PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS

ESCOLA DE CIÊNCIAS EXATAS E DA COMPUTAÇÃO

VITOR DE ALMEIDA SILVA

Matricula: 2016.1.0033.0549-7

LISTA DE EXCÍCIOS N2 REDES NEURAIS
(*machine e deep learning*)

TALLES MARCELO G DE A BARBOSA

GOIÂNIA,

2019

DADOS DO ALUNO E DO CONTEÚDO:

Aluno: Vitor de Almeida Silva

Matrícula: 2016.1.0033.05497

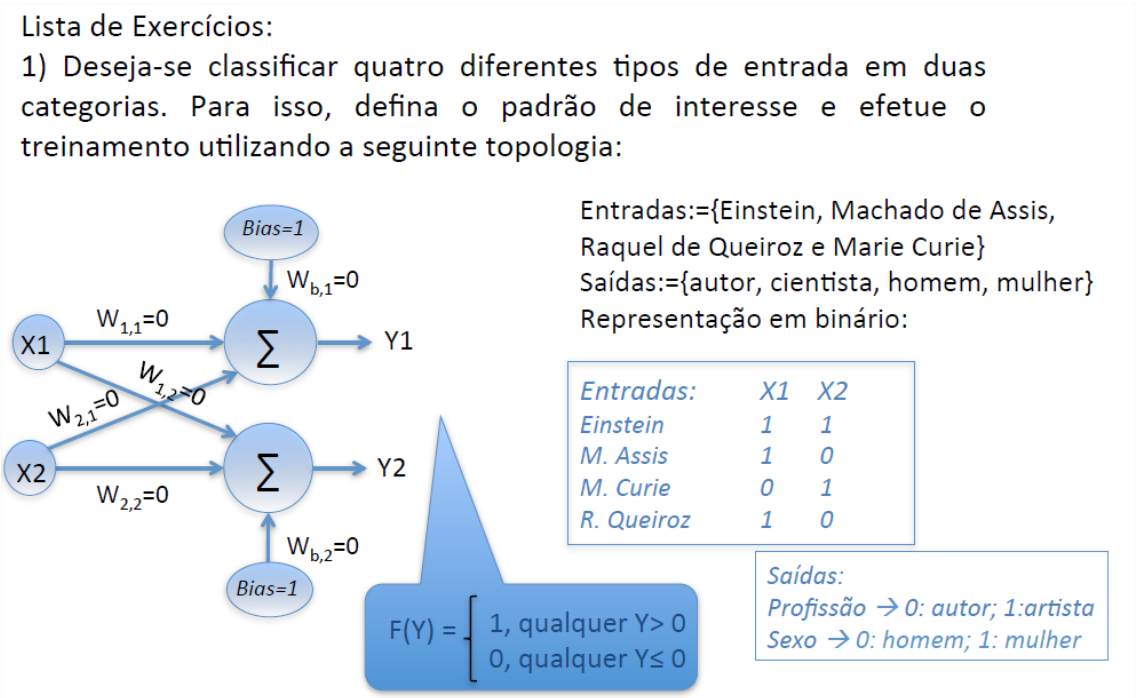
Conteúdo: Redes Neurais (machine learn e deep learn)

Perceptron, treino matlab (crap,wine,fat) alexNet

///EXERCÍCIOS REDES NEURAIS 1_7

1) **Exercício 1 mostrado na Figura 6.**

Figura 6: Enunciado exercício 1 redes neurais

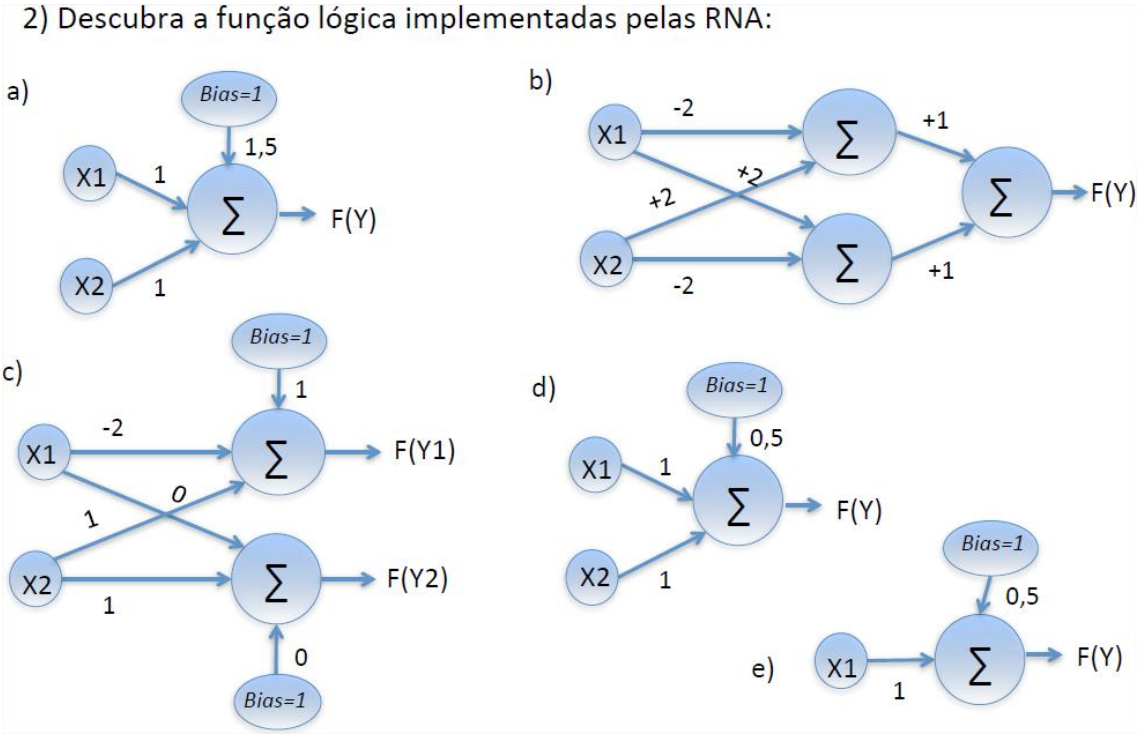


Fonte: Slide professor

//Exercício respondido na lista manuscrita em anexo, implementado no arquivo PerceptronVitorV1

2) **Exercício 2 mostrado na Figura 7:**

Figura 7: Enunciado exercício 1 redes neurais



Fonte: Slide professor

//Exercício respondido na lista manuscrita em anexo

3) **Refaça o treinamento do Exercício 1 com a taxa de aprendizagem igual a 0,2. Qual a conclusão?**

//Exercício respondido na lista manuscrita em anexo

4) **Refaca o treinamento do Exercicio 1 sem o bias. Qual a conclusao?**

//Exercício respondido na lista manuscrita em anexo

5) Suponha que você devera projetar um agente cuja função e reconhecer comandos (padrões) de entrada para o brinquedo Genius (Simon, na versão norte americana). O usuário deverá fornecer uma sequencia de entrada, composta por 4 toques. Esse padrão deverá ser uPlizado para o treinamento do Perceptron. Após o treinamento, o brinquedo deve acusar certo ou errado para cada interação com o usuário, composta também de quatro toques.

