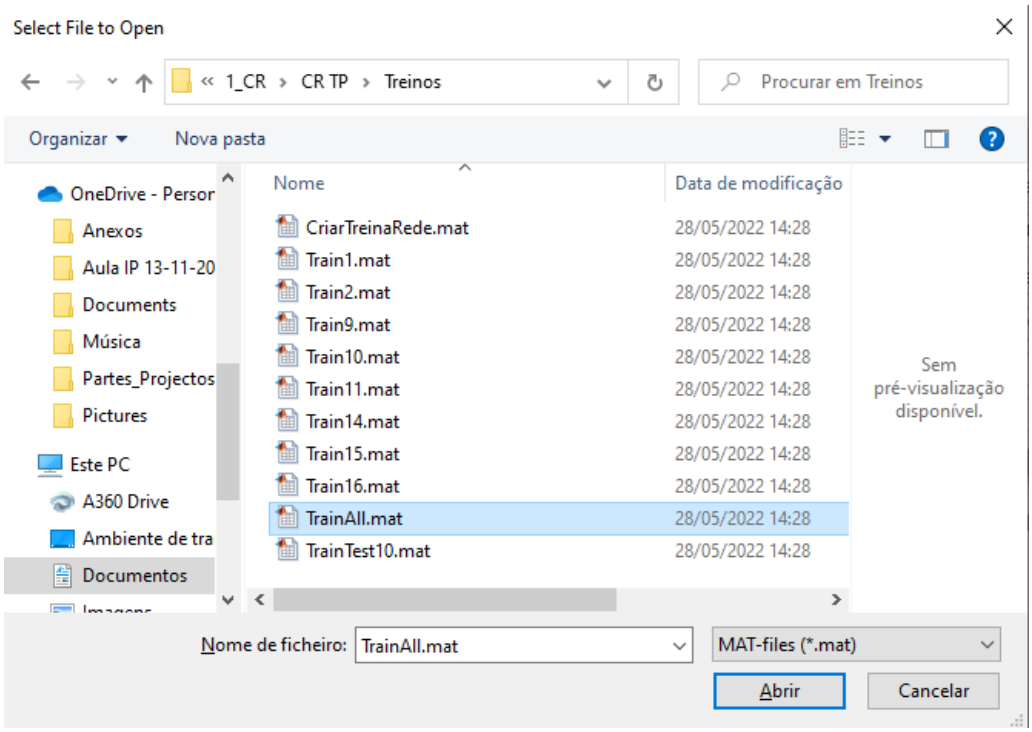


Figura 23 - Escolha de outra rede neuronal

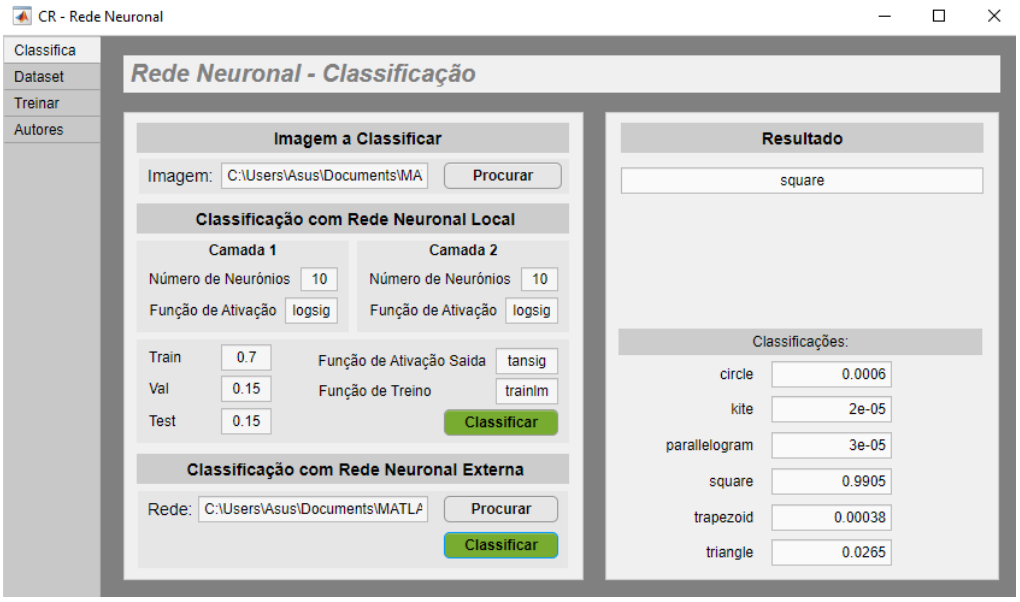


Figura 24 - Resultado com rede neuronal externa

É possível observar os resultados no lado direito da janela e as classificações para cada forma geométrica

6.1.2. Separador Dataset

O separador Dataset, permite medir a precisão de um *dataset*.

