Conhecimento e Raciocínio

Relatório Trabalho Prático: Redes Neuronais

Filipe Fernandes

a2020134826@isec.pt

Marco Daniel Neto Pereira

a2020133341@isec.pt

Instituto Superior de Engenharia de Coimbra

Engenharia Informática 2022/2023

1 ÍNDICE

1 Índice *2*
1 Introdução *3*
2 Análise e tratamento de imagem *4*
2.1 Pré processamento das imagens 4
2.2 Matriz de treinamento 4
2.3 Targets 4
3 Testes *5*
3.1 Alínea A 5
3.2 Alínea B 7
3.2.1 Rede *feedfoward* – Dígitos 9
3.2.2 Rede *feedfoward* – Operadores 11
3.3 Alínea C 12
3.3.1 Dataset 12
3.3.2 Testes 12
3.3.3 Conclusões 13
3.4 Alínea D 13
3.4.1 GUI 13
3.4.2 Implementação 14
4 Conclusão *18*
5 Bibliografia *18*

1 INTRODUÇÃO

Este trabalho foi realizado no âmbito da disciplina da Unidade Curricular de Conhecimento e Raciocínio, tem por objetivo o desenvolvimento de um projeto capaz de classificar corretamente 10 dígitos (0 a 9) e 4 operadores matemáticos (+. -, × , :), utilizando diferentes arquiteturas de redes neuronais do tipo ***feedforward***.

2 ANÁLISE E TRATAMENTO DE IMAGEM

2.1 PRÉ PROCESSAMENTO DAS IMAGENS

Para uma otimização da aplicação foi realizado um **redimensionamento** das imagens para a resolução de 25x25 com recurso a funções da *toolbox image processing* do Matlab.

Ao longo do processo de pré processamento de imagens, as mesma foram transformadas em uma matriz binária e passadas para uma matriz com os dados de treinamento.

```
% Define o caminho para a imagem atual
current_image = imread(strcat(current_folder, num2str(j), '.png'));

% Redimensiona a matriz binária para a dimensão das imagens
resized_image = imresize(current_image, image_size);

% Adiciona a matriz redimensionada à matriz de dados de treinamento
training_data((i*5)+j, :) = resized_image(:);
```

Figura 1 - Pré processamento de imagens (código)

2.2 MATRIZ DE TREINAMENTO

A matriz de treinamento é constituída pelo número de imagens e o seu respetivo tamanho.

Na Figura 2 podemos ver que, no caso da alínea a), a matriz é constituída por 70 linhas, que corresponde às 70 imagens presentes no dataset (14 classes x 5 imagens cada) e 625 colunas que representam o tamanho de cada imagem (25x25).

```
training_data = zeros(num_classes*5, image_size(1)*image_size(2)); % [70 , 625]
```

Figura 2 - Matriz de treinamento (código)

2.3 TARGETS

A matriz de targets (Figura 3), numa fase inicial é constituída por 70 linhas, que representam o número de imagens existentes no dataset (14 classes x 5 imagens cada) e 14 colunas que representam o número de classes existentes no dataset.

```
training_labels = zeros(num_classes*5, num_classes);  % Target [70  , 14]
```

Figura 3 - Targets (código)

3 TESTES

3.1 ALÍNEA A

Foi usada uma rede neuronal do tipo ***feedfoward*** com uma camada com 10 neurónios, com a função de treinamento ***'traingdx'***, com o parâmetro ***'net.divideFcn'*** vazio, usando então, todos os dados para o treinamento da rede, o número de épocas definido foi 2500 (Figura 4).

```
% Cria a rede neural
net = feedforwardnet(10);

% Define as opções de treinamento
net.trainFcn = 'traingdx';
net.divideFcn = '';
net.trainParam.epochs = 2500;
```

Figura 4 – Arquitetura da Rede Neuronal (código)

Na Figura 5, podemos observar os **resultados** da precisão **global** obtidos ao longo de 10 execuções, bem como a **média** da precisão global e o ***plotconfusion*** (Figura 5 e Figura 6).