//Exercício respondido na lista manuscrita em anexo

1) **Descreva a topologia e execute o treinamento para que o Perceptron proposto seja capaz de identificar os seguintes padrões.**

//Exercício respondido na lista manuscrita em anexo

2) **Encontre os valores de R1, R2 e R3 para que o sistema seja ativado quando u1 U u2 U u3. Considere u1, u2, u3 variáveis binárias.**

//Exercício respondido na lista manuscrita em anexo e programado no arquivo Exerc_7_Perceptron.cpp

//exercícios dia 04/11 treinos com MLP

1) **Fazer manualmente o exemplo do MLP p/ mais uma entrada.**

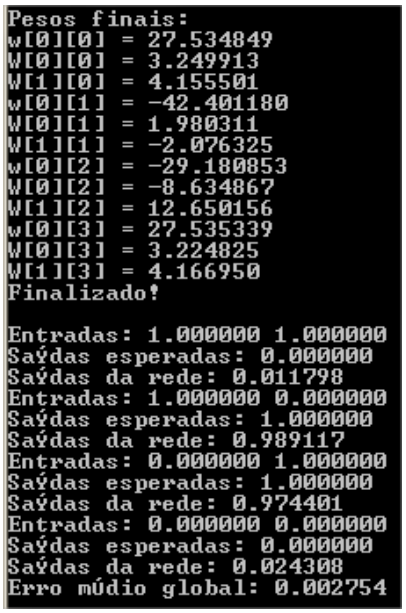
   //Exercício respondido na lista manuscrita

2) **Terminar o treinamento usando a topologia dada utilizando o MLP dado no grupo.**

   Resposta:

   O código MLP do slide tem como objetivo extrair a lógica de um xor. Este por sua vez não é linearmente separável, por isso é necessário utilizar de um preceptor multicamada. Após ajustado o código, o resultado obtido para os pesos para um total de 1000000 épocas foi o da Figura K. Foram testadas outras épocas, e foi notado que o resultado se aproxima do erro 0 para valores de taxa de aprendizagens menores e maiores valores de épocas.

Figura k: resultado treino MLP 1000000 épocas, erro=0, taxaAprendizagem =0.1



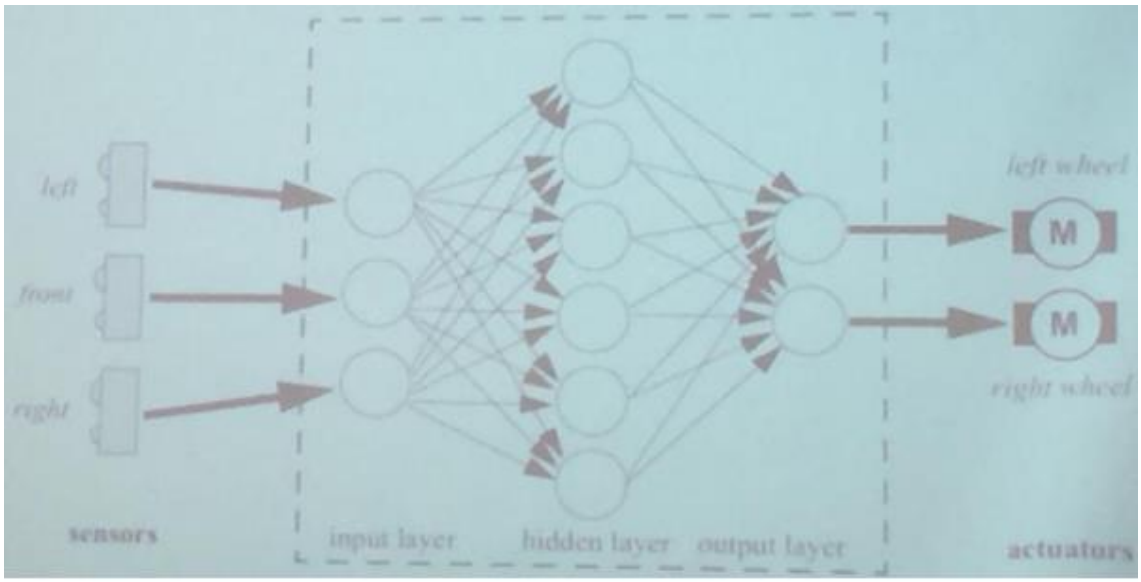
Fonte: Autoral

3) **Treino: Neural Network for driving a mobile robot, projetar esse treino.**

   Resposta:

   O modelo para o treino é mostrado na Figura E.

Figura E: Modelo do carro



Fonte: Livro

A Tabela B mostra as relações de entradas para o modelo da Figura E.

Tabela B: tabela de entradas

X1(left)	X2(front)	X3(right)	Y1 (left Wheel)	Y2 (right wheel)
0	0	0	1	1
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	1
1	0	1	1	1
1	1	0	1	0
1	1	1	0	0

Fonte: Autoral

Basicamente, sempre que um sensor for acionado, o veículo se move de forma a se desviar deste obstáculo, ele se locomoverá em outra direção. Considerei 0 como a condição na qual o motor gira em outro sentido, 1 horário e 0 anti-horário.

Foi realizado o treinamento, o qual retornou os padrões de saída mostrados na Figura 7b.

Figura 7b: Pesos treinamento

```
X= [ [0, 0, 0], [ 0,0,1],[0,1,0],[0,1,1],[1,0,0],[1,0,1],[1,1,0],[1,1,1] ]
y= [ [ 1,1], [1,0], [0,0], [0,1], [0,1], [1,1], [1,0], [0,0] ]

Pesos da camada do sensor da esquerda: [-0.13, -0.63, -22.34, -21.65, -0.55, -0.37, ]
Pesos da camada do sensor da frente: [-0.49, 7.71, -20.31, 21.89, -0.13, -8.43, ]
Pesos da camada do sensor da direita: [-0.46, -3.8, -13.91, 21.65, -0.13, -8.21, ]

Pesos da camada oculta saida roda esquerda: [0.62, 0.22, 20.93, -29.03, 0.76, -2.96, ]
Pesos da camada oculta saida roda direita:  [0.65, 1.23, 35.93, 0.26, 0.05, 4.91, ]

Vies da camada Oculta: [-0.59, -8.3, 12.91, -21.65, -0.3, -7.26, ]

Vies da camada de saida da roda esquerda: [1.1]
Vies da camada de saida da roda direita: [-0.7]
```

Fonte: Autoral

O código utilizado foi disponibilizado pelo colega de classe Lucas Macedo. Como segundo treinamento, foi utilizado o código do MLP, disponibilizado no grupo da classe e utilizado no exercício anterior. Os resultados do treinamento são mostrados na Figura 7c.

Figura 7c: resultados MLP para treino do carro

```
Pesos finais:
w[0][0] = 19.613640
w[1][0] = -5.361697
W[0][0] = -1.089062
W[1][0] = -13.272296
W[2][0] = 10.014665
w[0][1] = 42.202877
w[1][1] = 42.913078
W[0][1] = -2.880795
W[1][1] = -6.001485
W[2][1] = 1.090044
w[0][2] = -28.254608
w[1][2] = -30.778990
W[0][2] = 2.224548
W[1][2] = 6.366318
W[2][2] = 1.912584
w[0][3] = -4.906192
w[1][3] = 23.375805
W[0][3] = 9.918322
W[1][3] = -13.563569
W[2][3] = -1.407617
Finalizado!

Entradas: 0.000000 0.000000 0.000000
Saýdas esperadas: 1.000000 1.000000
Saýdas da rede: 0.447961 0.449861
Entradas: 0.000000 0.000000 1.000000
Saýdas esperadas: 1.000000 0.000000
Saýdas da rede: 1.000000 0.009020
Entradas: 0.000000 1.000000 0.000000
Saýdas esperadas: 0.000000 0.000000
Saýdas da rede: 0.392625 0.392079
Entradas: 0.000000 1.000000 1.000000
Saýdas esperadas: 0.000000 1.000000
Saýdas da rede: 0.392839 0.392217
Entradas: 1.000000 0.000000 0.000000
Saýdas esperadas: 0.000000 1.000000
Saýdas da rede: 0.009046 1.000000
Entradas: 1.000000 0.000000 1.000000
Saýdas esperadas: 1.000000 1.000000
Saýdas da rede: 0.999974 0.999968
Entradas: 1.000000 1.000000 0.000000
Saýdas esperadas: 1.000000 0.000000
Saýdas da rede: 0.392504 0.392014
Entradas: 1.000000 1.000000 1.000000
Saýdas esperadas: 0.000000 0.000000
Saýdas da rede: 0.392539 0.391978
Erro múdio global: 0.148769
```

Fonte: Autoral

//EXERCÍCIOS REDE NEURAL MATLAB EXEMPLOS OBS: os pesos estão presentes em arquivos de texto separados por serem muitos

Exemplo 1) executar o treinamento para classificar o sexo de caranguejos, usar os passos abaixo:

Fonte: < https://ch.mathworks.com/help/deeplearning/examples/crab-classification.html>

a) **executar para uma rede com 4 neurônios;**
b) **executar com 10 neurônios;**
c) **adicionar uma camada oculta com 4 neurônios cada e executar;**

Este exemplo ilustra o uso de uma rede neural como classificador para identificar o sexo de caranguejos a partir de suas dimensões físicas. Para isso, vão ser usadas 6 características físicas do caranguejo, são elas:

- *Species* (espécie);
- *Frontallip* (“medida frontal”);
- *Reawidth* (largura trazeira);
- *Length* (comprimento);
- *Width* (Largura);
- *Depth* (profundidade).

