Projeto de Pesquisa - Departamento de Estatística
Universidade Federal Fluminense

Classificação de Imagem

Jessica Quintanilha Kubrusly
Filipe Silva Nascimento
Matheus Alves Pereira
Thiago Augusto Santos Lima

Resumo

O projeto em questão iniciou-se como um estudo sobre reconhecimento de imagens. Nas primeiras leituras percebemos que o método mais adequado para resolver esse problema é o *Deep Learning*. Diante disso o objetivo do projeto passou a ser organizar e disponibilizar uma sequência de *scripts* em R que possa ser utilizada para diferentes problemas de reconhecimento de imagens. O *scripts* pode ser separado em três partes: leitura e redimensionamento das imagens, separação em bases de treino, validação e teste e criação do modelo de *Deep Learning* usado para classificar as imagens a partir de um rótulo pré-existente. Com um único *scripts*, composto por essas três partes, e adequado para cada dado, foi possível realizar três estudos de casos. Os estudos de casos apresentam diferentes situações e formas de resolvê-las, como por exemplo, imagens em preto e branco ou imagens coloridas, classificação binária ou de mais classes, bancos muito grandes e outros não tão grande assim. Esceto de um estudo de caso, com imagens coloridas e de base não muito grande, os resultados forma muito bons. Mostrado que de fato o *Deep Learning* é a ferramenta adequada para o problema de classificação de imagens.

Introdução

Problemas de processamento de imagens são, em geral, desafiadores. Tanto pela sua dimensão, uma vez que cada pixel da imagem é transformado em uma variável explicativa, quanto pela exigência computacional do pré-processamento, que precisa ser capaz de criar um banco estruturado a partir de arquivos de imagens.

Se por um lado os problemas são desafiadores, por outro as aplicações são interessantíssimas: reconhecimento facial, identificação de tumores em imagens médicas, identificação de objetos em fotos, entre outras.

Estre projeto tem como objetivo organizar uma sequência de scripts na linguagem de programação R [7] que facilite facilite a análise de dados. A hipótese é que o analista de dados dispões de um diretório de imagens com arquivos na extensão jpeg. Aplicando corretamente a sequência de *scripts* ele conseguirá aplicar métodos de *Deep Leanring* para um dos dois objetivos finais:

- classificar objetos nas imagens;
- localizar objetos nas imagens.

Vale ressaltar que a sequência de *scripts* não se resume na aplicação dos métodos de *Deep Learning*, nela também é tratado a parte de pré-processamento. Ou seja, a única suposição é que existe um diretório com imagens. Tanto a leitura dessas imagens pelo Programa R quanto a transformação delas em objetos possíveis de serem utilizados no método de *Deep Learning* fazem parte dos comandos contemplados pelos *scripts*. Todos os scripts foram implementados na linguagem de programação do R [7].

Todo material desenvolvido neste projeto será disponibilizado livremente pelo GitHub da professora Jessica Kubrusly[1].

[1] https://github.com/jessicakubrusly/image_classification.git

Materiais e Métodos

Materiais

O banco de dados utilizado neste estudo é formado por imagens. Cada imagem, na prática, é um quadriculado de cores, onde cada quadradinho deste quadriculado é chamado de *pixels*. Para compor uma imagem cada pixel é colorido de uma cor diferente.

A dimensão de uma imagem será definida pela quantidade de *pixels* na altura e na largura do quadriculado que define a área da imagem. Quanto mais *pixels* por área melhor a qualidade de imagem para a visão humana, maior a capacidade de zoom.

O **banco de imagens** bruto, que será o material de um problema de processamento de imagens, nem sempre está pronto para ser processado pelo computador. As imagens em geral não estão todas na mesma dimensão e, além disso, o formato jpeg e png, ou qualquer outro formato de visualização para humanos, não é adequado para a leitura dos modelos. Para que seja possível aplicar os métodos estatísticos, de classificação para categorizar as imagens, por exemplo, é necessário transformar o banco de imagens em um banco estruturado, isto é, um banco numérico. Esse processo, de transformar um quadriculado de pixels coloridos em um banco numérico é o que chamamos de leitura e redimensionamento de imagens.

O **problema** de leitura e redimensionamento de imagens consiste em ajustar as imagens para uma mesma dimensão de altura e largura e em seguida definir uma estrutura de dados numérica para representá-las.

Se as imagens forem em preto e branco a estrutura de dados utilizada será uma matriz de dimensão igual a largura e altura do quadriculado de *pixels*. O valor guardado na posição (i, j) será um número de 0 até 255 que representa uma tonalidade de cinza, em que o 0 é a cor preta e o 255 a cor branca. Vale destacar que o *pixel* mais a esquerda e mais acima tem a posição $(0, 0)$. Conforme caminhamos para a direita a abcissa i cresce e conforme caminhamos para baixo da imagem a coordenada j cresce.

Se as imagens forem coloridas estrutura de dados utilizada será equivalente a três matrizes, cada uma delas de dimensão igual a largura e altura do quadriculado de *pixels*. Cada uma das três matrizes guarda um número de 0 até 255 que representa a quantidade das cores vermelha, verde e azul, respectivamente, de acordo com com a escala (R,G,B) - *red, green and blue*.

No caso de trabalharmos com a linguagem de programação R [7] cada imagem será representada por um objeto do tipo `array` tridimensional que guarda na posição (i, j, k) um valor entre 0 e 255 referente ao *pixel* de posição (i, j) da matriz k.

Veja que os valores de i, j e k variam da seguinte forma: $i = 1, \ldots, h$, $j = 1, \ldots, w$ e $k = 1, 2$ e 3, sendo h a quantidade de *pixels* na altura da imagem e w a quantidade de *pixels* na largura da imagem. Se as imagens forem em preto e branco $k = 1$.

A função apresentada no próximo *script*, nomeada `info`, realiza a leitura de um banco de imagens pelo programa R. Isto é, ela transforma as imagens, arquivos jpg ou png, em objetos estruturados e reconhecidos pelo programa R [7]. Essa função recebe os seguinte objetos como argumento de entrada:

- `file_path`: o endereço do diretório onde estão as imagens, objeto da classe `character`;
- `image_height`: a nova dimensão de altura para todas as imagens, objeto da classe `numeric`.
- `image_width`: a nova dimensão de largura para todas as imagens, objeto da classe `numeric`.

A saída da função `info` será uma lista com dois objetos. O primeiro objeto da lista é do tipo `data.frame` com `N` linhas e 3 colunas, sendo `N` o número de imagens no banco. As colunas guardam as informações de `Id`, altura e largura originais das imagens.

O segundo objeto da lista de saída é uma outra lista de dimensão `N`. Cada elemento desta lista é um `array` que guarda as informações de cada imagem redimensionadas para altura igual a `image_height` e largura igual a `image_width`. Se as imagens originais forem em preto e branco, cada elemento dessa lista será um `array` de dimensão `(image_height,image_width,1)`. Se imagens originais forem coloridas, cada elemento dessa lista será um `array` de dimensão `(image_height,image_width,3)`.

A seguir o *Script 1* com a implementação da função `info`. Veja que para esse *script* rodar é preciso ter instalado os pacotes `"stringr"` [9], `"OpenImageR"` [5, 2, 1] e `"EBImage"`[6]. Para instalar o pacote `"EBImage"` pela primeira vez use as duas linhas de comando a seguir.

```
install.packages("BiocManager")
BiocManager::install("EBImage")
```

```
library(stringr)
library(OpenImageR)
library(EBImage)


info <- function(file_path, image_height,image_width){

  image_names <- list.files(file_path)
  image_names <- image_names[which(image_names!="desktop.ini")]

  # Criando variaveis

  image_id <- vector("character")
  image_height_v <- vector("numeric")
  image_width_v <- vector("numeric")
  lista_imagens <- list()
  lista_final <-list()

  for (j in 1:length(image_names)) {

    # Caminho da imagem
    image_id[length(image_id)+1] <-
      paste0(str_remove(image_names[j],".jpg"))

    # Lendo imagem
    image <- readImage(paste0(file_path,"/",image_names[j]))

    # Dimensão da imagem
    image_height_v[length(image_height_v) + 1] <- dim(image)[1]
    image_width_v[length(image_width_v) + 1] <- dim(image)[2]

    image <- EBImage::resize(x=image,h=image_height,w=image_width)
    if(length(dim(image))==2){
      null_data  = rep(0,image_height*image_width)
      imagem_array_3D = array(null_data, c(image_height,image_width,1))
      imagem_array_3D[,,1] = image
      lista_imagens[[j]] <- image
    } else {
      lista_imagens[[j]] <- image
    }
  }

  BD1 <- data.frame(image_id,image_height_v,image_width_v)

  lista_final[[1]] <- BD1
  lista_final[[2]] <- lista_imagens

  return(lista_final)
```

```
}
```

Divisão do Banco em Treino, Validação e Teste

Para melhor aproveitamento e análise dos resultados pelos métodos de classificação adotados é importante ter um banco de treino e validação, conjuntos de imagens utilizados para treinar o modelo, e um banco de teste, conjunto de imagens para medir a capacidade de classificação para novas imagens. Nessa seção veremos como fazer isso.

O segundo *script* apresentado neste relatório (*Script 2*) contém a implementação da função `Treino_Teste_Valid`, que realiza a tarefa de dividir o banco de imagens em três subconjuntos: treino, teste e validação. Essa divisão é feita de forma aleatória e sem reposição.

Os argumentos de entrada da função `Treino_Teste_Valid` são:

- `Tabela_info`: saída da função `info` explicada anteriormente, objeto da classe `list`;
- `Vetor_class`: vetor de rótulos das imagens contidas no banco, que deve ser pariado com o objeto `Tabela_info`, isto é, a posição i do `Vetor_class` guarda o rótulo da imagem i de `Tabela_info`;
- `Prop`: proporção de separação do banco, objeto de da classe `numeric` de tamanho 3 contendo na primeira posição a proporção de imagens no banco de treino, na segunda posição a proporção de imagens no banco de teste e na terceira posição a proporção de imagens no banco de validação. Cada um dos três valores do vetor `Prop` deve ser um número não negativo e a soma dos três tem que resultar em 1. Caso esse elemento não seja informado será adotado o valor padrão `c(0.6, 0.2, 0.2)`, que indica 60% das imagens para o banco de treino, 20% para o banco de teste e 20% para o banco de validação.
- `COR`: indica se as imagens são coloridas, nesse caso `COR=T`, ou em preto e branco, `COR=F`, objeto do tipo `boolean`. Caso essa entrada não seja informada será entendido que o banco é formado por imagens coloridas.
- `BALANCEAMENTO`: indica se a partição em treino, teste e validação deve se preocupar com o valanceamento dos rótulos, se `BALANCEAMENTO=T` cada um dos três bancos será composto pela mesma quantidade de imagens de cada rótulo, se `BALANCEAMENTO=F` isso não será necessarimaente verdade.