Resultados:

```
>> Fase1V3
[1]->[98.571429%]
[2]->[97.142857%]
[3]->[100.000000%]
[4]->[100.000000%]
[5]->[100.000000%]
[6]->[100.000000%]
[7]->[100.000000%]
[8]->[100.000000%]
[9]->[98.571429%]
[10]->[94.285714%]
Média da precisão total (10 execuções) nos 70 exemplos: 98.9 %
```

Figura 5 - Fase1V3 resultados

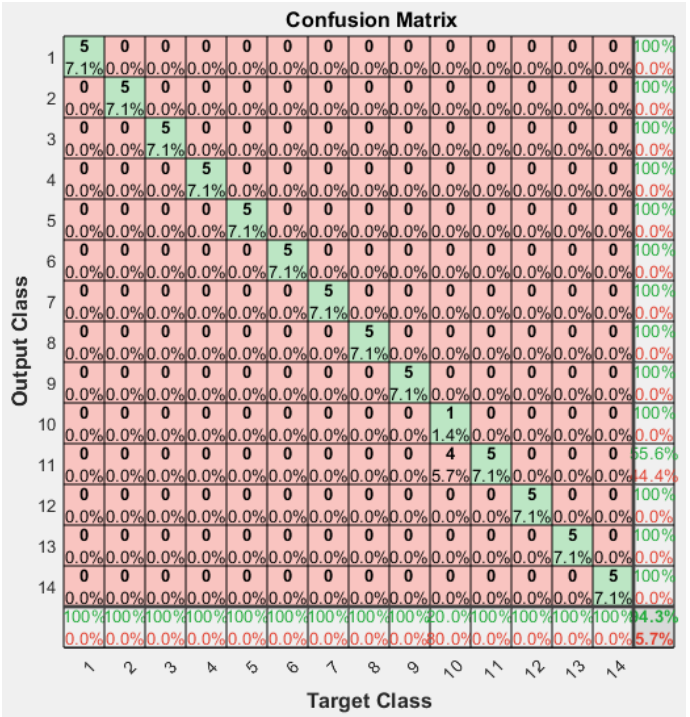

Figura 6 - Plotconfusion

3.2 ALÍNEA B

Para tentar obter os melhores resultados na classificação das imagens fornecidas, foram testadas várias **funções** e **parâmetros** diferentes. Podendo assim comparar com a configuração inicial.

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos		Precisão Global	Precisão Teste
O número e dimensão das camadas encondidas influencia o desempenho?								
Conf1	4	100 50 30 10	tansig, tansig	traingdx	dividerand = {0.7, 0, 0.3}		90,20%	50,20%
Conf2	4	300 150 75 40	tansig, tansig	traingdx	dividerand = {0.7, 0, 0.3}		94,00%	72,50%
Conf3	5	200 100 50 30 10	tansig, tansig	traingdx	dividerand = {0.7, 0, 0.3}		89,70%	49,20%
Conf4	3	200 100 50	tansig, tansig	traingdx	dividerand = {0.7, 0, 0.3}		93,90%	69,80%

Com a realização dos primeiros testes, foi possível concluir que a alteração da quantidade de camadas **afeta relativamente** a precisão global e a de teste, percebemos também que o aumento do número e dimensão de camadas **não** está diretamente ligado com uma boa ***performance*** da rede neuronal. Um parâmetro útil poderia ser o **tempo** decorrido para a conclusão dos testes, quanto maior o número de camadas e neurônios, maior o tempo de conclusão dos treinamentos e testes das redes.

A função de treino influencia o desempenho?								
Conf1	4	200 100 50 30	tansig, tansig	traingd	dividerand = {0.7, 0, 0.3}		23,70%	8,80%
Conf2	4	200 100 50 30	tansig, tansig	trains	dividerand = {0.7, 0, 0.3}		75,80%	27,70%
Conf3	4	200 100 50 30	tansig, tansig	trainoss	dividerand = {0.7, 0, 0.3}		92,00%	71,00%
Conf4	4	200 100 50 30	tansig, tansig	traingdm	dividerand = {0.7, 0, 0.3}		19,40%	9,80%

Após a testagem da rede com diversas funções de treino, podemos concluir que as mesmas **influenciam** de uma forma bastante **acentuada** a ***performance***, tanto a nível de precisão global, como a nível de precisão de teste.