O problema em mãos é identificar o sexo do carambguejo a partir dos valores dados em cada uma dessas características. Deste modo, foi executado no exemplo os seguintes passos.

1) **Obtenção do *dadaset* (conjunto de dados):** Um conjunto de dados já presente no matlab, produzido por especialistas, contem os dados dos caranguejos;
2) **Construir a rede neural classificadora:** especifica as características da rede (camadas, performance e treino);
3) **Testa o classificador:** usa-se um pedaço do conjunto de dados para testar a rede treinada e verificar seus resultados.

Definido os pontos principais do exemplo, pode-se partir para a parte prática.

a) executar para uma rede com 4 neurônios;

Após adicionado a data set, para criar uma rede de 4 neurônios, basta usar o comando da Figura 25.

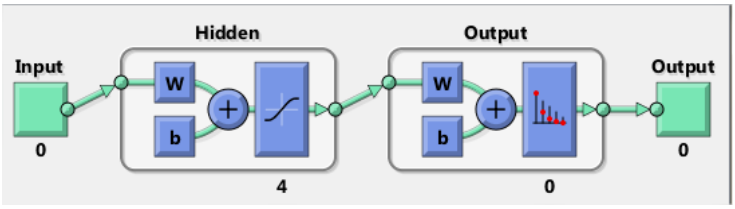
Figura 25: comando cria rede

```
net = patternnet(4);
view(net)
```

Fonte: Matlabworks (2015)

Dessa Forma a topologia adquiriu a Configuração da Figura 26 configurações.

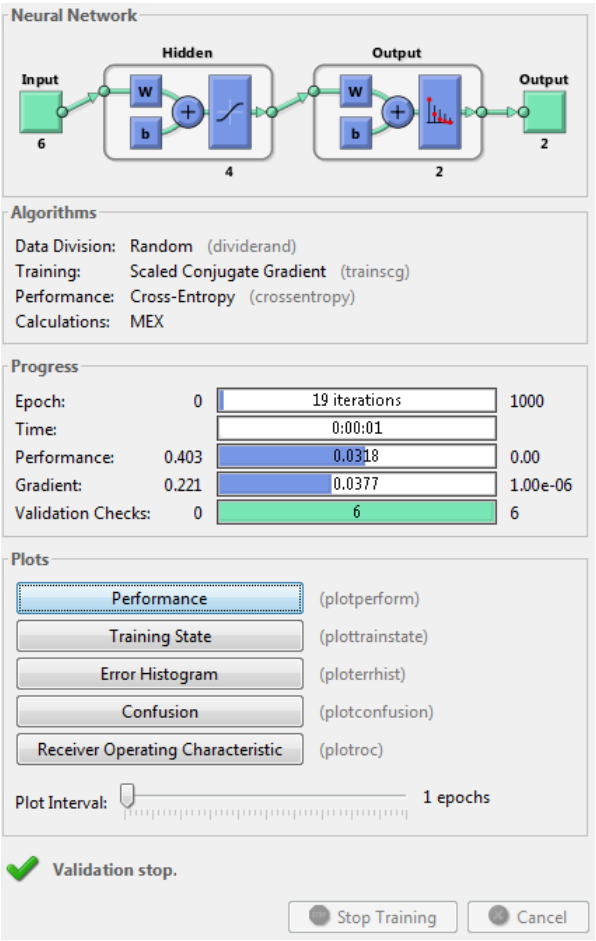
Figura 26: Topologia da rede



Fonte: Autoral

Com essa topologia ela conseguiu os dados de desempenho mostrados na Figura 27. Treino consistiu de 19 interações e um tempo de 0,01 segundos.

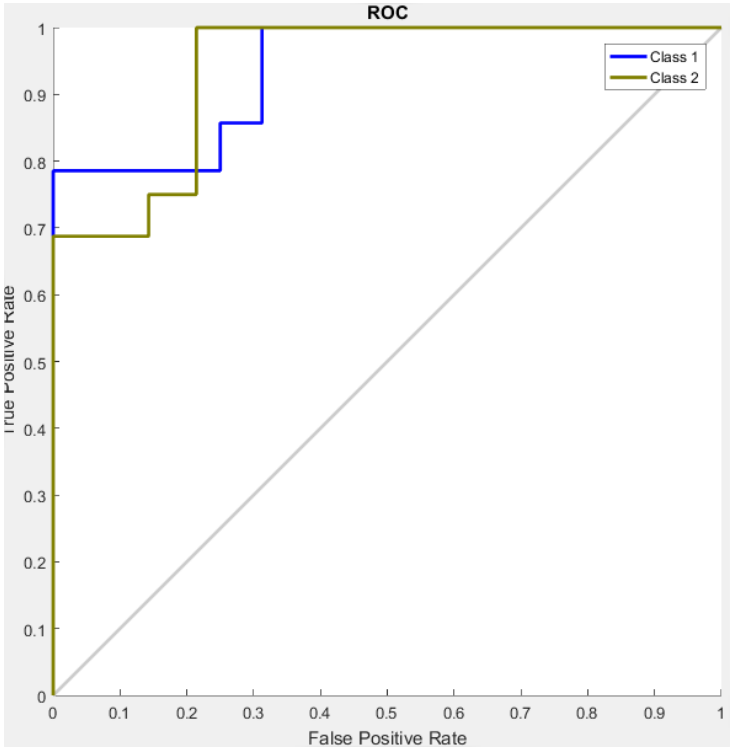
Figura 27: Treino da rede



Fonte: Autoral

A Figura 28 apresenta o gráfico ROC do atual modelo. Analisando o Gráfico é possível notar que o modelo permaneceu acima da linha horizontal, indicando que é um modelo bom.

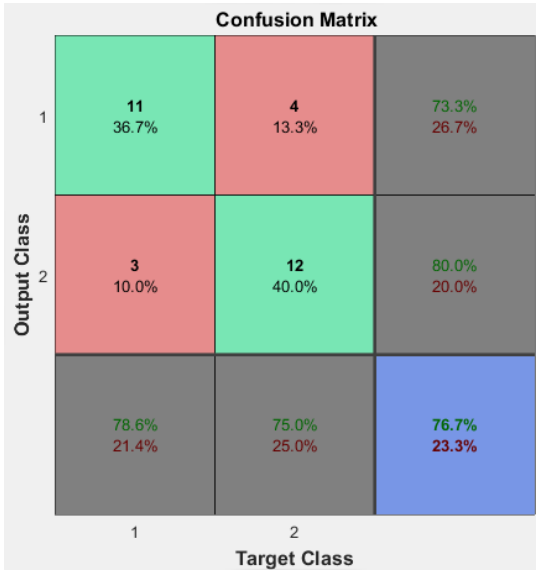
Figura 28: Curva ROC



Fonte: Autoral

A Figura 29 mostra a matriz de confusão do modelo.

Figura 29: Matriz de confusão modelo a



Fonte: Autoral

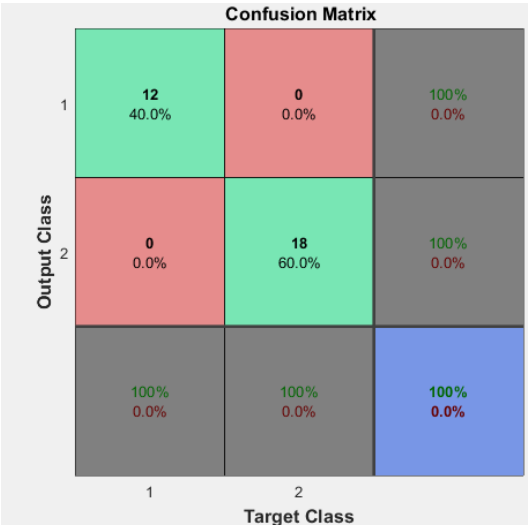
A partir da matriz de confusão é possível visualizar a quantidade de acertos que o modelo obteve. Para a presente rede com 4 neurônios foi obtido

11 acertos para a classe 1 e 12 acertos para a classe 2. Sendo que nos quadros vermelhos, ou seja, os erros, foram 3 classificações erradas para 2 e 4 erradas para 1.

b) executar com 10 neurônios.