A saída da função `Treino_Teste_Valid` será uma lista de tamanho três. O primeiro objeto da lista se refere ao banco de treino, o segundo ao banco de teste e o terceiro e último ao banco de validação. Cada um dos três objetos é uma nova lista com duas posições, vamos chamá-las de listas de treino, teste e validação. Considere `n` o tamanho do banco de treino, teste ou validação.

A primeira posição da lista de treino, teste ou validação é um `array` com as informações das imagens no banco em questão. Se as imagens originais forem em preto e branco, esse `array` tem dimensão `(n,image_height,image_width,1)`. Se imagens originais forem coloridas, esse `array` tem dimensão `(n,image_height,image_width,3)`.

A segunda posição da lista de treino, teste ou validação é um vetor de tamanho `n` com os rótulos de cada imagem no banco em questão.

```
Treino_Teste_Valid <- function(Tabela_info,
                               Vetor_class,
                               Prop = c(0.6,0.2,0.2),
                               COR = T,
                               BALANCEAMENTO = T){

  # Numero de imagens no diretorio
  N_Imagens <- dim(Tabela_info[[1]])[1]
  # Dimensões da imagem
  Dimensao1 <- dim(Tabela_info[[2]][[dim(Tabela_info[[1]])[1]]])[1]
  Dimensao2 <- dim(Tabela_info[[2]][[dim(Tabela_info[[1]])[1]]])[2]
  chanels = ifelse(COR,3,1)
```

```
# Fazer a divisao treino, teste e validacao

if(BALANCEAMENTO){

  #primeiro calcular a quantidade de imagem em cada banco de forma a fazer o balanceamento
  indices = 1:N_Imagens
  dados_imagens = cbind(indices,Tabela_info[[1]],Vetor_class)
  rotulos = names(table(Vetor_class))
  quantidade_por_rotulo = as.numeric(table(Vetor_class))
  indice_do_rotulo_com_menos_quantidade = which.min(quantidade_por_rotulo)
  rotulo_com_menos_quantidade = rotulos[indice_do_rotulo_com_menos_quantidade]
  quantidade_do_rotulo_com_menos_quantidade = quantidade_por_rotulo[indice_do_rotulo_com_menos_quantic

  #determinar quantidade de cada rotulo (que sera a mesma) em cada banco
  #TREINO
  n_por_rotulo_treino = (Prop[1]*quantidade_do_rotulo_com_menos_quantidade)%/%1
  #TESTE
  n_por_rotulo_teste = (Prop[2]*quantidade_do_rotulo_com_menos_quantidade)%/%1
  #VALID
  n_por_rotulo_valid = quantidade_do_rotulo_com_menos_quantidade - n_por_rotulo_teste - n_por_rotulo_

  #separar os indices por rotulo para sorteio do treino
  lista_indices_por_rotulo = list()
  for(r in 1:length(rotulos)){
    lista_indices_por_rotulo[[r]] = dados_imagens[Vetor_class==rotulos[r],"indices"]
  }
  #sorteio do treino
  obs_treino = NULL
  for(r in 1:length(rotulos)){
    obs_treino = c(obs_treino,
                   sample(lista_indices_por_rotulo[[r]],
                          size=n_por_rotulo_treino,
                          replace=F))
  }

  #separar os indices por rotulo para sorteio do treino
  dados_imagens_menor = dados_imagens[-obs_treino,]
  Vetor_class_menor = dados_imagens_menor[,"Vetor_class"]
  lista_indices_por_rotulo = list()
  for(r in 1:length(rotulos)){
    lista_indices_por_rotulo[[r]] =
      dados_imagens_menor[Vetor_class_menor==rotulos[r],"indices"]
  }
  #sorteio do teste
  obs_teste =  NULL
  for(r in 1:length(rotulos)){
    obs_teste = c(obs_teste,
                  sample(lista_indices_por_rotulo[[r]],
                         size=n_por_rotulo_teste,
                         replace=F))
  }
  #separar os indices por rotulo para sorteio da validacao
  dados_imagens_menor = dados_imagens[-sort(c(obs_treino,obs_teste)),]
```

```
  Vetor_class_menor = dados_imagens_menor[,"Vetor_class"]
  lista_indices_por_rotulo = list()
  for(r in 1:length(rotulos)){
    lista_indices_por_rotulo[[r]] =
      dados_imagens_menor[Vetor_class_menor==rotulos[r],
                          "indices"]
  }
  #sorteio da valid
  obs_valid =  NULL
  for(r in 1:length(rotulos)){
    obs_valid = c(obs_valid,
                  sample(lista_indices_por_rotulo[[r]],
                         size=n_por_rotulo_valid,
                         replace=F))
  }
} else {
  indices = 1:N_Imagens
  dados_imagens = cbind(indices,Tabela_info[[1]],Vetor_class)

  #determinar quantidade de cada rotulo (que sera a mesma) em cada banco
  n_treino = (Prop[1]*N_Imagens)%/%1
  #TESTE
  n_teste = (Prop[2]*N_Imagens)%/%1
  #VALID
  n_valid = N_Imagens - n_teste - n_treino


  #sorteio do treino
  obs_treino = NULL
  obs_treino = sample(indices,size=n_treino,replace=F)

  #separar os indices por rotulo para sorteio do treino
  dados_imagens_menor = dados_imagens[-obs_treino,]
  indices_menor = dados_imagens_menor[,"indices"]
  obs_teste = sample(indices_menor,size=n_teste,replace=F)

  #separar os indices por rotulo para valid
  dados_imagens_menor = dados_imagens[-c(obs_treino,obs_teste),]
  indices_menor = dados_imagens_menor[,"indices"]
  obs_valid = sample(indices_menor,size=n_valid,replace=F)
}
#Separacao do banco de imagens
#TREINO
null_data   = rep(0, length(obs_treino)*Dimensao1*Dimensao2*chanels)
x_train     = array(null_data,
                    c(length(obs_treino),Dimensao1,Dimensao2,chanels))
k=1
for(i in obs_treino){
  if(COR){
    x_train[k,,,] = Tabela_info[[2]][[i]]
  } else {
    if(length(dim((Tabela_info[[2]][[i]])))==2){
      x_train[k,,,1] = Tabela_info[[2]][[i]]
```

```
    } else {
      #cat("Aviso: imagem ",Tabela_info[[1]][i,1]," esta salva como colorida quando deveria ser P&B.\
      x_train[k,,,1] = Tabela_info[[2]][[i]][,,1]
    }
  }
  k=k+1
}

y_train = Vetor_class[obs_treino]


train = list()
train[[1]] = x_train
train[[2]] = y_train
names(train) = c("x","y")

#TESTE
null_data    = rep(0, length(obs_teste)*Dimensao1*Dimensao2*chanels)
x_test       = array(null_data,
                     c(length(obs_teste),Dimensao1,Dimensao2,chanels))
k=1
for(i in obs_teste){
  if(COR){
    x_test[k,,,] = Tabela_info[[2]][[i]]
  } else {
    if(length(dim((Tabela_info[[2]][[i]])))==2){
      x_test[k,,,1] = Tabela_info[[2]][[i]]
    } else {
      #cat("Aviso: imagem ",Tabela_info[[1]][i,1]," esta salva como colorida quando deveria ser P&B.\
      x_test[k,,,1] = Tabela_info[[2]][[i]][,,1]
    }
  }
  k=k+1
}

y_test = Vetor_class[obs_teste]


test = list()
test[[1]] = x_test
test[[2]] = y_test
names(test) = c("x","y")


#VALIDACAO
null_data   = rep(0, length(obs_valid)*Dimensao1*Dimensao2*chanels)
x_valid     = array(null_data, c(length(obs_valid),Dimensao1,Dimensao2,chanels))

k=1
for(i in obs_valid){
  if(COR){
    x_valid[k,,,] = Tabela_info[[2]][[i]]
  } else {
```

```
      if(length(dim((Tabela_info[[2]][[i]])))==2){
        x_valid[k,,,1] = Tabela_info[[2]][[i]]
      } else {
        #cat("Aviso: imagem ",Tabela_info[[1]][i,1]," esta salva como colorida quando deveria ser P&B.\
        x_valid[k,,,1] = Tabela_info[[2]][[i]][,,1]
      }
    }
    k=k+1
  }

  y_valid = Vetor_class[obs_valid]


  valid = list()
  valid[[1]] = x_valid
  valid[[2]] = y_valid
  names(valid) = c("x","y")

  # Construcao objeto final para rodar o modelo

  base_dividida = list()
  base_dividida[[1]] = train
  base_dividida[[2]] = test
  base_dividida[[3]] = valid

  names(base_dividida) = c("train", "test", "valid")

  return(base_dividida)
}
```

Classificação de Imagens

O Banco de Dados

O **banco de dados** do problema de classificação de imagem é formado por imagens com objetos de um certo tipo e juntos às imagens são associados rótulos que indicam o tipo do objeto da imagem em questão.

O **problema** a ser resolvido é encontrar um classificador capaz de melhor classificar as imagens, isto é, a partir de uma imagem qualquer o classificador deve ser capaz de acertar o rótulo da imagem.

Os rótulos podem ser binários, por exemplo se indicam que a imagem contém ou não um certo item, ou com mais de duas classe, por exemplo imagens de apenas um objeto onde o rótulo indica o objeto na imagem. O classificador deve acertar o tipo de classe de cada imagem.

Redes Neurais

As Redes Neurais artificiais são modelos computacionais inspirados no funcionamento do sistema nervoso de seres vivos, que são compostos, principalmente, por neurônios biológicos ou células nervosas, responsáveis pela caracterização do sistema. Os neurônios têm como papel principal a recepção e transmissão de sinais, permitindo assim que haja respostas aos estímulos recebidos.

As Figuras 1 e 2 a seguir apresentam uma comparação entre as estruturas de um neurônio biológico e um neurônio artificial. O neurônio biológico (Figura 1) apresenta três partes principais: os dendritos, responsáveis pela captação contínua dos estímulos vindos dos diversos outros neurônios ou do próprio meio; o corpo celular, responsável por processar toda a informação vinda dos dendritos; e o axônio, que tem a responsabilidade de conduzir os sinais para outros neurônios.