As funções de ativação influenciam o desempenho?							
Conf1	4	200 100 50 30	logsig, purelin	traingdx	dividerand = {0.7, 0, 0.3}	64,20%	44,70%
Conf2	4	200 100 50 30	purelin, logsig	traingdx	dividerand = {0.7, 0, 0.3}	83,90%	41,50%
Conf3	4	200 100 50 30	tansig, logsig	traingdx	dividerand = {0.7, 0, 0.3}	94,20%	72,80%
Conf4	4	200 100 50 30	satlin, tribas	traingdx	dividerand = {0.7, 0, 0.3}	91,10%	59,00%
Conf5	4	200 100 50 30	tansig, tansig,tansig	traingdx	dividerand = {0.7, 0, 0.3}	93,70%	72,20%
Conf6	4	200 100 50 30	logsig, satlin,purelin	traingdx	dividerand = {0.7, 0, 0.3}	64,60%	17,60%

Após a testagem da rede com diversas funções de ativação, podemos concluir que as mesmas influenciam de uma forma evidente a ***performance***. Percebemos que o uso das funções de ativação ***'tansig'*** e ***'logsig'*** aumentam consideravelmente o desempenho da rede. Por outro lado, as funções ***'satlin'*** e ***'logsig'*** (caso seja usada na primeira camada), diminuem o desempenho da rede neuronal.

A divisão de exemplos pelos conjuntos influencia o desempenho?							
Conf1	4	200 100 50 30	tansig, tansig	traingdx	dividerand = {0.33, 0.33, 0.33}	61,80%	22,50%
Conf2	4	200 100 50 30	tansig, tansig	traingdx	dividerand = {0.9, 0.05, 0.05}	75,10%	24,90%
Conf3	4	200 100 50 30	tansig, tansig	traingdx	dividerand = {0.5, 0.25, 0.25}	67,60%	20,30%
Conf4	4	200 100 50 30	tansig, tansig	traingdx	dividerand = {0.1, 0.45, 0.45}	32,70%	11,80%
Conf5	4	200 100 50 30	tansig, tansig	traingdx	dividerand = {0.7, 0.3, 0}	67,60%	NaN %
Conf6	4	200 100 50 30	tansig, tansig	traingdx	dividerand = {0.9, 0.05, 0.05}	72,00%	22,80%

Com a realização de vários testes, foi possível concluir que a divisão de exemplos tem um **papel fundamental** no que diz respeito à ***performance*** das redes neuronais.

3.2.1 Rede *feedfoward* – Dígitos

Com os testes realizados anteriormente, percebemos quais eram os parâmetros na arquitetura de uma rede neuronal *feedfoward* que ao serem alterados, realmente poderiam fazer a diferença para alcançar um bom desempenho.

A rede com uma arquitetura apenas para classificar dígitos, é do tipo ***feedfoward*** constituída por **7** camadas escondidas, com a função de treino ***'traingdx'*** e **6** funções de ativação (Figura 7).

```
% Cria a rede neural
net = feedforwardnet([400 300 200 100 50 30 10]);

% Define as opções de treinamento
net.trainFcn = 'traingdx';
net.layers{1}.transferFcn = 'tansig';
net.layers{2}.transferFcn = 'logsig';
net.layers{3}.transferFcn = 'tansig';
net.layers{4}.transferFcn = 'tansig';
net.layers{5}.transferFcn = 'tansig';
net.layers{6}.transferFcn = 'tansig';
```

Figura 7 - Rede e Parâmetros de Treinamento (código)

A função de divisão utilizada é ***'dividerand'*** e a divisão dos exemplos é pode ser observada na Figura 8.

```
net.divideFcn = 'dividerand';
net.divideParam.trainRatio = .7;
net.divideParam.valRatio = 0.0;
net.divideParam.testRatio = 0.3;
```

Figura 8 - Função e divisão de exemplos (código)