A Figura 30 apresenta a matriz de confusão para uma rede de 10 neurônios e 1 *hidden layer* (camada oculta).

Figura 30: matriz de confusão b



Fonte: Autoral

Com uma análise da matriz da Figura 30, nota-se que, o modelo conseguiu uma melhora significativa. Os resultados para uma rede neural de 10 neurônios geraram 100% de acerto para no presente exemplo.

d) adicionar uma camada oculta com 4 neurônios cada e executar;

Para aumentar a quantidade de camadas ocultas basta utilizar códigos semelhantes ao da Figura 31.

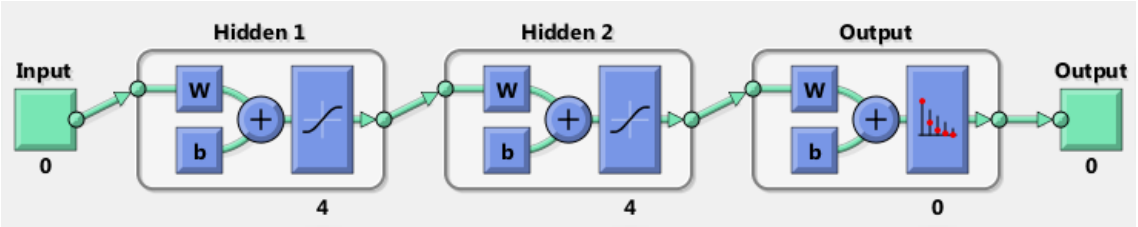
Figura 31: rede neural com duas camadas ocultas

```
hiddenLayer1Size = 4;
hiddenLayer2Size = 4;
%%net = fitnet([hiddenLayer1Size hiddenLayer2Size]);
net = patternnet( [hiddenLayer1Size hiddenLayer2Size] );
```

Fonte: Autoral

A Figura 32 mostra a topologia da rede neural com 2 camadas ocultas.

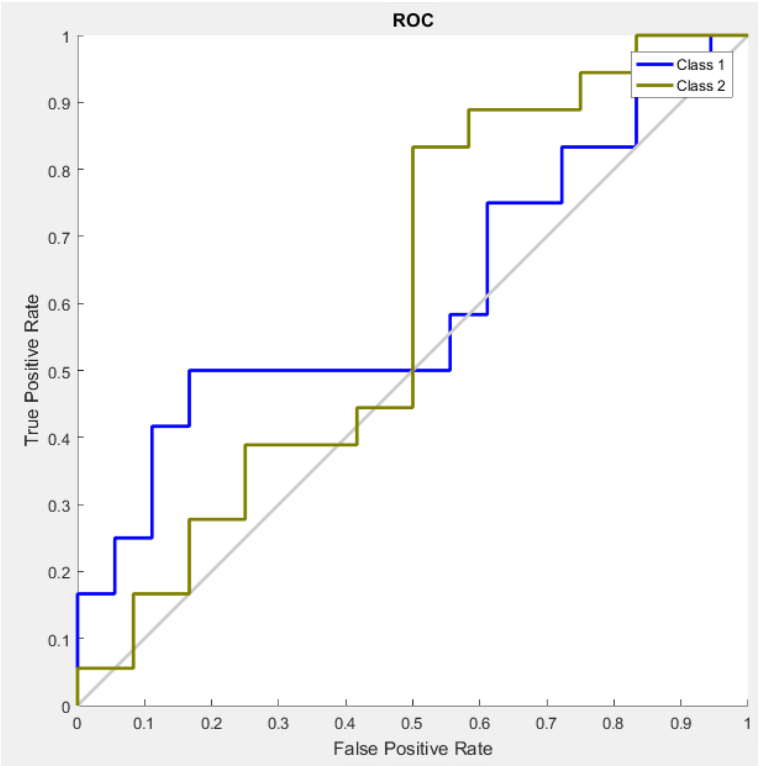
Figura 32: rede neural com duas camadas ocultas



Fonte: Autoral

A Figura 33 mostra o gráfico de ROC, o qual mostra que o modelo de 2 camadas ocultas não conseguiu ser melhor que os outros 2 modelos.

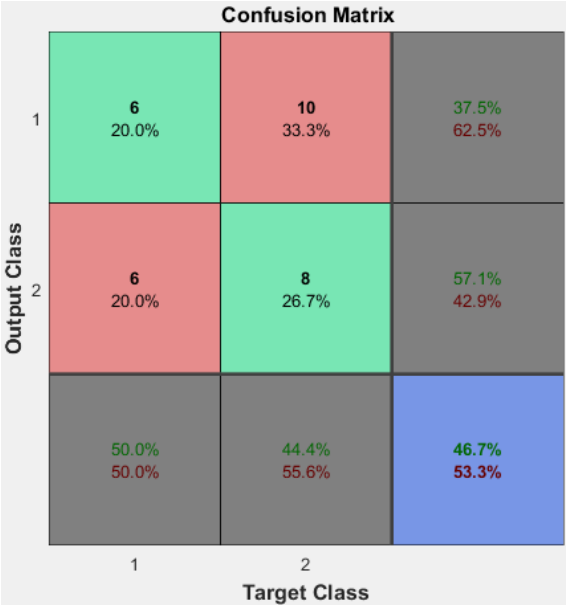
Figura 33: Gráfico ROC



Fonte: Autoral

A Figura 34 mostra a matriz de confusão para o modelo corrente.

Figura 34: Matriz de confusão c



Fonte: Autoral

Dessa forma, com a análise da matriz, é possível notar que ouve uma piora dos resultados. A porcentagem de erros aumentou de 0 para 6 na classe 2 e de 0 para 10 na classe 1. Visto isso, pode se concluir que **nem sempre mais camadas significam melhores resultados.** Para concluir, a melhor topologia ficou sendo a do item b, com 10 neurônios e 1 camada oculta.

Exemplo 2) executar o treinamento para classificar o vinho, usar os passos abaixo:

Fonte: < https://ch.mathworks.com/help/deeplearning/examples/wine-classification.html>

a) **executar para uma rede com 4 neurônios;**
b) **executar com 10 neurônios;**
c) **adicionar uma camada oculta com 4 neurônios cada e executar;**

Este exemplo desenvolve a criação de uma rede neural para classificação vinhos por meio de suas características químicas. Este exemplo se focou na classificação de vinhos de três vinharias diferentes usando 13 atributos diferentes, sendo eles:

- *Alcohol;*
- *Malic acid;*
- *Ash;*
- *Alkalinity of ash;*
- *Magnesium;*
- *Total phenols;*
- *Flavonoids;*
- *Nonflavonoid phenol;*
- *Proanthocyanidin;*
- *Color intensity;*
- *Hue;*
- *OD280/OD315 of diluted wines;*
- *Proline.*