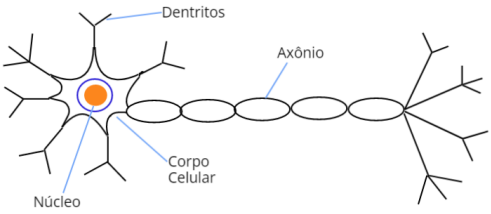


Figure 1: Neurônio biológico.

Assim como no caso biológico, o modelo de neurônio artificial coleta os sinais de entrada, processa esses sinais e conduz outros sinais em sua saída. As redes neurais trabalham de forma semelhante, onde as camadas ocultas são responsáveis pelo processamento dos sinais recebidos na camada de entrada, e são formadas pelos neurônios artificiais. Também é importante destacar que a saída de cada neurônio está ligada a entrada de outro neurônio na camada seguinte.

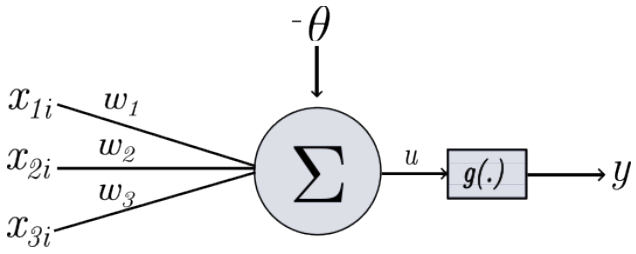


Figure 2: Neurônio artificial.

A estrutura do neurônio artificial apresentado na Figura 2 foi proposta por [4]. Nela são destacados os principais elementos de um neurônio artificial:

- Sinais de Entrada: $x_{1i}, x_{2i}, \ldots, x_{pi}$. Vetor de covariáveis referentes às características de interesse.
- Pesos sinápticos: $w_1, w_2, \ldots, w_p$. Valores que irão ponderar cada uma das variáveis de entrada da rede (parâmetros desconhecidos a serem estimados).
- Combinador Linear: Σ. Agregador dos sinais de entrada ponderados por seus respectivos pesos.
- Limiar de ativação: $-\theta$. Variável que especifica qual será o limite para que o valor produzido pelo combinador linear possa gerar um valor de disparo.
- Função de Ativação: $g(.)$. Função responsável por receber os sinais de entrada após terem passado pelo combinador linear e aplica uma transformação não linear, que é enviada para a próxima camada da rede neural.

Perceptron Para entender o funcionamento das redes neurais é interessante conhecer sua arquitetura mais simples, para entendimento de conceitos e terminologias. O *Perceptron* é um modelo de rede neural, proposto por [8], onde sua estrutura é composta por apenas uma camada neural e um neurônio artificial. O modelo Perceptron é utilizado em problemas com dados linearmente separáveis (Figura 3).

No modelo Perceptron, a camada de entrada recebe as P covariáveis, que são combinadas linearmente com os pesos sinápticos e adicionadas de um único neurônio presente na camada de saída. O resultado desse processo passa pela função de ativação que é responsável pela saída do modelo, veja Equação 1.

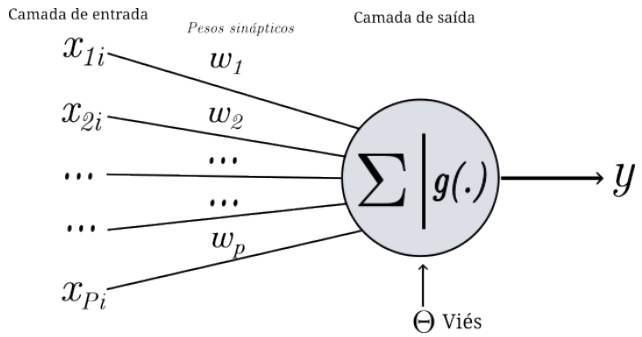


Figure 3: Arquitetura do modelo *Perceptron*.

$$y_i \;=\; g\left(\sum_{p=1}^{P} w_p x_{pi} \;+\; \Theta\right),\; i=\; 1,2,\ldots,n \tag{1}$$

***Percepton* Multicamadas** Visando trabalhar com problemas onde as variáveis de interesse não são linearmente separáveis, possuem mais de duas classes ou até mesmo são contínuas, foi desenvolvido o modelo *Perceptron* Multicamadas, que tem como característica principal a presença de mais de uma camada computacional, essas novas camadas são chamadas de camadas escondidas, já que os cálculos realizados nessas camadas não são visíveis para o usuário.

Como apresentado nas Figuras 4 e **??** a seguir, na arquitetura de redes neurais multicamadas todas as camadas se conectam em direção a camada de saída. Sendo assim todos os neurônios de cada camada estão conectados aos neurônios da camada seguinte.

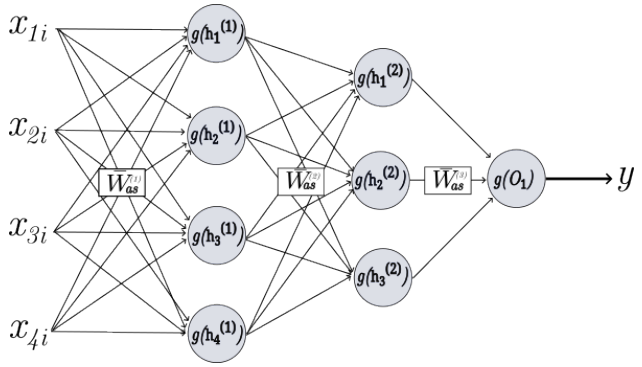


Figure 4: Percepton multicamadas com uma saída.

Podemos escrever que entra no neurônio s da camada 1 o valor de $h_s^{(1)}$: $h_s^{(1)} = \sum_{\forall a} x_{a,i} W_{a,s}^{(1)}$. Sai do neurônio s da camada (1) o valor de $g\left(h_s^{(1)} + \Theta_1\right)$: $g\left(h_s^{(1)} + \Theta_1\right) = g\left(\sum_{\forall a} x_{a,i} W_{a,s}^{(1)} + \Theta_1\right)$. Este valor vai compor as entradas dos neurônios da camada (2). Entra no neurônio s da camada (2) $h_s^{(2)} = \sum_{\forall a} g\left(h_a^{(1)} + \Theta_1\right) W_{a,s}^{(1)}$.

e podemos escrever $h_a^{(L+1)} = \sum_{\forall j} g(h_a^{(L)} + \Theta_j)$

O que diferencia a estruturas apresentada na Figura 4 daquela apresentada na **??** é o número de saídas. Quando temos um problema com uma única saída, como por exemplo um problema de classificação binária, a estrutura adequada é a da esquerda, para o exemplo citado a saída será a probabilidade da unidade amostral pertencer a uma das classes. Já se o problema conta com mais saídas, como por exemplo o problema de classificação com mais de duas classes, a estrutura adequada é a da direita, para o exemplo citado cada uma das saídas retorna a probabilidade da unidade amostral pertencer a cada uma das classes.

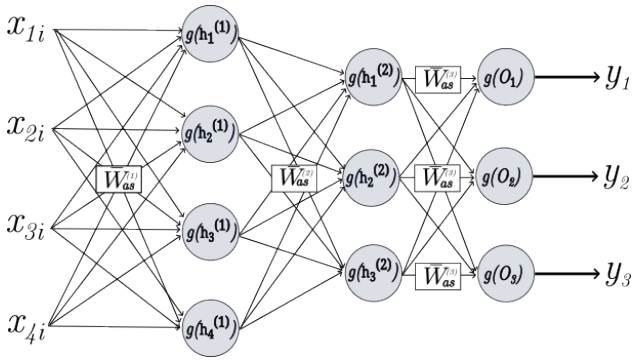


Figure 5: Percepton multicamadas com três saídas.

Função de Ativação São muitas as possibilidades de funções de ativação e a escolha de qual usar depende do problema a ser resolvido. Aqui serão apresentadas três delas.

A primeira será a Função Degrau:

$$g(x) = \begin{cases} 1\ , & \text{se } x\ > 0 \\ 0, & \text{caso contrário.} \end{cases}$$

Essa função é usada quando se quer uma resposta binária. A segunda função de ativação apresentada é a Função Sigmoidal:

$g(x) = \frac{1}{1+e^{-(x)}}$.

Vale ressaltar que está é a Função Logística e é usada quando se quer retornar a probabilidade de pertencer a classe 1.

Ainda tem a Função Softmax, baseada no Modelo de Regressão Multinomial, que é uma generalização do Modelo de Regressão Logística, e retorna a probabilidade de cada classe ser a correta.

$$g(x)_j = \frac{e^{\bar{x_i}}}{\sum_{j=1}^{J} e^{x_i \bar{j}}}.$$

A Função Softmax é uma combinação de funções sigmoidais, porém utilizada em problemas de classificação onde existem mais de duas classes, tendo como saída as probabilidades de cada particular classe da variável de interesse.

Redes Neurais Convolucionais

As redes neurais convolucionais foram desenvolvidas para trabalhar com dados de entrada estruturados em um *grid* com fortes dependências espaciais. O exemplo mais óbvio de dados estruturados em *grid* é uma imagem bidimensional.

O que caracteriza as redes neurais convolucionais é a operação de convolução, que trata-se de uma operação de produto escalar entre um conjunto de pesos estruturado em *grid*. Este tipo de operação é útil para dados com um elevado nível de espacialidade, que é o caso de dados de imagem. Sendo assim, as redes neurais convolucionais são definidas como redes que utilizam a operação de convolução em pelo menos uma camada, embora a maioria das redes neurais convolucionais utilizem esta operação em múltiplas camadas (@aggarwal2018neural).

Cada camada da rede convolucional é uma estrutura de *grid* tridimensional, que tem uma altura, largura e profundidade. A palavra "profundidade" refere-se ao número de canais em cada nível, como por exemplo o número de cores primárias (azul, verde e vermelho) na imagem de entrada ou o número de mapas nas camadas ocultas. As camadas mais comuns em uma rede neural convolucional, que serão detalhadas neste relatório, são: Camada Convolucional, *pooling*, *dropout* e *fletten*.

Camada Convolucional A camada convolucional (Figura 6) é geralmente a primeira camada de um CNN. Nesta camada o *grid* que representa a imagem é *varrido* por quadrados, de dimensão 2×2, 3×3 ou 4×4, por exemplo, e para cada quadrado da imagem é definido um novo valor que será atribuído ao *grid* de saída que tem menor dimensão do que o de entrada. Veja a Figura 6, onde esse processo é apresentado.