Foram implementados (Figura 9) também outros parâmetros que resultaram numas melhoras dos valores de precisão da rede. O primeiro foi o ***'net.trainParam.lr'*** que se refere ao **coeficiente de aprendizagem**, este parâmetro determina o tamanho dos passos que os parâmetros da rede neuronal (pesos e viés) são atualizados durante o processo de treinamento. E por fim, foi adicionado o parâmetro ***'net.performParam.regularization'*** que se refere ao parâmetro de **regularização**, com o objetivo de evitar o ***overfitting*** e melhorar a capacidade de generalização do modelo.

```
net.trainParam.lr = 0.01;
net.performParam.regularization = 0.001;
```

Figura 9 - Coeficiente de aprendizagem e regularização (código)

Esta rede neuronal apresentou uma **precisão global** de **94.6%** e uma **precisão de teste** de **76.7%** (Figura 10).

```
Precisao total 94.6 %
Precisao teste 76.7 %
```

Figura 10 - Precisão global e de teste

3.2.2 Rede *feedfoward* – Operadores

A rede com uma arquitetura apenas para classificar operadores, é do tipo ***feedfoward*** constituída por **5** camadas escondidas, com a função de treino ***'traingdx'*** e **4** funções de ativação (Figura 11).

```
% Cria a rede neural
net = feedforwardnet([300 200 100 50 30]);


% Define as opções de treinamento
net.trainFcn = 'traingdx';
net.layers{1}.transferFcn = 'tansig';
net.layers{2}.transferFcn = 'logsig';
net.layers{3}.transferFcn = 'tansig';
net.layers{4}.transferFcn = 'tansig';
```

Figura 11 - Rede e Parâmetros de Treinamento (código)

A função de divisão utilizada é ***'dividerand'*** e a divisão dos exemplos pode ser observada na Figura 12.

```
net.divideFcn = 'dividerand';
net.divideParam.trainRatio = .7;
net.divideParam.valRatio = 0.0;
net.divideParam.testRatio = 0.3;
```

Figura 12 - Função e divisão de exemplos (código)

Por fim, foi adicionado o parâmetro ***'net.trainParam.lr'*** (Figura 13).

```
net.trainParam.lr = 0.001;
```

Figura 13 - Coeficiente de aprendizagem (código)

Esta rede neuronal apresentou uma **precisão global** de **95.5%** e uma **precisão de teste** de **81.7%** (Figura 14).

```
Precisao total 95.5 %
Precisao teste 81.7 %
```

Figura 14 - Precisão global e de teste

3.3 ALÍNEA C

3.3.1 Dataset

Após a construção do ***dataset*** com desenhos de dígitos/símbolos feitos manualmente, na Figura 15 podemos ver uma pequena amostra de imagens presentes no *dataset*.

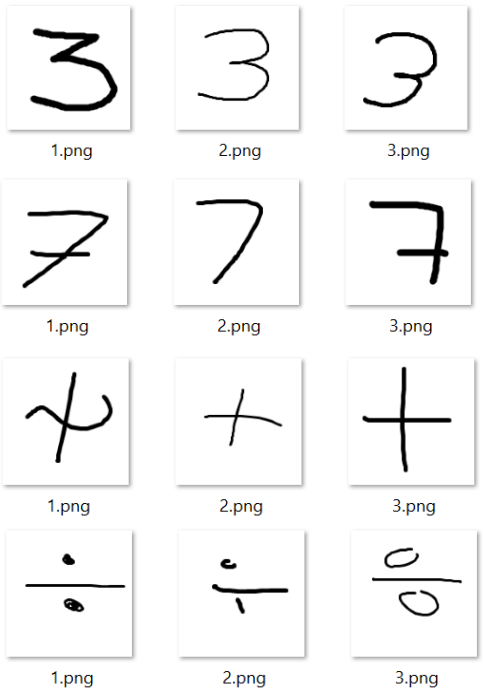

Figura 15 - Números(3 e 7) , Operadores (Adição e Divisão)