É dada opção ao utilizador de poder classificar a imagem com uma rede predefinida (melhor rede obtida nos testes anteriores), ou se quiser pode carregar uma rede neuronal externa para medir a precisão do *dataset*.

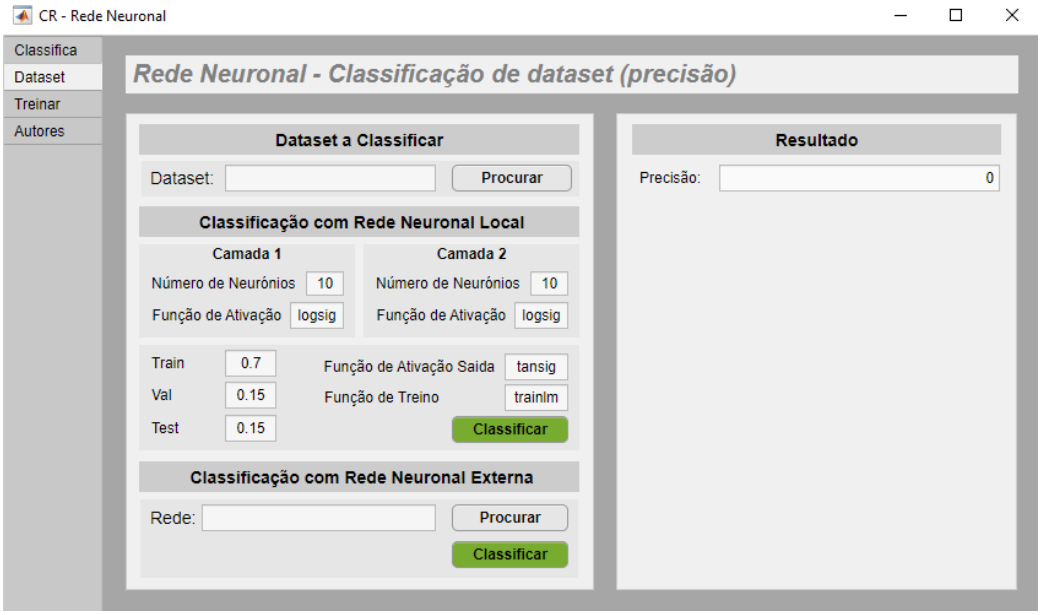


Figura 25 - Layout do separador Dataset

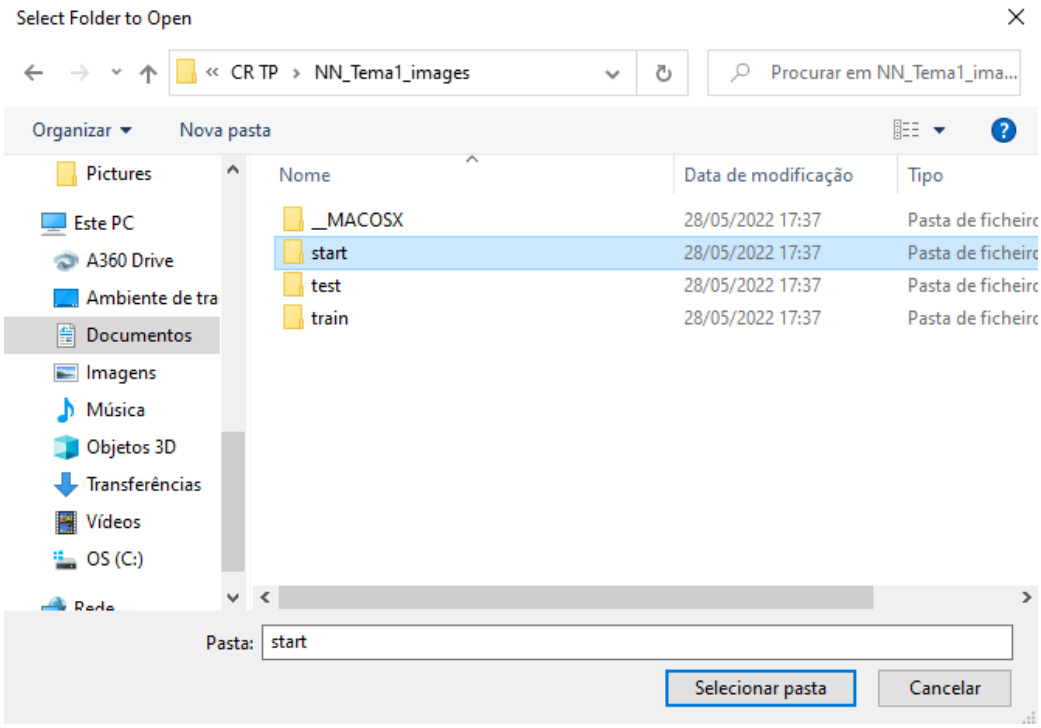


Figura 26 - Escolha do Dataset

Figura 27 - Resultado de precisão

6.1.3. Separador Treinar

O separador Treinar, permite treinar um dos três *datasets,* (start, test, train).

O utilizador pode alterar os variadíssimos parâmetros da rede neuronal, como o número de camadas, número de neurónios de cada camada e as suas funções de ativação, funções de saída e de treino e por fim os valores de divisão.