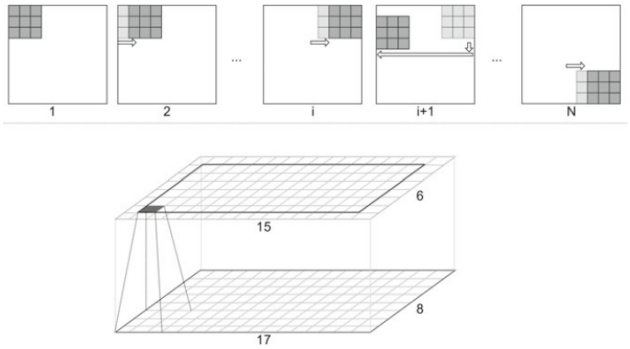


Figure 6: Exemplo de Camada Convolucional (fonte: @skansi2018introduction).

O valor atribuído ao elemento do novo *grid* de saída é definido por uma regressão onde o número de covariáveis é o número de elementos dentro do quadrado de varredura do *grid* de entrada. Esse processo é feito sobre toda a imagem, caminhando em passos de tamanho dado pelo parâmetro *stride*. No exemplo apresentado da figura anterior foi usado *stride*=1, mas pode ser usado outros valores, como por exemplo *stride*=2 como mostra a Figura 7.

[h]

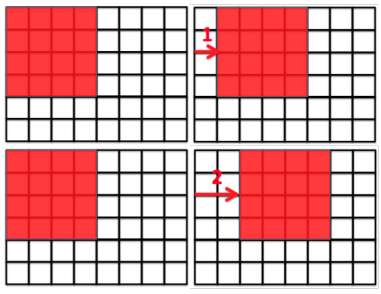


Figure 7: Exemplo de *Stride* = 1 e *Stride* = 2 (fonte: Medium.com)

A dimensão do *grid* de saída é menor que o de entrada e essa redução depende do tamanho do quadrado de varredura e do tamanho do passo dado (*stride*).

Além disso essa camada ainda permitem definir o número de filtros, que representa o número de *grids* de saída da camada. A Figura 8 seguir mostra um exemplo de camada convolucional com filtro = 6 e um único *grid* de entrada, que é o caso das imagens em preto e branco.

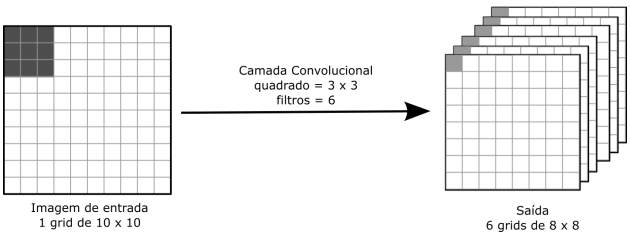


Figure 8: Camada Convolucional, imagem em preto e branco e filtro = 1.

A Figura 9 seguir mostra um exemplo de camada convolucional com filtro = 6 e 3 *grids* de entrada, que é o

caso das imagens coloridas.

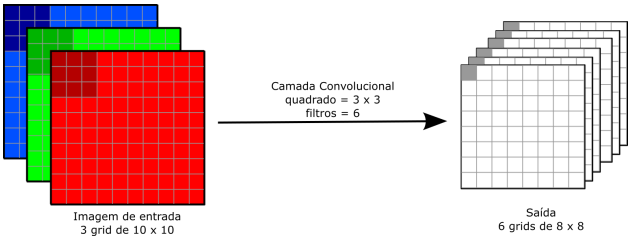


Figure 9: Camada Convolucional, imagem colorida e filtro = 1.

Os parâmetros da regressão citada serão parâmetros (pesos) desconhecidos da camada e serão estimados pelo treinamento da rede. O número de parâmetros a serem estimados em cada uma dessas camadas depende do tamanho do quadrado de varredura, do valor do filtro e do número de canais da imagem. Por exemplo, se a camada convolucional é aplicada em uma imagem em preto e branco (1 canal) considerando um quadrado 2×2 e filtro $= 10$ o número de parâmetros a serem estimados é $((1 \times 2 \times 2) + 1) \times 10 = 50$. Se a imagem for colorida (3 canais), pensando ainda em um quadrado de varredura de dimensão 2×2 com filtro $= 10$, o número de parâmetros a serem estimados será $((3 \times 2 \times 2) + 1) \times 10 = 130$. Um último exemplo, considere uma imagem colorida com quadrados de varredura de tamanho 3×3 e filtro $= 3$, o número de parâmetros será $((3 \times 3 \times 3) + 1) \times 3 = 84$.

Para essa camada ainda é possível incluir uma função de ativação. O comum é usar a função ReLu.

Camada *pooling* Na camada *pooling* a seguinte operação é feita. Novamente um quadrado de tamanho pré-estipulado (2×2 ou 3×3, por exemplo) varre cada canal de *grid* da imagem e produz um novo *grid*, com mesmo número de canais, porém menor dimensão.

A camada *pooling* mais usada é a *max-pooling*. Nesta camada para cada quadrado é claculado o valor máximo e esse será o valor atribuído ao *grid* de saída. Outros tipos de *pooling* podemos ser executados, como o *average-pooling*, que atribuí à camada de saída o valor médio do quadrado, por exemplo. Mas *poolings* diferentes do *max-pooling* são raramente encontrados (@aggarwal2018neural).

A Figura 10 a seguir mostra um exemplo de camada *max-pooling* 3×3 com *stride* $= 1$.

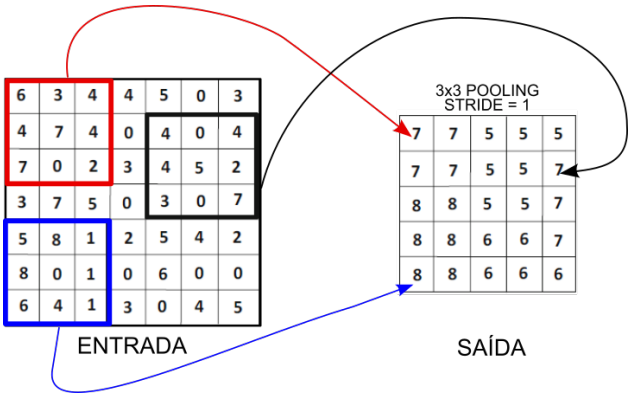


Figure 10: Exemplo de camada de 3x3 *max-pooling* com *stride*=1.

Já Figura 11 a seguir mostra um exemplo de camada *max-pooling* 3×3 com *stride* $= 2$.

As camadas *pooling* não definem parâmetros a serem estimados.

Camada *dropout* A camada *dropout* tem como objetivo diminuir o problema de sobreposição (*overfitting*). O *dropout* previne o *overfitting* e proporciona uma forma de combinar exponencialmente e de forma eficiente

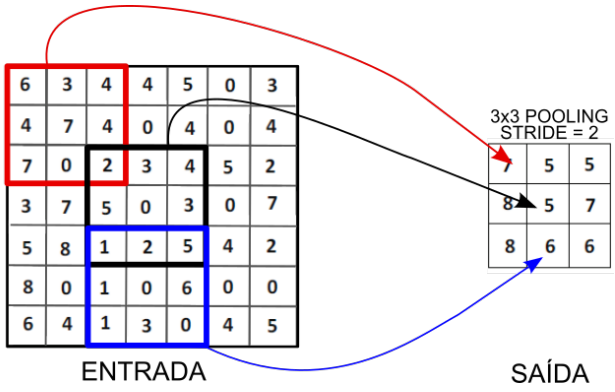


Figure 11: Exemplo de camada de 3x3 *max-pooling* com *stride*=2.

diferentes arquitecturas de redes neurais. O termo "*dropout*" refere-se ao desligamento de alguns neuronios da rede, como mostra a Figura 12. A escolha dos neurônios a serem desligados é feita de forma aleatória. No caso mais simples, cada neurônio é desligado com uma probabilidade p independente de forma independente de outros neurônios. É comum escolher p entre 0.2 e 0.5 (@aggarwal2018neural), porém, segundo @srivastava2014dropout, $p = 0.5$ é uma escolha interessante por estar, em geral, perto da escolha ótima.

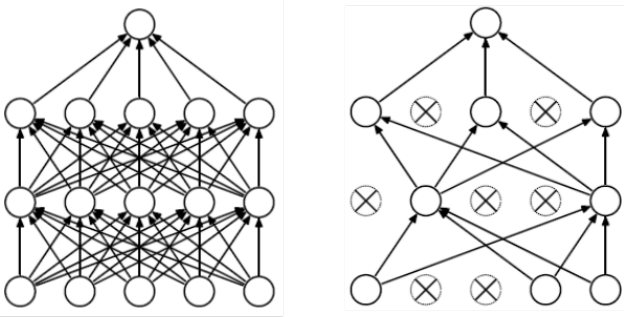


Figure 12: (a) Uma rede neural padrão com 2 camadas ocultas. (b) A mesma rede porém realizada a aplicação de *dropout*. (fonte: @srivastava2014dropout).

Camada *flatten* É nesta camada que os dados em forma de *grid* são re-organizados para o formato vetorial, números em fila. A entrada dessa camada é um *grid* de dimensão $n \times n$, por exemplo, e a saída serão $n \times n$ valores. Camadas adicionais virão depois desta a fim de diminuir o número de saídas.

Esta também é uma camada sem parâmetros para serem estimados.

Camada *deep* Depois da camada *flatten* os dados não são mais em forma de *grid* e sim em forma vetorial. Camadas adicionais, como as usadas em redes neurais simples, com uma função de ativação, podem ser adicionadas. O importante é usar essas camadas para diminuir o número de valores de saída.

Camada de saída A camada de saída é a última camada de uma rede neural e define o número de valores de saída da rede. Quando se trata de um problema de classificação, a rede neural retorna é um número entre 0 e 1 para cada classes, que indica a probabilidade da observação de entrada pertencer a classe em questão. Para esse padrão de saída é usada a função de ativação *softmax*.

Esta camada vem geralmente depois da camada *flatten* ou de uma camada *deep*. O número de parâmetros a serem estimados nesta camada é definido da seguinte forma: sejam n a dimensão de saída da camada anterior e k o número de classes do problema de classificação, o número de parâmetros será $k \times (n + 1)$

A imagem a seguir representa uma CNN com as camadas explicadas neste relatório.