3.3.2 Testes

Passando agora para a fase de testes das redes neuronais criadas anteriormente, a rede neuronal responsável pela classificação dos digitos (**digitos.m**), obteve uma **precisão total** de **36.7%** (Figura 16), já a rede neuronal responsável pela classificação de operadores (**operadores.m**), obteve uma **precisão total** de **50%** (Figura 17).

```
Precisao total 36.7 %
```

Figura 16 - Precisão total (digitos.m)

```
Precisao total 50.0 %
```

Figura 17 - Precisão total (operadores.m)

3.3.3 Conclusões

Através dos resultados obtidos na fase de testes das redes neuronais, percebemos que apesar de terem sido obtidos bons resultados na fase de treinamento (Figura 10 e Figura 14), as mesmas não conseguiram desempenhar um bom trabalho em classificar o dataset criado manualmente. Estes resultados (Figura 16 e Figura 17) podem derivar das imagens criadas, uma vez que existem algumas imagens que apresentam diferenças em relação ao dataset de treinamento.

3.4 ALÍNEA D

3.4.1 GUI

A **interface** da aplicação é uma interface **simples**, onde o User tem a possibilidade de **desenhar** dígitos/operadores matemáticos, **escolher** a rede neuronal que vai classificar a imagem desenhada e por fim desencadear o processo de **classificação** de imagem através do botão **'Classificar'**, o *container* ´**Resultado'** apresenta o número/operador que a rede classificou a partir da imagem desenhada pelo User.

Na Figura 18 podemos observar a ***Graphical User Interface*** implementada para esta aplicação.

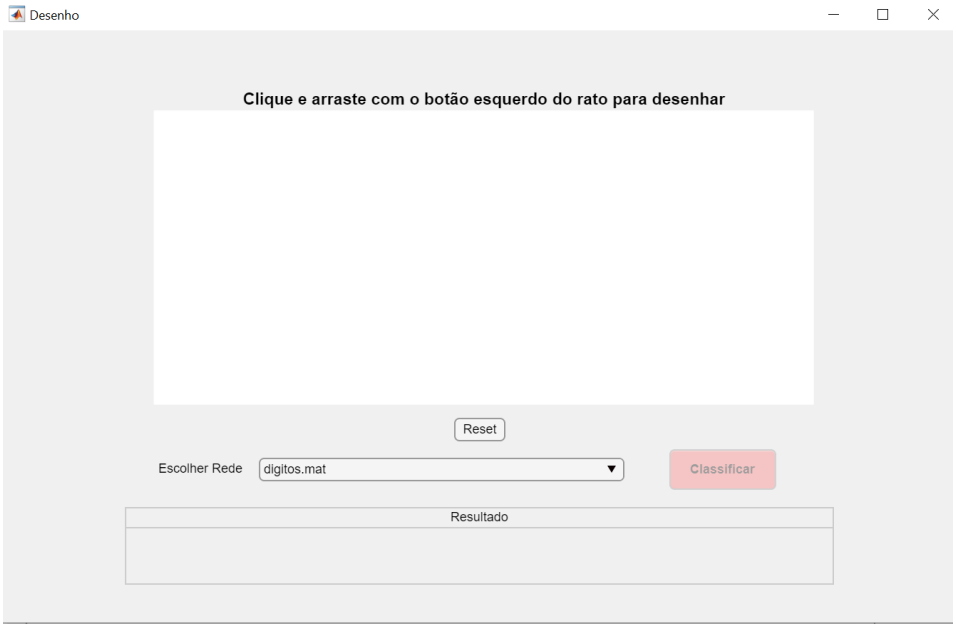

Figura 18 - App (GUI)

3.4.2 Implementação

Nesta parte vamos falar de alguma da **lógica** por detrás do processo de **classificação** de imagens desta aplicação.

3.4.2.1 Armazenamento de coordenadas

A função **'*UIAxesButtonDown()*'** desempenha um papel importante nesta aplicação pois é **responsável** por **captar**, **desenhar** e **guardar** a imagem desenhada pelo User.