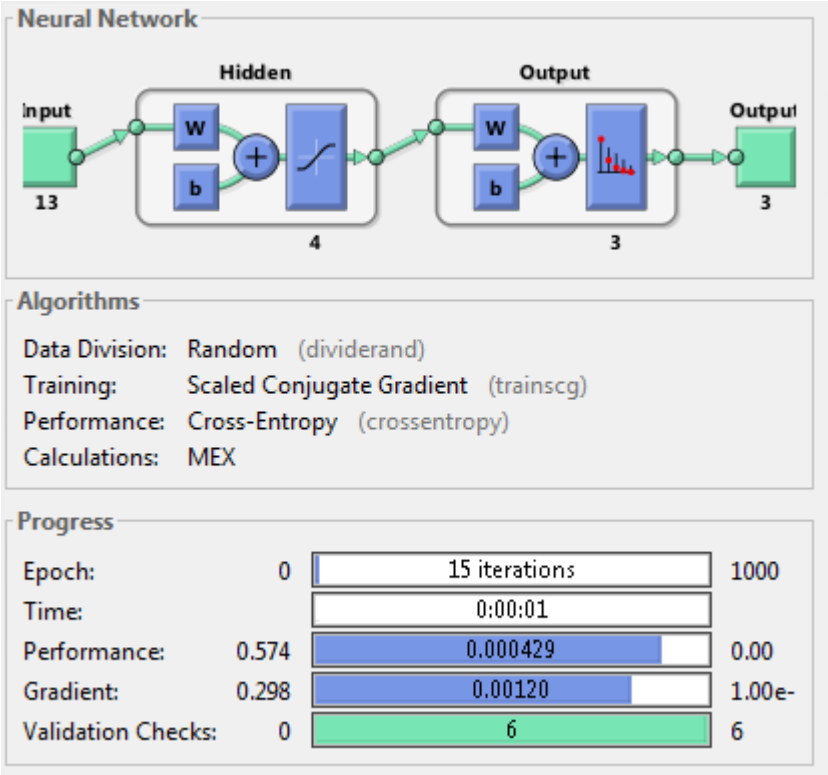
A ideia é criar uma rede neural que possa ser generalizada para classificar outros vinhos fora *data base* de treino. Para organizar os dados do problema foram definidas duas matrizes, uma matriz alvo e uma matriz de entrada. Cada matriz é composta por 3 classes diferentes, o campo que se associar ao vinho correspondente recebe 1 caso o contrário recebe 0 (MATLABWORKS, 2015).

O treino é seguido como o exercício 1. Dessa forma basta partir para os testes.

a) executar para uma rede com 4 neurônios;

Depois de carregado o *dataSet* e realizado a criação e treino da rede, se obteve a topologia com seus dados mostrada na Figura 35.

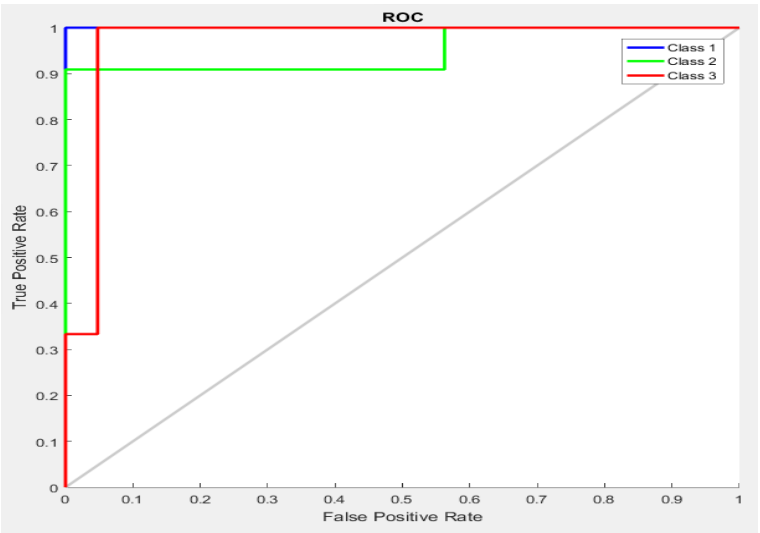
Figura 35: Rede vinho



Fonte: Autoral

A curva ROC é mostrada na Figura 36.

Figura 36: Grafico ROC

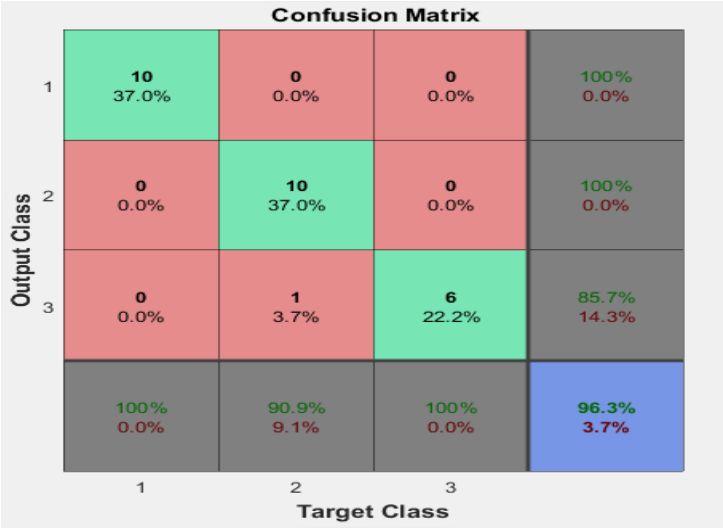


Fonte: Autoral

O gráfico indica que o modelo fez uma boa classificação dos dados.

A matriz de confusão é mostrada na Figura 37.

Figura 37: Matriz de confusão 2-a



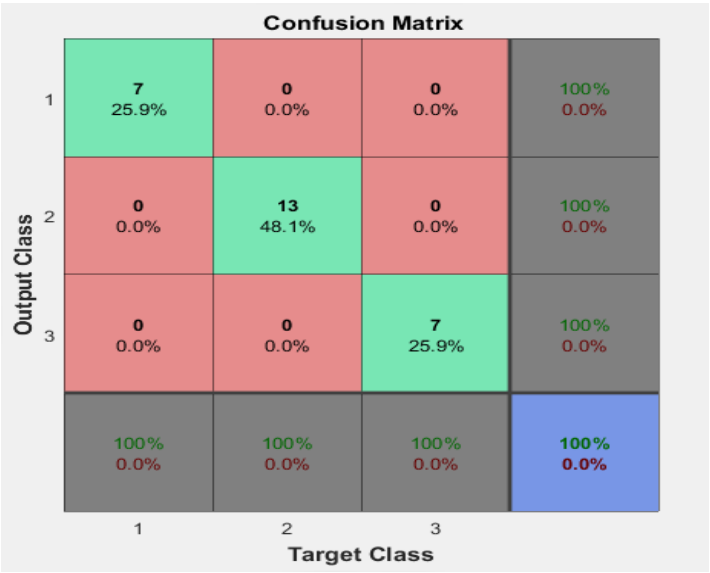
Fonte: Autoral

A matriz de confusão indicou somente um erro na classificação.

b) executar com 10 neurônios;

Mudando a topologia de 4 para 10 neurônios se obteve a Matriz de confusão da Figura 38.

Figura 38: Matriz de confusão 2-b



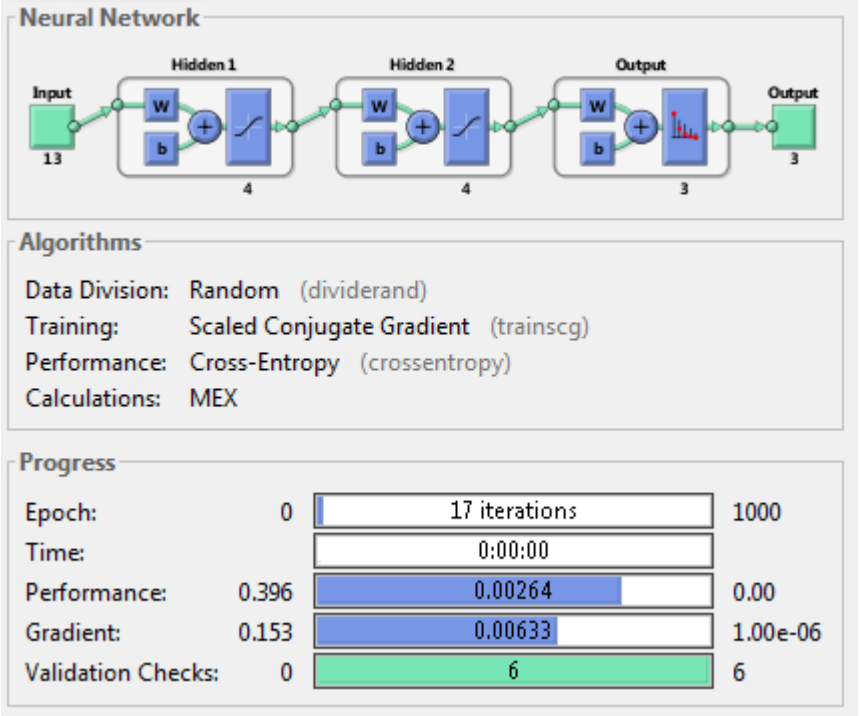
Fonte: Autoral

A matriz indica que a topologia com 10 neurônios e 1 camada oculta foi capaz de classificar corretamente todos os vinhos.

c) adicionar uma camada oculta com 4 neurônios cada e executar;

A topologia e o treino são mostrados na Figura 39.