Como Treinar uma CNN no R

O treinamento de uma CNN no R (@R) é feito através dos pacotes `caret` (@caret), `tensorflow` (@tensorflow) e `keras` (@keras). Por isso primeiro é necessário instalar e carregar tais pacotes.

```
library(caret)
library(tensorflow)
library(keras)
```

A sequência de códigos apresentado a seguir cria uma rede neural para realizar a classificação de um banco de imagens cujas imagens estão no formato jpg ou png guardadas no diretório nomeado como `local`. O objeto `local` é do tipo `character` e indica um endereço de diretório do computador, como por exemplo, `"C:/Users/PC/Documents/imagens"`, atenção para o padrão de barras invertidas usado pelo R.

O primeiro passo é a leitura das imagens feita pelo R. Isso será feito a partir da função `info` já apresentada. As variáveis `h` e `w` guardam a altura e largura que as imagens terão depois de redimensionadas.

```
banco   = info(file_path=local,image_height = h,image_width = w)
```

Após esse passo a variável banco guarda uma lista com dois objetos, um `data.frame` e uma lista de `array`. O `data.frame` guarda as informações da imagens e a lista guarda os `arrays` com os `grids` que definem a(s) camada(s) de cor(es) de cada imagem do banco.

Em seguida serão construídos os bancos de treino, validação e teste a partir da função `Treino_Teste_Valid`. Para construir bancos balanceados basta colocar `BALANCEAMENTO = T`. Ao argumento de entrada `COR` será atribuído o valor `F` se as imagens do banco for em preto e branco e `T` se as imagens forem coloridas. Atenção: antes de chamar a função `Treino_Teste_Valid` como descrita a seguir é preciso criar um vetor de rótulos, que no código está definido pela variável `rotulos`.

```
base = Treino_Teste_Valid(Tabela_info = banco,
                          Vetor_class = rotulos,
                          Prop = c(0.6,0.2,0.2),
                          COR = F,
                          BALANCEAMENTO = T)
```

O objeto base ao final do código anterior é uma lista de tamanho três, cada uma das posições desta lista guarda uma outra lista, de tamanho 2, com as bases de treino, teste e validação, nesta ordem.

A primeira posição da lista que define cada base é um `array` de dimensão dimensão `(n,h,w,c)`, onde `n` é o número de imagens na base (de treino, teste ou validação), `h` a altura de cada imagem, `w` a largura de cada imagem e `c` o número de canais (1 para imagens em preto e branco e 3 para imagens coloridas). A segunda posição da lista que define cada base é um vetor de tamanho `n` com os rótulos de cada imagem na base em questão.

As bases de treino e validação serão usadas para o treinamento da CNN. A construção dessa rede neural será feita de forma sequancial e apresentada no código a seguir. Depois do código seguem os comentários sobre ele.

```
model <- keras_model_sequential() %>%
  layer_conv_2d(
    kernel_size = c(3, 3),
    filter = 10,
    strides = 1,
    activation = "relu",
    input_shape = c(h, w, 1)
  ) %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_dropout(rate = 0.5) %>%
```

```
  layer_flatten() %>%
  layer_dense(units = num_de_classes, activation = 'softmax')
```

A primeira camada é definida pela função `layer_conv_2d`, e é uma camada convolucional. Para esse exemplo foi considerado um quadrado de varredura de dimensão 3×3, filtro= 10, tamanho do passo (*strides*) igual a 1 e função de ativação *ReLu*. Nesta função também informamos a dimensão dos *arrays* que definem as figuras quando incluímos `input_shape = c(h, w, 1)`.

A camada seguinte é uma camada *max-pooling* a partir da função `layer_max_pooling_2d`. Quando faz-se `pool_size = 2` indica-se que o quadrado de varredura é de dimensão 2×2. Depois desta camada foi adicionada uma camada *dropout* (função `layer_dropout`) e o argumento de entrada `rate = 0.5` indica que 50% das unidades a serão descartadas.

A camada definida pela função `layer_flatten` realiza a camada *fletten*, que transforma os dados em forma de *grid* para o formato vetorial. Por fim a última camada, criada a partir da função `layer_dense` realiza a camada de saída, usando a função de ativação `softmax` e criando o número de saídas igual ao valor guardado na variável `num_de_classes`, que deve ser o número de classes do problema de classificação.

Por fim, foi definido o compilador `"Adam"`, a função de perda `"sparse_categorical_crossentropy"` e a métrica da acurácia para a avaliação (`accuracy`), como mostra o código a seguir.

```
model %>% compile(
  optimizer = "adam",
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)
```

Feitos os comandos anteriores a CNN está definida e um resumo dela pode ser visto usando o comando `summary`. Para estimar os parâmetros da CNN utiliza-se a função `fit` e para avaliar as estimativas na base de treino a função `evaluate`.

```
history <- model %>%
  fit(
    x = base$train$x,
    y = base$train$y,
    validation_data = unname(base$valid)
  )

evaluate(model,base$test$x, base$test$y, verbose = 0)
```

Estudos de Casos

Nesta seção serão apresentados três estudos de casos. Cada um deles foi realizado por um dos alunos participantes do projeto de pesquisa que gerou este relatório. Nos estudos de casos serão usadas as funções implementadas nos `scripts` apresentados anteriormente.

Imagens de Raio-X

Estudo de Caso realizado por Matheus Alves Pereira Dos Santos. Neste estudo foi tratado o caso de classificação de imagens em preto e branco em duas categorias.

Banco de Dados

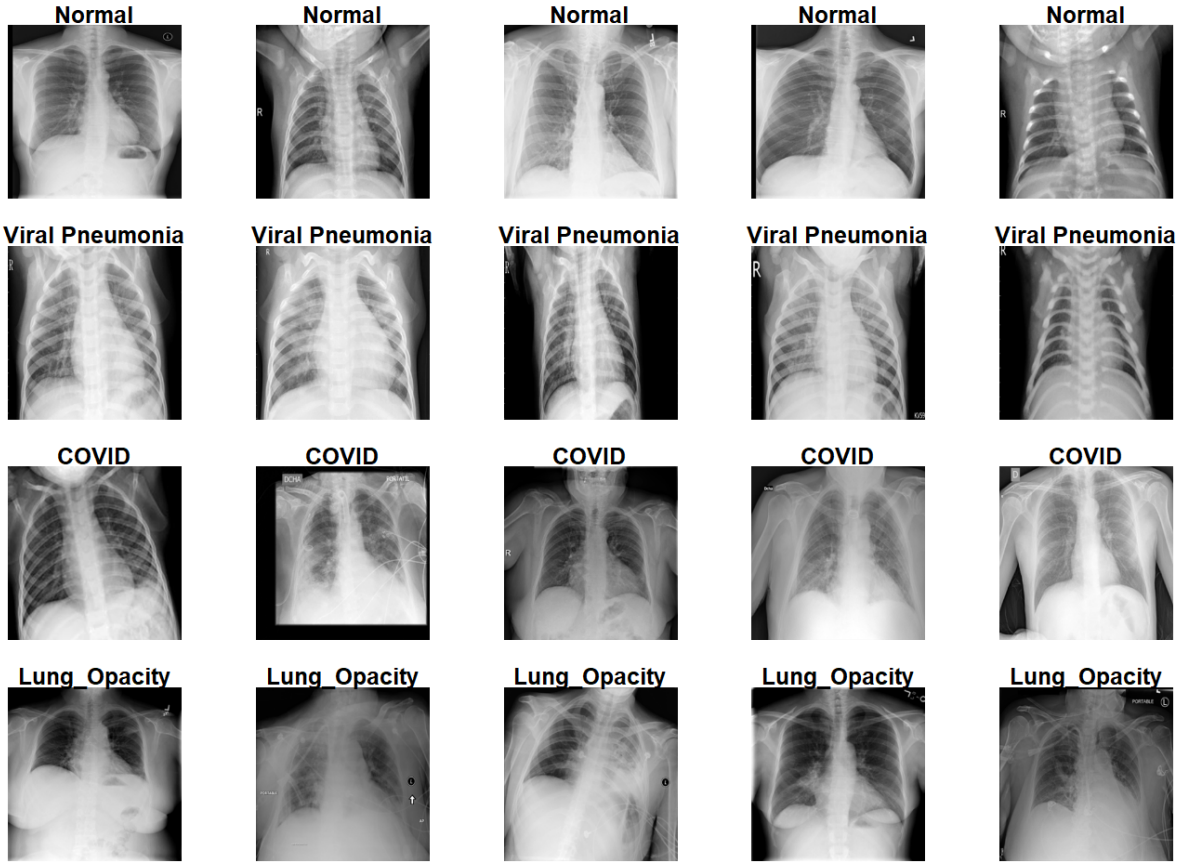
Este estudo utilizou um banco de radiografias disponível na plataforma Kaggle[2]. Esta base foi criada por uma equipe de pesquisadores da Universidade do Catar (QU) e da Universidade de Dhaka, juntamente com

[2]https://www.kaggle.com/preetviradiya/covid19-radiography-dataset (consultado em 04/08/2022)

os seus colaboradores médicos do Paquistão e da Malásia, que reuniram imagens de radiografias de tórax classificadas como COVID-19, opacidade pulmonar, normal e pneumonia viral.

O banco, baixado da plataforma em 04 de agosto de 2022, era composto por: 3.615 imagens de radiografias de tórax de pacientes com COVID, 6.012 imagens de radiografias de tórax com opacidade pulmonar, 10.191 imagens de radiografias de tórax de pacientes sem doença pulmonar diagnosticada (normal) e 1.345 imagens de radiografias de tórax de pacientes com pneumonia viral. Sendo assim, o banco completo continha 21.161 imagens de radiografia de tórax, todas no formato png.

As imagens estão organizadas em 4 diretórios: COVID, Lung_Opacity, Normal e Viral Pneumonia. Veja a seguir alguns exemplos de imagens deste banco, selecionadas de forma aleatória dentro de cada categoria.



Objetivo

O objetivo do estudo é criar um classificador capaz de identificar imagens em preto e branco rotuladas em duas ou três classes. Um dos desafios é a grande quantidade de imagens no banco original.

Esse estudo foi separado em três etapas.

Primeira etapa. Nesta etapa foram consideradas apenas duas classes, COVID e NORMAL. A base de dados desta etapa tem um total de 13.806 imagens: 3.615 rotuladas como COVID e 10.191 rotuladas como NORMAL.

Segunda etapa. Nesta etapa foram consideradas novamente duas classes, COVID e PNEUMONIA VIRAL. A base de dados desta etapa tem um total de 4.959 imagens: 3.615 rotuladas como COVID e 1.344 rotuladas como PNEUMONIA VIRAL.