Na Figura 19, podemos observar o código implementado responsável por **armazenar** as **coordenadas** do desenho do User e apresentar esse mesmo desenho em tempo real, para que o User consiga saber o que está a desenhar.

```
function UIAxesButtonDown(app, event)
    hFH = drawfreehand(app.UIAxes); % Cria um objeto "freehand" na área 'app.UIAxes'

    % Obtem as coordenadas xy onde o User desenhou
    xy = hFH.Position;

    % Remover o objeto hFH pois as coordenadas já foram guardadas em 'xy'
    delete(hFH);
    hold(app.UIAxes, 'on');
    xCoordinates = xy(:, 1); % Extrair as coordenadas x do desenho da matriz 'xy'
    yCoordinates = xy(:, 2); % Extrair as coordenadas y do desenho da matriz 'xy'

    line(app.UIAxes, xCoordinates, yCoordinates, 'Color', 'red', 'LineWidth', 5); % Apresenta o desenho em tempo real
```

Figura 19 - Armazenamento das coordenadas x e y, plotagem do desenho (código)

3.4.2.2 Construção de imagem

Na Figura 20, podemos observar o código responsável por construir/desenhar e espessar as linhas defenidas pelas coordenadas **'*xCoordinates*'** e **'*yCoordinates*'** na imagem **'*app.grayImage*'**.

```
% Desenhar as linhas definidas pelas coordenadas 'xCoordinates' e 'yCoordinates'
for k = 1 : length(xCoordinates)-1
    x1 = round(xCoordinates(k));
    y1 = round(yCoordinates(k));
    x2 = round(xCoordinates(k+1));
    y2 = round(yCoordinates(k+1));

    % Desenhar linhas -> 'app.grayImage'
    x = linspace(x1, x2, max(abs(x2 - x1), abs(y2 - y1))+1);
    y = linspace(y1, y2, max(abs(x2 - x1), abs(y2 - y1))+1);

    for i = 1:length(x)
        app.grayImage(round(y(i)), round(x(i))) = 0;
    end
end

% Aumentar a espessura dos contornos das linhas
contornos = bwperim(app.grayImage);
espessura = 4; % Espessura desejada dos contornos
elementoEstruturante = strel('disk', espessura);
contornosEspessos = imdilate(contornos, elementoEstruturante);

% Desenhar os contornos espessos na imagem
app.grayImage(contornosEspessos) = 0;
```

Figura 20 - Desenhar e espessar linhas (código)

3.4.2.3 Tratamento e armazenamento de imagem

Na Figura 21, podemos observar o processo de **tratamento** e **armazenamento** de imagem. Foi usada a função **'*flipud*'**, esta função inverte as linhas da matriz verticalmente da imagem **'*app.grayImage*'** e é útil uma vez que a matriz irá ser alterada para se adequar a uma convenção específica, que neste caso é o redimensionamento de imagem (25x25) que será feito posteriormente e permite também que a imagem seja exibida corretamente, uma vez que será guardada em formato **'*PNG*'**.

```
app.grayImage = flipud(app.grayImage); % Inverter a imagem

% Salvar a imagem como um arquivo PNG
imwrite(app.grayImage,'D:/Faculdade/2º-3ºAno/2º Semestre/CR/Projeto/imagem.png');
```

Figura 21 - Tratamento e armazenamento de imagem (código)

3.4.2.4 Processamento e classificação de Imagem

Na função **'*ClassificarButtonPushed()*' ,** que é desencadeada pelo User assim que o mesmo carrega no botão **'Classificar'** (Figura 18), ocorrem os procedimentos de **processamento** e **classificação** de Imagem, através da função **'*identificarImagem()*'**.