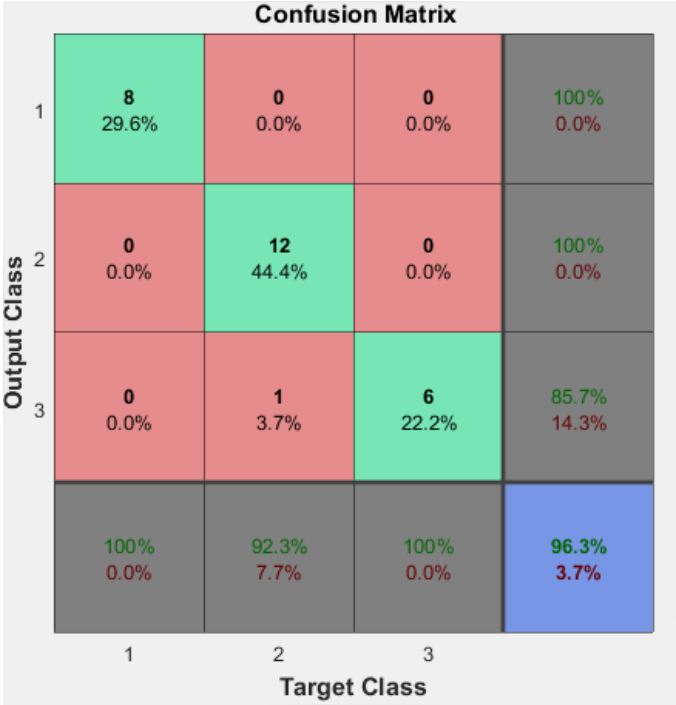
Figura 39: Matriz de confusão 2-b



Fonte: Autoral

A matriz de confusão é mostrada na Figura 40.

Figura 40: Matriz de confusão 2-c



Fonte: Autoral

Para essa topologia de 2 camadas ocultas com 4 neurônios cada, o classificador ainda obteve 1 erro. Dessa forma a melhor topologia para o classificador é a do item b.

Exemplo 3) executar o treinamento para classificar de gordura do corpo humano, usar os passos abaixo:

Fonte: <https://ch.mathworks.com/help/deeplearning/examples/body-fat-estimation.html>

a) **executar para uma rede com 4 neurônios;**
b) **executar com 10 neurônios;**
c) **adicionar uma camada oculta com 4 neurônios cada e executar;**

Esse exemplo se foca na criação de uma rede neural do tipo *fittin* (combinação, apropriado) que seja capaz de estimar o percentual de gordura do corpo de uma pessoa descrita. Para isso, serão usados 13 atributos físicos, são eles:

- *Age (years)*
- *Weight (lbs)*
- *Height (inches)*
- *Neck circumference (cm)*
- *Chest circumference (cm)*
- *Abdomen circumference (cm)*
- *Hip circumference (cm)*
- *Thigh circumference (cm)*
- *Knee circumference (cm)*
- *Ankle circumference (cm)*
- *Biceps (extended) circumference (cm)*
- *Forearm circumference (cm)*
- *Wrist circumference (cm)*

Este exemplo segue o modelo de processos dos dois anteriores, sendo ele:

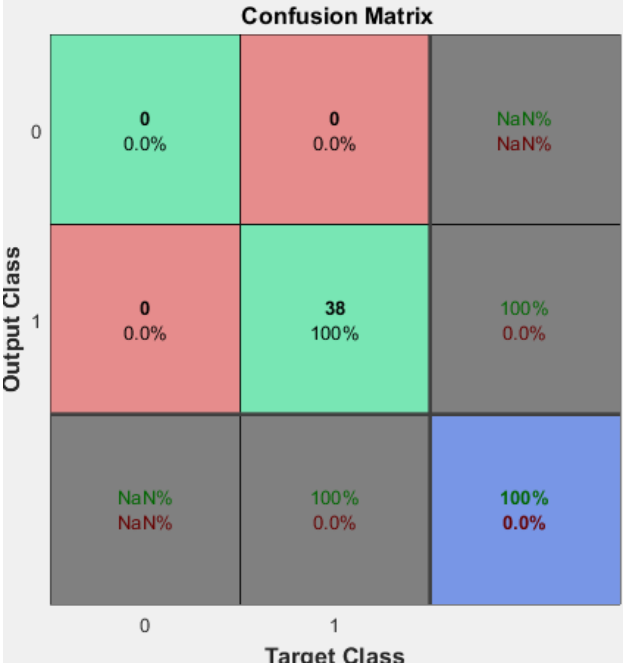
1) **Obtenção do *dadaset* (conjunto de dados):** Um conjunto de dados já presente no matlab, produzido por especialistas, contém os dados dos caranguejos;
2) **Construir a rede neural classificadora:** especifica as características da rede (camadas, performance e treino);
3) **Testa o classificador:** usa-se um pedaço do conjunto de dados para testar a rede treinada e verificar seus resultados.

Definido os pontos principais do exemplo, pode-se partir para a parte prática. cabe destacar que esse exemplo usa uma rede neural do tipo *fit*

a) executar para uma rede com 4 neurônios;

Desse modo foi gerado uma rede neural com 1 camada oculta e 4 neurônios em cada camada. A Figura 41 mostra a Matriz de confusão do problema.

Figura 41: Matriz de confusão 3_a



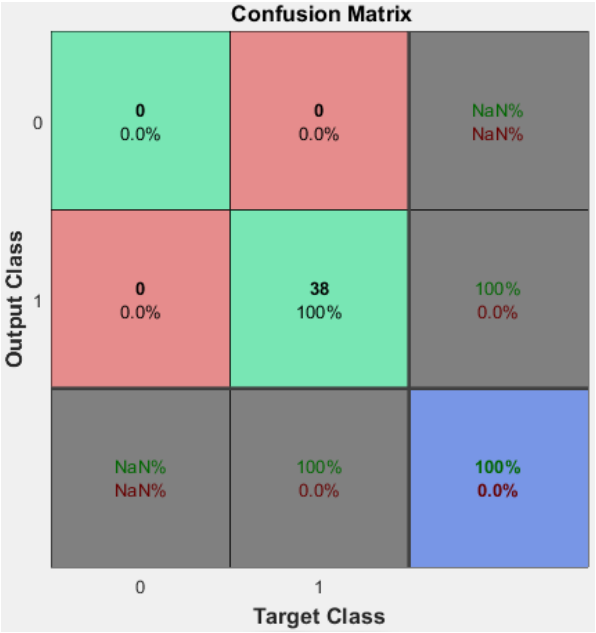
Fonte: Autoral

Logo de primeira é possível ver que todas as classificações ficaram corretas. Foram 38 classificações corretas para a classe 1.

b) executar com 10 neurônios;

Desse modo foi gerado uma rede neural com 1 camada oculta e 10 neurônio em cada camada. A Figura 42 mostra a Matriz de confusão do problema.