Terceira etapa. Nesta etapa foram consideradas três classes, COVID, NORMAL e PNEUMONIA VIRAL. A base de dados desta etapa tem um total de 15.150 imagens: 3.615 rotuladas como COVID, 10.191 rotuladas

como NORMAL e 1.344 rotuladas como PNEUMONIA VIRAL.

Resultados

Antes de iniciar as etapas deste estudo a leitura e redimensionamento das imagens foi realizada a partir da função `info` definida anteriormente. Essa função foi chamada uma vez para cada diretório de interesse e depois os objetos de saída foram concatenados em um único objeto de acordo com os objetivos de cada etapa.

A seguir é apresentado o código para ler todas as imagens deste banco e redimensioná-las para 100 *pixels* de altura por 100 de largura. O objeto `local` é do tipo `"character"` e guarda o endereço do diretório onde estão as pastas COVID, Lung_Opacity, Normal e Viral Pneumonia. Por tratar-se de um banco grande, o processo é demorado e por isso optamos por salvar os dados em arquivos .Rdata para que não seja necessário repetir esse processo futuramente.

```
inicio = Sys.time()
h = 100
w = 100

local_covid   = paste0(local,"/COVID")
banco_covid   = info(file_path=local_covid,
                     image_height = h,
                     image_width = w)
save(banco_covid,file="banco_covid.Rdata")


local_normal  = paste0(local,"/Normal")
banco_normal  = info(local_normal,
                     image_height = h,
                     image_width = w)
save(banco_normal,file="banco_normal.Rdata")


local_pneumo  = paste0(local,"/Viral Pneumonia")
banco_pneumo  = info(local_pneumo,
                     image_height = h,
                     image_width = w)
save(banco_pneumo,file="banco_pneumo.Rdata")
fim = Sys.time()
fim - inicio
```

```
## Time difference of 5.163606 mins
```

Primeira Etapa Nesta etapa serão considerados as imagens dos diretórios NORMAL e COVID. As imagens categorizadas como NORMAL foram rotuladas com `0`, enquanto as imagens categorizadas como COVID foram rotuladas com `1`. É importante que os rótulos sejam números inteiros a partir do `0`.

```
num_de_classes = 2

n_normal = length(banco_normal[[2]])
n_covid = length(banco_covid[[2]])

banco = list()
banco[[1]] = rbind(banco_normal[[1]],banco_covid[[1]])
banco[[2]] = c(banco_normal[[2]],banco_covid[[2]])

rotulos = c(rep(0,n_normal),rep(1,n_covid))
```

Em seguida serão construídos os bancos de treino, validação e teste a partir da função `Treino_Teste_Valid`. Vamos construir bancos balanceados e lembrar que essas imagens são em preto e branco.

```
inicio = Sys.time()
base_etapa_1 = Treino_Teste_Valid(Tabela_info = banco,
                                  Vetor_class = rotulos,
                                  Prop = c(0.6,0.2,0.2),
                                  COR = F,
                                  BALANCEAMENTO = T)
fim = Sys.time()
fim - inicio
```

```
## Time difference of 2.089039 secs
```

Após a separação em treino, validação e teste as bases ficaram com as seguintes tamanhos: a base de treino com 4338 imagens, sendo 2169 de cada classe; a base de validação com 1446 imagens, 723 de cada classe; e a base de teste com 1448 imagens, 724 de cada classe. Algumas imagens foram desconsideradas devido a realização do balanceamento nas três bases deste estudo.

Uma vez concluída a construção das bases de treino, teste e validação parte-se para a definição da CNN e o seu treinamento, como já explicado no Capítulo de Materiais e Métodos.

```
inicio = Sys.time()
model <- keras_model_sequential() %>%
  layer_conv_2d(
    kernel_size = c(3, 3),
    filter = 10,
    strides = 1,
    activation = "relu",
    padding = "same",
    input_shape = c(h, w, 1)
  ) %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_dropout(rate = 0.5) %>%
  layer_flatten() %>%
  layer_dense(units = num_de_classes, activation = 'softmax')
```

```
## Loaded Tensorflow version 2.9.1
```

```
model %>% compile(
  optimizer = "adam",
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)
summary(model)
```

```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## =========================================================================
##  conv2d (Conv2D)                    (None, 100, 100, 10)             100
##  max_pooling2d (MaxPooling2D)       (None, 50, 50, 10)               0
##  dropout (Dropout)                  (None, 50, 50, 10)               0
##  flatten (Flatten)                  (None, 25000)                    0
##  dense (Dense)                      (None, 2)                        50002
## =========================================================================
## Total params: 50,102
```

```
## Trainable params: 50,102
## Non-trainable params: 0
## _________________________________________________________________________
```

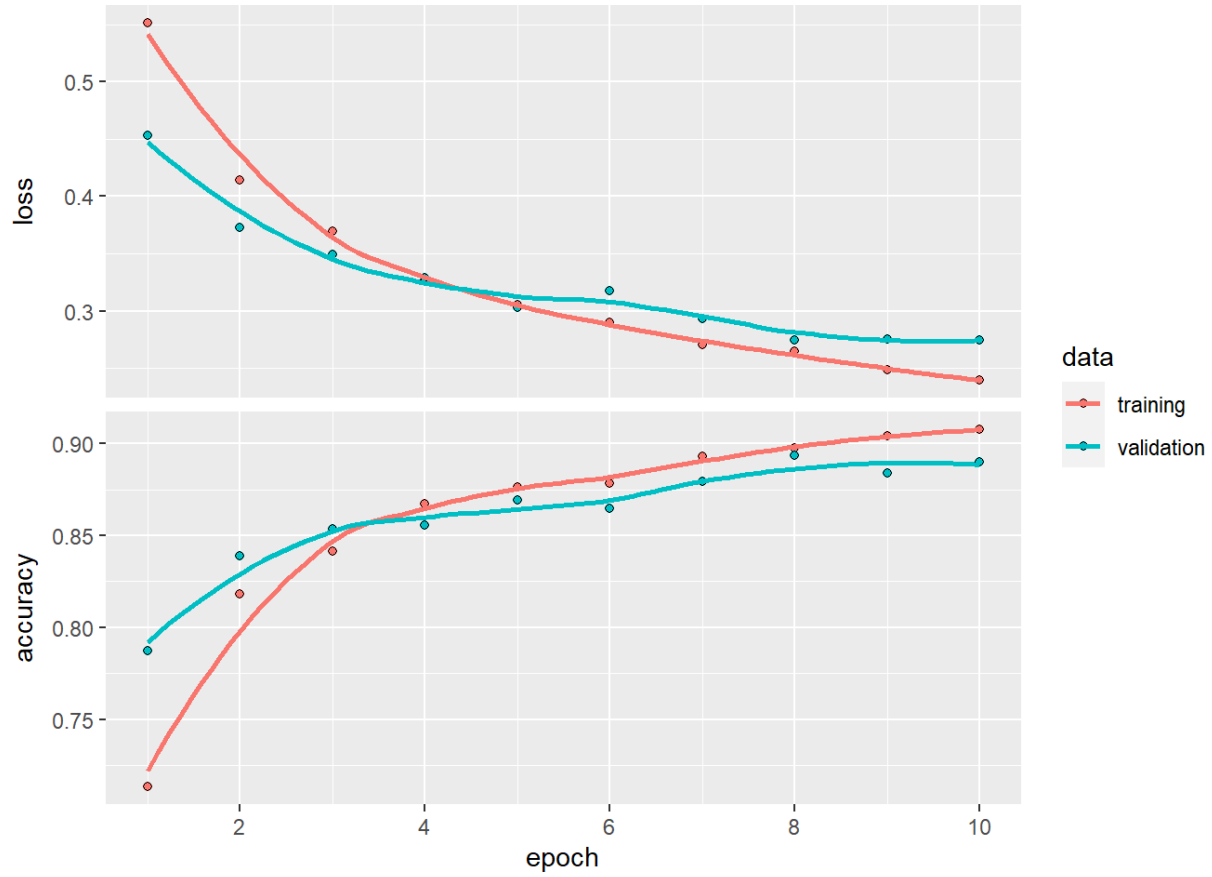
```
fim = Sys.time()
fim - inicio
```

```
## Time difference of 14.14895 secs
```

O modelo construído contém 5.0102×10^4 parâmetros para serem estimados. Os parâmetros do modelo serão definidos usando a função `fit`. Foi considerado o banco de validação e usados 10 passos (`epochs = 10`) no método interativo *backpropagation*.

```
inicio = Sys.time()
history <- model %>%
  fit(
    x = base_etapa_1$train$x,
    y = base_etapa_1$train$y,
    validation_data = unname(base_etapa_1$valid),
    epochs = 10
  )
fim = Sys.time()
fim - inicio
```

```
## Time difference of 2.095612 mins
```



A função `evaluate` usada da forma a seguir retorna o desempenho do classificador para os dados de teste.

```
inicio = Sys.time()
evaluate(model,base_etapa_1$test$x, base_etapa_1$test$y, verbose = 0)
```

```
##      loss  accuracy
## 0.2925652 0.8852006
```

```
fim = Sys.time()
fim - inicio
```

```
## Time difference of 1.472033 secs
```

O próximo código cria as matrizes de confusão em cada banco de dados: treino, validação e teste.

```
inicio = Sys.time()
predictions_treino  = predict(model, base_etapa_1$train$x)
pred_treino         = ifelse(predictions_treino[,1] > 0.5,0,1)
MC_treino = confusionMatrix(as.factor(base_etapa_1$train$y),as.factor(pred_treino))
MC_treino$table
```

```
##           Reference
## Prediction    0    1
##          0 1900  269
##          1   96 2073
```

```
predictions_valid <-  predict(model, base_etapa_1$valid$x)
pred_valid = ifelse(predictions_valid[,1] > 0.5,0,1)
MC_valid = confusionMatrix(as.factor(base_etapa_1$valid$y),as.factor(pred_valid))
MC_valid$table
```

```
##           Reference
## Prediction   0   1
##          0 608 116
##          1  65 659
```

```
predictions_teste <-  predict(model, base_etapa_1$test$x)
pred_teste = ifelse(predictions_teste[,1] > 0.5,0,1)
MC_teste = confusionMatrix(as.factor(base_etapa_1$test$y),as.factor(pred_teste))
MC_teste$table
```

```
##           Reference
## Prediction   0   1
##          0 605 118
##          1  48 675
```

```
fim = Sys.time()
fim - inicio
```

```
## Time difference of 8.444762 secs
```

As matrizes de confusão permitem o cálculo de diversas métricas de qualidade do ajuste. Os resultados estão na Tabela @tab:qualidade-etapa-1). Nela é possível ver que o classificador teve um ótimo desempenho em distinguir as imagens de Rraio x de pacientes sem doença alguma daqueles diagnosticados com COVID.