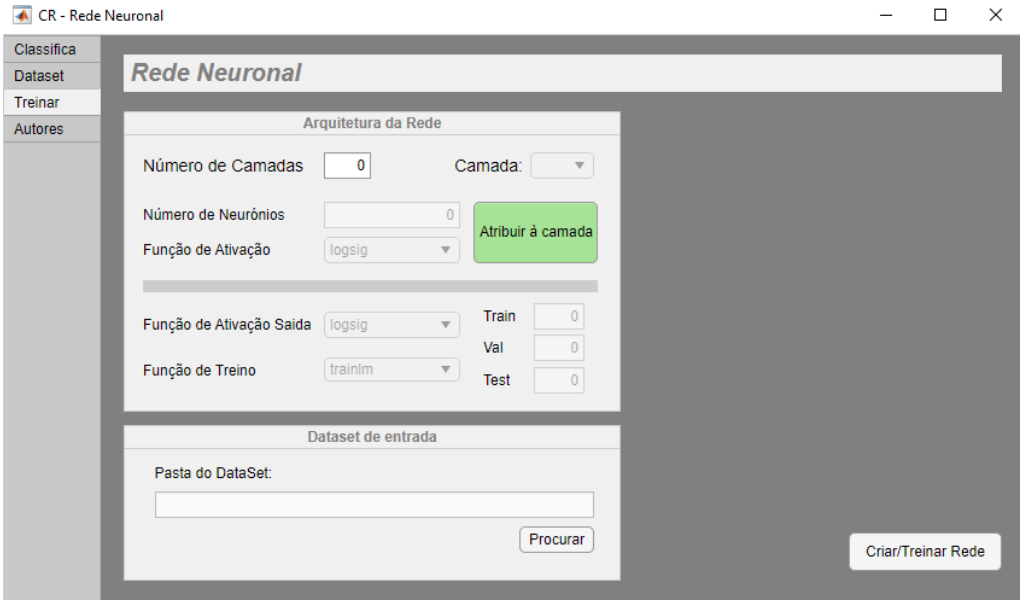


Figura 28 - Layout do separador Treinar

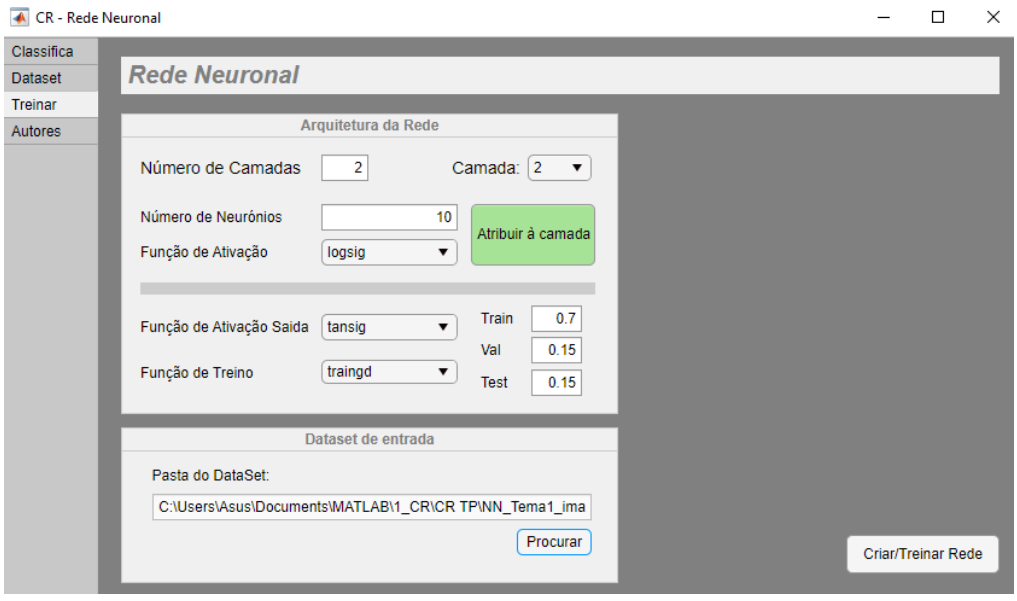


Figura 29 - Configuração da arquitetura da rede

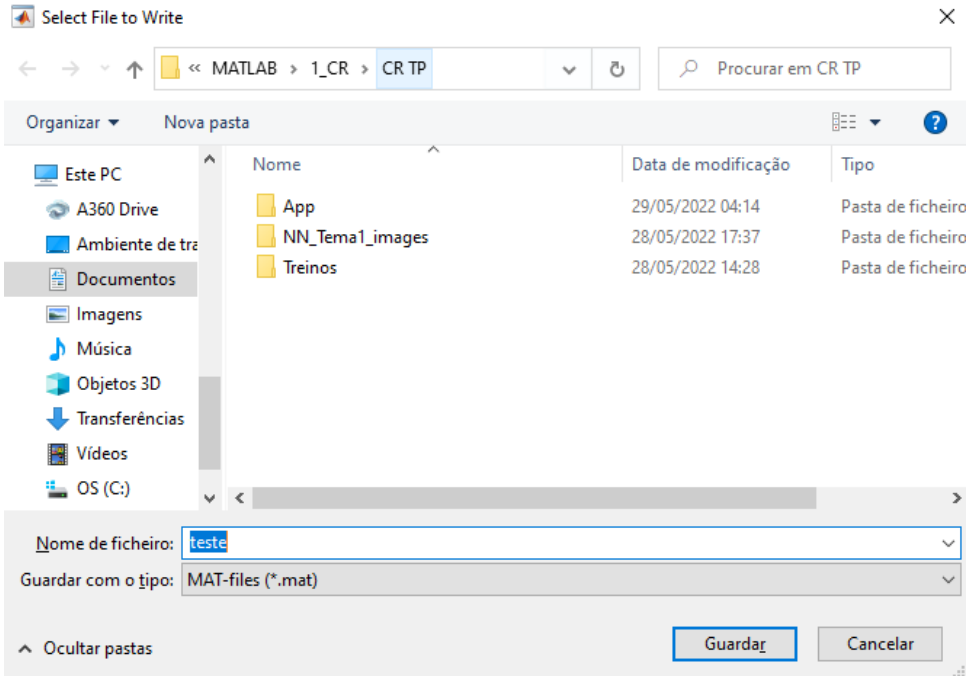


Figura 30 - Escolha do Dataset

6.1.4. Separador Autores

Figura 31 - Layout do separador Autores

7. Conclusão

Em suma, conseguimos perceber que quanto maior for o *dataset*, menor será o erro e vice-versa. Algumas funções são demasiadas complexas para o nosso poder computacional, tal como se pode verificar com a função de treino *trainbfg.*

Dá para notar que o treinamento, baseia-se muito na posição pixéis, isto porque as imagens do *dataset*, estão muito centradas. Por exemplo se desenharmos um quadrado muito próximo das bordas da imagem, é certo que não vai conseguir detetar a forma geométrica, isto porque os quadrados com que a rede foi treinada estão mais próximos ao centro.

Ainda se verificou, que os treinos mais rápidos acabam por ter resultados melhores, o que acaba por ser um pouco estranho.

Mas no fundo conseguimos, resultados bastante razoáveis, tendo em conta a natureza da rede neuronal.

Com o último treino onde, se reuniu todas as imagens das três pastas, é possível já ter resultados com precisão maiores que 80 por cento.