Figura 42: Matriz de confusão 10_b



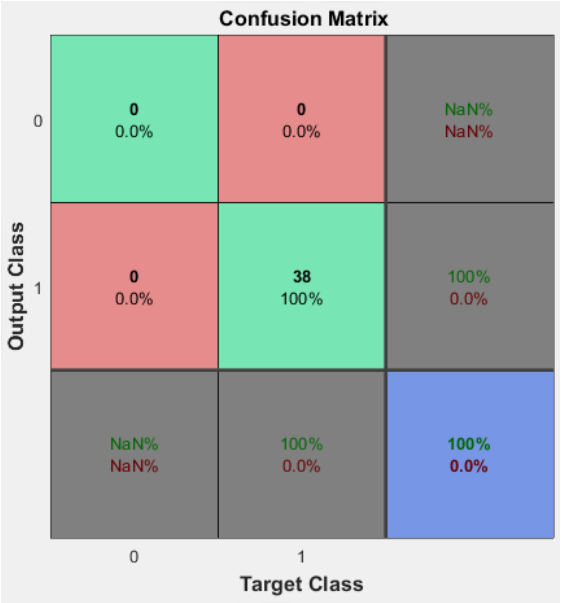
Fonte: Autoral

Da mesma forma que o item 3_a, a matriz de confusão para o exemplo com 10 neurônios mostrou uma rede com 100% de acertos.

c) adicionar uma camada oculta com 4 neurônios cada e executar;

Para este teste, o modelo será mudado para um com 2 *hidden layers* e 4 neurônios em cada uma. A Figura 43 mostra a matriz de confusão para o modelo atual.

Figura 43: Matriz de confusão 3_c



Fonte: Autoral

De forma semelhante aos tópicos 3_a e 3_b, a matriz de confusão ficou igual, com 100% de acertos. Isso indica que o modelo já convergiu para o desejado logo na questão a. Dessa mesma forma, o problema não apresentou mais erros.

//ESTUDO SOBRE DEEP LEARNING (TREINAMENTOS) AlexNet()

Nesta parte iniciou-se os estudos referentes a *deep leaning* (aprendizagem profunda). *Deep leaning* é o nome dado ao tipo de rede neural com centenas de camas ocultas e que lida tanto com o pré-processamento da informação e com o treino. Isto é, em uma rede *deep leaning* é possível usar como entrada uma imagem diretamente, sem nem um tratamento (se desejar). A imagem irá passar pela análise das diversas *hiden-leyers* de forma que seja possível extrair as informações da mesma e o modelo será também treinado.

Por conta de ser um tipo de rede muito grande, é muito complexo compreender oque ocorre dentro de seu interior. Por essa causa, redes de *deep learning* funcionam normalmente como caixa preta, isso implica que o desenvolvedor normalmente terá pouco controle em relação a como a rede faz os treinos, com isso, faz-se necessário verificar as classificações e retreinar caso necessário.

Dessa forma, redes desse tipo acabam necessitando de *dataSets* (conjuntos de dados) muito grandes para melhor classificarem os dados.

Uma rede neural profunda consiste de um modelo baseado na biologia. Esta representação é na verdade uma grande quantia de neurônios conectados formando sistema não lineares. Cada conjunto de neurônios representa uma *hiden-leayer*, estas usam a saída da layer anterior como entrada e formar uma saída, ao final os resultados da classificação são mandas na camada de saída.

AlexNet entra nesse contexto como uma rede já treinada por experts. Ela é capaz de reconhecer 1000 objetos diferentes. *AlexNet* pode ser usada para reconhecer mais objetos, através do treino com outros pequenos dataSets, essa é uma técnica chamada de *transfer Leaning* (transferência de aprendizagem).

Redes neurais utilizam da repetição de 3 processos principais para extrair e reconhecer características de imagens, são elas:

- **Convolução:** é utilizada como um filtro, usado para extrair características da imagem;

- **Pooling:** simplifica a saída por uma amostragem não linear, dessa forma, reduz os parâmetros sobre o qual a rede tem que aprender;
- **Unidade linear retificada (Rectified linear unit (ReLU)):** usada para um treino mais efetivo e rápido, através do mapeamento de valores negativos levando-os para 0 e mantendo os valores positivos;

Esses três pontos são repetidos até a rede ter detectados as características.

1) Treinar a alexNet para identificar alguma classe escolhida.

Resposta:

Para treinar a alexNet é preciso instalar no matlab 2 pacotes de suport:

- **webCan:** para poder utilizar a webCan para obternção de imagens;
- **AlexNet:** pacote de suporta para a própria rede alexNet;

Também é necessário redimensionar as imagens para 270x270 pixels.

Essas condições acima permitem utilizar a alexNet. Porém, para treinar a rede para reconhecer algum objeto fora do conjunto dela é necessário utilizar de *tranfer leaning.*

Transfer learning é realizar uma aproximação do conhecimento de algum tipo a um problema diferente porém relacionado ao existente. Neste caso, pega-se as 3 ultimas camadas da rede e realiza-se o retreino com as imagens originais selecionadas.

Para retreinar a rede é preciso seguir alguns passos nos códigos do matlab, são eles:

1) Montar um conjunto de dados com as imagens;
    - Separar em pastas com os nomes das classes;
2) Carregar a rede alexNet;
3) Modificar a rede para usar o número de categorias desejadas;
4) Configurar os dados de treino;

5) Retreinar a rede;
6) Mostrar os resultados de desempenho;
7) Testar a rede para as imagens novas;

Definido os passos, o que foi definido para ser classificado neste exercício foram gatos e patos. Para tanto, foram cridas duas pastas cada uma representando uma classe, um total de 10 imagens para cada pasta. Também foi criada uma pasta com algumas imagens para treinar o modelo.

Feito isso foi realizado o treino. A figura 49 mostra os dados de retreino da alexNet tendo em vista CPU e a normalização da imagem.

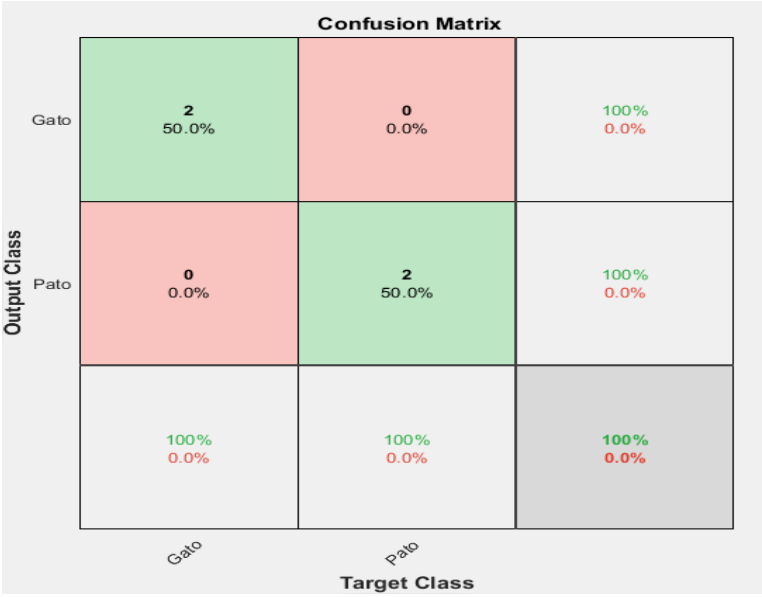
Figura 49: dados da classificação

```
>> alexnet_retreino
Training on single CPU.
Initializing image normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:08 |       36.84% |       1.6072 |          0.0010 |
|      20 |          20 |       00:02:49 |      100.00% |   8.3636e-06 |          0.0010 |
|=========================================================================================|

acuracia =

     1
```

Fonte: autor

A acurácia do modelo foi 1. A matriz de confusão é mostrada na Figura 50.

Figura 50: matriz de confusão modelo pato_gato



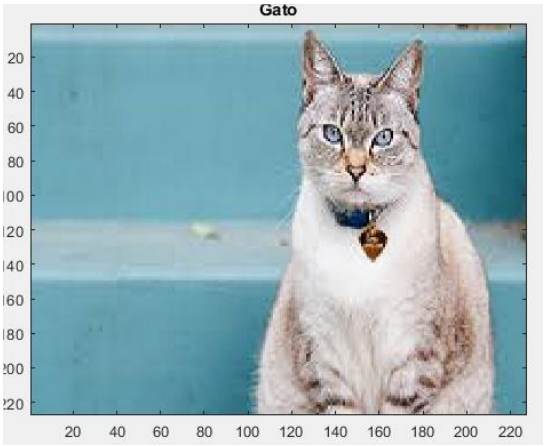
Fonte: autoral

Analisando a Figura 50, é possível nota que, o modelo não teve nem um erro na classificação, as imagens testas estão todas na diagonal da matriz. Porém, para testes, serão usadas tanto imagens de gatos e patos, quanto imagens aleatórias para ver a classificação. Com isso, as Figuras 51, 52, 53, 54, 55, 56, 57 mostram os testes realizados.