Table 1: Medidas de Qualidade do Ajuste nos Dados de Treino, Teste e Validação para Etapa 1

	Treino	Valid.	Teste
Sensitivity	0.95	0.90	0.93

	Treino	Valid.	Teste
Specificity	0.89	0.85	0.85
Pos Pred Value	0.88	0.84	0.84
Neg Pred Value	0.96	0.91	0.93
Precision	0.88	0.84	0.84
Recall	0.95	0.90	0.93
F1	0.91	0.87	0.88
Prevalence	0.46	0.46	0.45
Detection Rate	0.44	0.42	0.42
Detection Prevalence	0.50	0.50	0.50
Balanced Accuracy	0.92	0.88	0.89
Accuracy	0.92	0.88	0.89
Kappa	0.83	0.75	0.77

Segunda Etapa Nesta etapa serão considerados as imagens dos diretórios PNEUMONIA VIRAL e COVID. As imagens categorizadas como PNEUMONIA VIRAL foram rotuladas com `0`, enquanto as imagens categorizadas como COVID foram rotuladas com `1`. Todo o processo da etapa 2 é semelhante ao da etapa 1, apenas trocando as imagens categorizadas por NORMAL por PNEUMONIA VIRAL. Por esse motivo, para essa etapa será apresentado somente o código sem repetir a explicação das linhas de comando dele.

```
inicio = Sys.time()
num_de_classes = 2

n_pneumo = length(banco_pneumo[[2]])
n_covid = length(banco_covid[[2]])

banco = list()
banco[[1]] = rbind(banco_pneumo[[1]], banco_covid[[1]])
banco[[2]] = c(banco_pneumo[[2]], banco_covid[[2]])

rotulos = c(rep(0,n_pneumo),rep(1,n_covid))

#0 = PNEUMO
#1 = COVID
fim = Sys.time()
fim - inicio
```

```
## Time difference of 0.01477194 secs
```

```
inicio = Sys.time()
base_etapa_2 = Treino_Teste_Valid(Tabela_info = banco,
                                  Vetor_class = rotulos,
                                  Prop = c(0.6,0.2,0.2),
                                  COR = F,
                                  BALANCEAMENTO = T)
fim = Sys.time()
fim - inicio
```

```
## Time difference of 0.786607 secs
```

```
inicio = Sys.time()
model <- keras_model_sequential() %>%
  layer_conv_2d(
    kernel_size = c(3, 3),
    filter = 10,
```

```
    strides = 1,
    activation = "relu",
    input_shape = c(h, w, 1)
  ) %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_dropout(rate = 0.5) %>%
  layer_flatten() %>%
  layer_dense(units = num_de_classes, activation = 'softmax')
fim = Sys.time()
fim - inicio
```

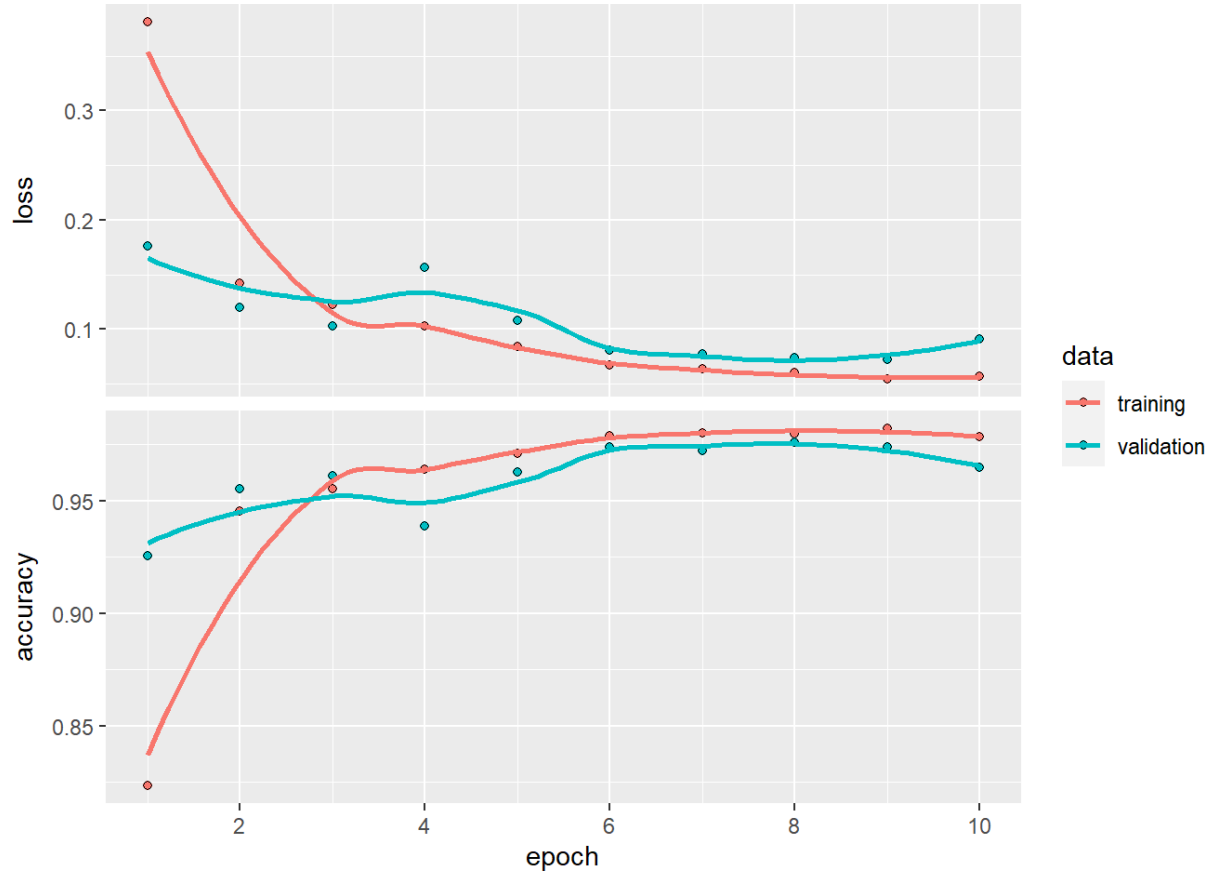
```
## Time difference of 0.07016301 secs
```

```
inicio = Sys.time()
model %>% compile(
  optimizer = "adam",
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)
fim = Sys.time()
fim - inicio
```

```
## Time difference of 0.008748055 secs
```

```
inicio = Sys.time()
history <- model %>%
  fit(
    x = base_etapa_2$train$x,
    y = base_etapa_2$train$y,
    validation_data = unname(base_etapa_2$valid),
    epochs = 10
  )
fim = Sys.time()
fim - inicio
```

```
## Time difference of 48.60367 secs
```




```
inicio = Sys.time()
evaluate(model,base_etapa_2$test$x,base_etapa_2$test$y, verbose = 0)
```

```
##       loss   accuracy
## 0.06490357 0.97211897
```

```
predictions_treino  = predict(model, base_etapa_2$train$x)
pred_treino         = ifelse(predictions_treino[,1] > 0.5,0,1)
MC_treino = confusionMatrix(as.factor(base_etapa_2$train$y),as.factor(pred_treino))
MC_treino$table
```

```
##           Reference
## Prediction   0   1
##          0 801   6
##          1  13 794
```

```
predictions_valid <-  predict(model, base_etapa_2$valid$x)
pred_valid = ifelse(predictions_valid[,1] > 0.5,0,1)
MC_valid = confusionMatrix(as.factor(base_etapa_2$valid$y),as.factor(pred_valid))
MC_valid$table
```

```
##           Reference
## Prediction   0   1
##          0 266   3
##          1  10 259
```

```
predictions_teste <-  predict(model, base_etapa_2$test$x)
```

```
pred_teste = ifelse(predictions_teste[,1] > 0.5,0,1)
MC_teste = confusionMatrix(as.factor(base_etapa_2$test$y),as.factor(pred_teste))
MC_teste$table
```

```
##           Reference
## Prediction   0   1
##          0 268   1
##          1  14 255
```

```
fim = Sys.time()
fim - inicio
```

```
## Time difference of 3.576445 secs
```

Table 2: Medidas de Qualidade do Ajuste nos Dados de Treino, Teste e Validação para Etapa 2

	Treino	Valid.	Teste
Sensitivity	0.98	0.96	0.95
Specificity	0.99	0.99	1.00
Pos Pred Value	0.99	0.99	1.00
Neg Pred Value	0.98	0.96	0.95
Precision	0.99	0.99	1.00
Recall	0.98	0.96	0.95
F1	0.99	0.98	0.97
Prevalence	0.50	0.51	0.52
Detection Rate	0.50	0.49	0.50
Detection Prevalence	0.50	0.50	0.50
Balanced Accuracy	0.99	0.98	0.97
Accuracy	0.99	0.98	0.97
Kappa	0.98	0.95	0.94

Terceira Etapa A diferença entre as etapas anteriores e essa é que nesta serão considerados três categorias, NORMAL, PNEUMONIA VIRAL e COVID. Dessa forma o classificador não será mais de duas classes e sim de três.