No inicio da função **'*ClassificarButtonPushed()*'**, são declaradas quatro variáveis do tipo **persistent**, que permitem manter o mesmo valor ao logo de várias execuções da função. Este tipo de variáveis são úteis pois permitem guardar os valores referentes às classificações feitas pelas redes neuronais, que depois são mostrados no **container '*Resultado*'** (Figura 22).

```
app.Resultado.Text = sprintf("[%d] %d %s %d = %d",i+1,firstNumb,op,secondNumb,resultado);
```

Figura 22 - Output de valores (código)

Na Figura 23, podemos observar a declaração das variáveis ***persistent*** bem como a chamada da função **'*identificarImagem()*'** à medida que o User vai desenhando a **expressão matemática**.

```
function ClassificarButtonPushed(app, event)
    persistent i;
    persistent firstNumb;
    persistent op;
    persistent secondNumb;
    resultado='';
    if isempty(i)
        i=0;
    else
        i=i+1;
    end

    if(app.EscolherRede.Value ~= "")
       if(i==0)
           [~,firstNumb]=identificarImagem(app.EscolherRede.Value,0);
           app.Resultado.Text = sprintf("[%d] %d",i+1,firstNumb);
       elseif(i==1) % Operador
           [op,~]=identificarImagem(app.EscolherRede.Value,1);
           app.Resultado.Text = sprintf("[%d] %d %s",i+1,firstNumb,op);
       elseif(i==2)
           [~,secondNumb]=identificarImagem(app.EscolherRede.Value,0);
           if op == '+'
               resultado=firstNumb+secondNumb;
           elseif op == '-'
               resultado=firstNumb-secondNumb;
           elseif op == ':'
               resultado=firstNumb:secondNumb;
           elseif op == 'x'
               resultado=firstNumb*secondNumb;
           elseif op == '?'
               resultado=firstNumb+secondNumb;
           end
```

Figura 23 - Variáveis persistent e chamada da função identificarImagem() (código)

identificarImagem()

Esta função é responsável por devolver o resultado da **classificação** da respectiva rede neuronal a partir da imagem que lhe foi dada.

Na Figura 24, podemos observar parte do código implementado na função.

```
function [resultado,number] = identificarImagem(nomeRede , flag)

    caminhoRede = append("D:/Faculdade/2º-3ºAno/2º Semestre/CR/Projeto/",nomeRede);

    load(caminhoRede,'net');

    matrizBinaria = imagemBinaria();
    out = sim(net , matrizBinaria);

    [~,b] = max(out);
    disp(b);
```

Figura 24 - identificarImagem() (código)

imagemBinaria()

Esta função é responsável por processar a imagem guardada anteriormente (Figura 21), na Figura 25 podemos observar o **processamento** da imagem desenhada pelo User.

```
function[matrizBinaria] = imagemBinaria(~)

    % Define as dimensões da imagem
    image_size = [25 25];

    matrizBinaria = zeros(1 , image_size(1)*image_size(2)); % [1 , 25*25]

    caminhoImagem = 'D:/Faculdade/2º-3ºAno/2º Semestre/CR/Projeto/imagem.png';

    if exist(caminhoImagem, 'file') ~= 2
        error('O arquivo de imagem não existe ou o caminho está incorreto.');
    end

    current_image = imread(caminhoImagem);

    %Redimensiona a matriz binária para a dimensão das imagens
    resized_image = imresize(current_image, image_size);

    % Verificar se as bordas são pretas
    if ~all(resized_image(1,:) == 0) || ~all(resized_image(end,:) == 0) || ...
            ~all(resized_image(:,1) == 0) || ~all(resized_image(:,end) == 0)
        % Preencher as bordas com valor máximo (branco)
        resized_image(1,:) = 255; % Primeira linha
        resized_image(end,:) = 255; % Última linha
        resized_image(:,1) = 255; % Primeira coluna
        resized_image(:,end) = 255; % Última coluna
    end

    % Adiciona a matriz redimensionada à matriz de dados
    matrizBinaria(1, :) = resized_image(:)';

    matrizBinaria = matrizBinaria';

    imshow(resized_image);
end
```

Figura 25 - imagemBinaria() (código)

4 CONCLUSÃO

Com a realização do trabalho prático foi possível aprender mais sobre a utilização e capacidades das redes neuronais do tipo **"*feedfoward*"**, conseguindo entender a sua importância em diferentes situações.

Através deste trabalho, percebemos também que a parametrização e o tratamento prévio das imagens, são características importantes para um bom desempenho da rede.

5 BIBLIOGRAFIA

PowerPoints informativos da Teórica: https://moodle.isec.pt/moodle/