Figura 51: pato comum

Fonte: google imagens

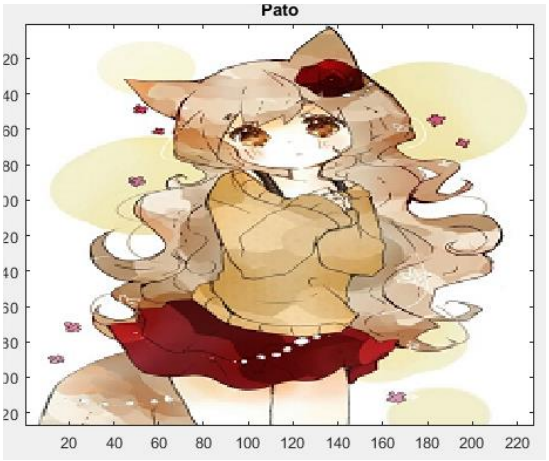
Figura 52: gato



Fonte: google imagens

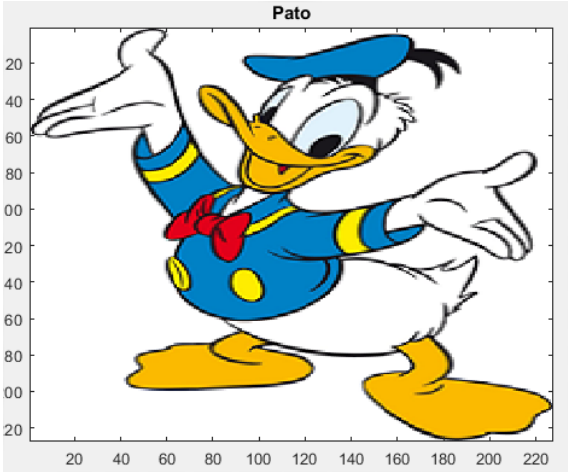
Os dois animais foram classificados corretamente, as Figuras a seguir usam imagens mais diferentes.

Figura 53: teste com uma ***neko girl*** (personagem de anime vestida de gato)



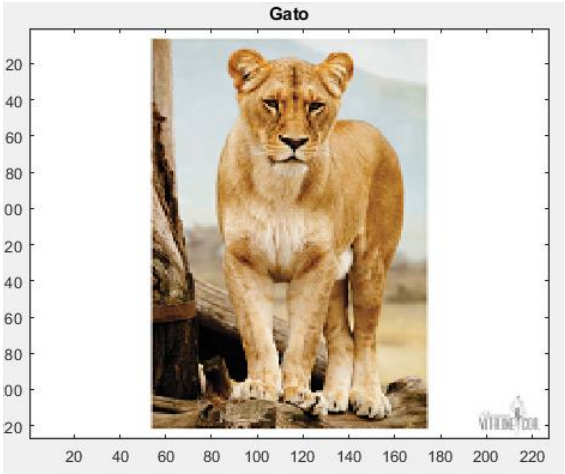
Fonte: google imagens

Figura 54: pato donald



Fonte: google imagens

Figura 55: leoa

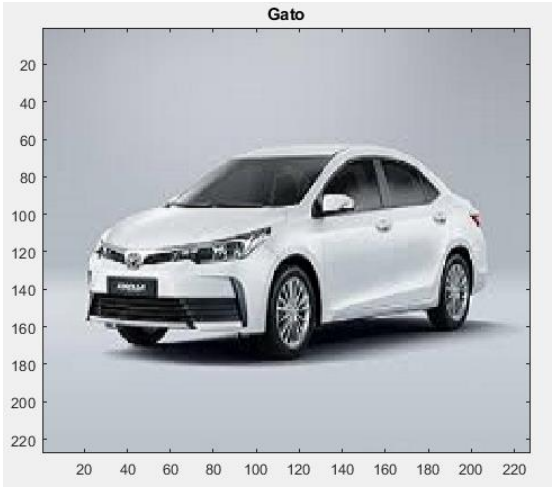


Fonte: google imagens

Figura 56: porco cartoon

Fonte: google imagens

Figura 57: carro



Fonte: google imagens

A figura 54 foi classificada corretamente como um pato, provavelmente as características como cor das penas e dos bicos tenha contribuído. A Figura 55 é de uma leoa, está foi classificada como sendo um gato, pode-se considerar que o modelo conseguiu adequar bem o resultado, visto que, só existem duas classes. As Figuras 53, 56 e 57 estavam muito diferentes de patos ou gatos então foram classificadas como estão.

O código utilizado para retreinar a rede é mostrado na 58.

Figura 58: código *tranfers learning* alexNet

```
alex = alexnet;
layers = alex.Layers;

% Modifica a rede para usar cinco categorias
layers(23) = fullyConnectedLayer(2);
layers(25) = classificationLayer;

% Configurando os dados de treino
imagens = imageDatastore('D:\8º Periodo\IA N2\Redes Neurais\AlexNet\Vitor\Treino',
'IncludeSubfolders', true, 'LabelSource', 'foldernames');
[imagens_treino, imagens_teste] = splitEachLabel(imagens, 0.8, 'randomize');

% Retreinando a redes
opts = trainingOptions('sgdm', 'InitialLearnRate', 0.001, 'MaxEpochs', 20, 'MiniBatchSize',
64);
minhaRede = trainNetwork(imagens_treino, layers, opts);

% Desempenho da rede
predictedLabels = classify(minhaRede, imagens_teste);
acuracia = mean(predictedLabels == imagens_teste.Labels)

plotconfusion (predictedLabels, imagens_teste.Labels)


figure();
img = imread('D:\8º Periodo\IA N2\Redes Neurais\AlexNet\Vitor\Teste\1.png');
label = classify(minhaRede, img );
% Classify the picture
image(img); % Show the picture
title(char(label)); % Show the label

figure();
img = imread('D:\8º Periodo\IA N2\Redes Neurais\AlexNet\Vitor\Teste\2.png');
label = classify(minhaRede, img );



% Classify the picture
image(img); % Show the picture
title(char(label)); % Show the label

figure();
img = imread('D:\8º Periodo\IA N2\Redes Neurais\AlexNet\Vitor\Teste\3.png');
label = classify(minhaRede, img );
% Classify the picture
image(img); % Show the picture
title(char(label)); % Show the label

figure();
img = imread('D:\8º Periodo\IA N2\Redes Neurais\AlexNet\Vitor\Teste\4.png');
label = classify(minhaRede, img );
% Classify the picture
image(img); % Show the picture
title(char(label)); % Show the label

figure();
img = imread('D:\8º Periodo\IA N2\Redes Neurais\AlexNet\Vitor\Teste\6.png');
label = classify(minhaRede, img );
% Classify the picture
image(img); % Show the picture
title(char(label)); % Show the label

figure();
img = imread('D:\8º Periodo\IA N2\Redes Neurais\AlexNet\Vitor\Teste\5.png');
label = classify(minhaRede, img );
% Classify the picture
image(img); % Show the picture
title(char(label)); % Show the label

figure();
img = imread('D:\8º Periodo\IA N2\Redes Neurais\AlexNet\Vitor\Teste\7.png');
label = classify(minhaRede, img );
% Classify the picture
image(img); % Show the picture
title(char(label)); % Show the label
```

Fonte: autor

REFERÊNCIA

MATLABWORKS (Eua). **Crab Classification.** 2015. Disponível em: <https://ch.mathworks.com/help/deeplearning/examples/crab-classification.html>. Acesso em: 18 nov. 2019.

MATLABWORKS (Eua). **Wine Classification.** 2015. Disponível em: <https://ch.mathworks.com/help/deeplearning/examples/wine-classification.html>. Acesso em: 18 nov. 2019.

RUSSEL, Stuart; NORVIG, Peter. **Inteligência artificial.** Rio de Janeiro: Elsevier Editora, 2010.