As imagens categorizadas como NORMAL foram rotuladas com `0`, as imagens categorizadas PNEUMONIA VIRAL foram rotuladas com `1` e as imagens categorizadas como COVID foram rotuladas com `2`.

```
num_de_classes = 3

n_normal = length(banco_normal[[2]])
n_pneumo = length(banco_pneumo[[2]])
n_covid = length(banco_covid[[2]])


banco = list()
banco[[1]] = rbind(banco_normal[[1]],
                   banco_pneumo[[1]],
                   banco_covid[[1]])

banco[[2]] = c(banco_normal[[2]],
               banco_pneumo[[2]],
               banco_covid[[2]])
```

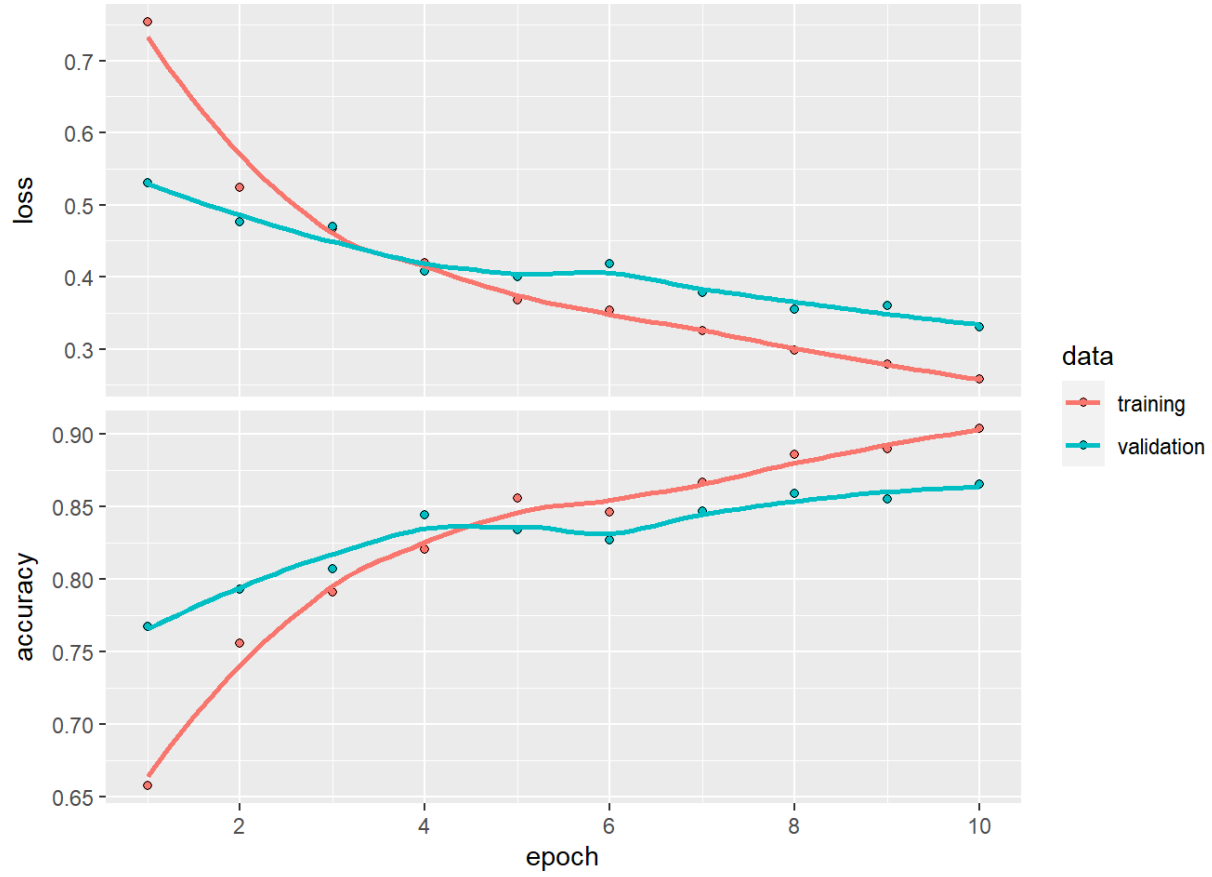
```
#0 = NORMAL
#1 = PNEUMO
#2 = COVID
rotulos = c(rep(0,n_normal),rep(1,n_pneumo),rep(2,n_covid))
```

```
base_etapa_3 = Treino_Teste_Valid(Tabela_info = banco,
                                  Vetor_class = rotulos,
                                  Prop = c(0.6,0.2,0.2),
                                  COR = F,
                                  BALANCEAMENTO = T)
```

```
model <- keras_model_sequential() %>%
  layer_conv_2d(
    kernel_size = c(3, 3),
    filter = 10,
    strides = 1,
    activation = "relu",
    input_shape = c(h, w, 1)
  ) %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_dropout(rate = 0.5) %>%
  layer_flatten() %>%
  layer_dense(units = num_de_classes, activation = 'softmax')

model %>% compile(
  optimizer = "adam",
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)

history <- model %>%
  fit(
    x = base_etapa_3$train$x,
    y = base_etapa_3$train$y,
    validation_data = unname(base_etapa_3$valid),
    epochs = 10
  )
```




```
evaluate(model,base_etapa_3$test$x, base_etapa_3$test$y, verbose = 0)
```

```
##      loss  accuracy
## 0.3334535 0.8550186
```

```
predictions_treino  = predict(model, base_etapa_3$train$x)
posicao_maior       = apply(predictions_treino,1,FUN=which.max)
pred_treino         = posicao_maior - 1
MC_treino = confusionMatrix(as.factor(base_etapa_3$train$y),
                            as.factor(pred_treino))
MC_treino$table
```

```
##           Reference
## Prediction   0   1   2
##          0 685  25  97
##          1   4 800   3
##          2  42   5 760
```

```
predictions_valid <-  predict(model, base_etapa_3$valid$x)
posicao_maior = apply(predictions_valid,1,FUN=which.max)
pred_valid         = posicao_maior - 1
MC_valid = confusionMatrix(as.factor(base_etapa_3$valid$y),
                           as.factor(pred_valid))
MC_valid$table
```

```
##           Reference
## Prediction   0   1   2
```

```
##          0 203  22  44
##          1   6 258   5
##          2  25   4 240
```

```
predictions_teste <-  predict(model, base_etapa_3$test$x)
posicao_maior = apply(predictions_teste,1,FUN=which.max)
pred_teste    = posicao_maior - 1
MC_teste = confusionMatrix(as.factor(base_etapa_3$test$y),
                           as.factor(pred_teste))
MC_teste$table
```

```
##           Reference
## Prediction   0   1   2
##          0 201  21  47
##          1   2 261   6
##          2  30  11 228
```

Table 3: Medidas de Qualidade do Ajuste nos Dados de Treino para Etapa 3

	Class: 0	Class: 1	Class: 2
Sensitivity	0.937	0.964	0.884
Specificity	0.928	0.996	0.970
Pos.Pred.Value	0.849	0.991	0.942
Neg.Pred.Value	0.971	0.981	0.938
Precision	0.849	0.991	0.942
Recall	0.937	0.964	0.884
F1	0.891	0.977	0.912
Prevalence	0.302	0.343	0.355
Detection.Rate	0.283	0.330	0.314
Detection.Prevalence	0.333	0.333	0.333
Balanced.Accuracy	0.932	0.980	0.927

Table 4: Medidas de Qualidade do Ajuste nos Dados de Validação para Etapa 3

	Class: 0	Class: 1	Class: 2
Sensitivity	0.868	0.908	0.830
Specificity	0.885	0.979	0.944
Pos.Pred.Value	0.755	0.959	0.892
Neg.Pred.Value	0.942	0.952	0.909
Precision	0.755	0.959	0.892
Recall	0.868	0.908	0.830
F1	0.807	0.933	0.860
Prevalence	0.290	0.352	0.358
Detection.Rate	0.252	0.320	0.297
Detection.Prevalence	0.333	0.333	0.333
Balanced.Accuracy	0.876	0.944	0.887

Table 5: Medidas de Qualidade do Ajuste nos Dados de Teste para Etapa 3

	Class: 0	Class: 1	Class: 2
Sensitivity	0.863	0.891	0.811
Specificity	0.882	0.984	0.922
Pos.Pred.Value	0.747	0.970	0.848
Neg.Pred.Value	0.941	0.941	0.901
Precision	0.747	0.970	0.848
Recall	0.863	0.891	0.811
F1	0.801	0.929	0.829
Prevalence	0.289	0.363	0.348
Detection.Rate	0.249	0.323	0.283
Detection.Prevalence	0.333	0.333	0.333
Balanced.Accuracy	0.872	0.938	0.867

Imagens de Números Manuscritos

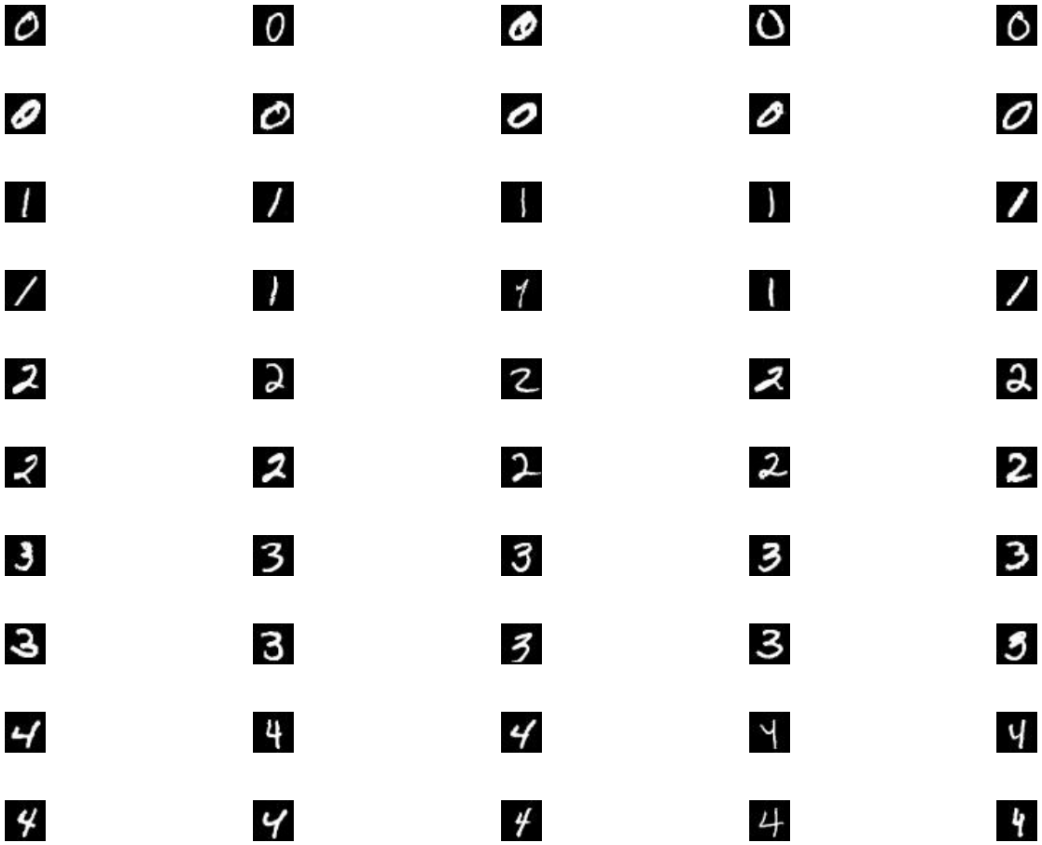
Estudo de Caso realizado por Filipe Silva Nascimento. Neste estudo foi tratado o caso de classificação de imagens em preto e branco em dez categorias.

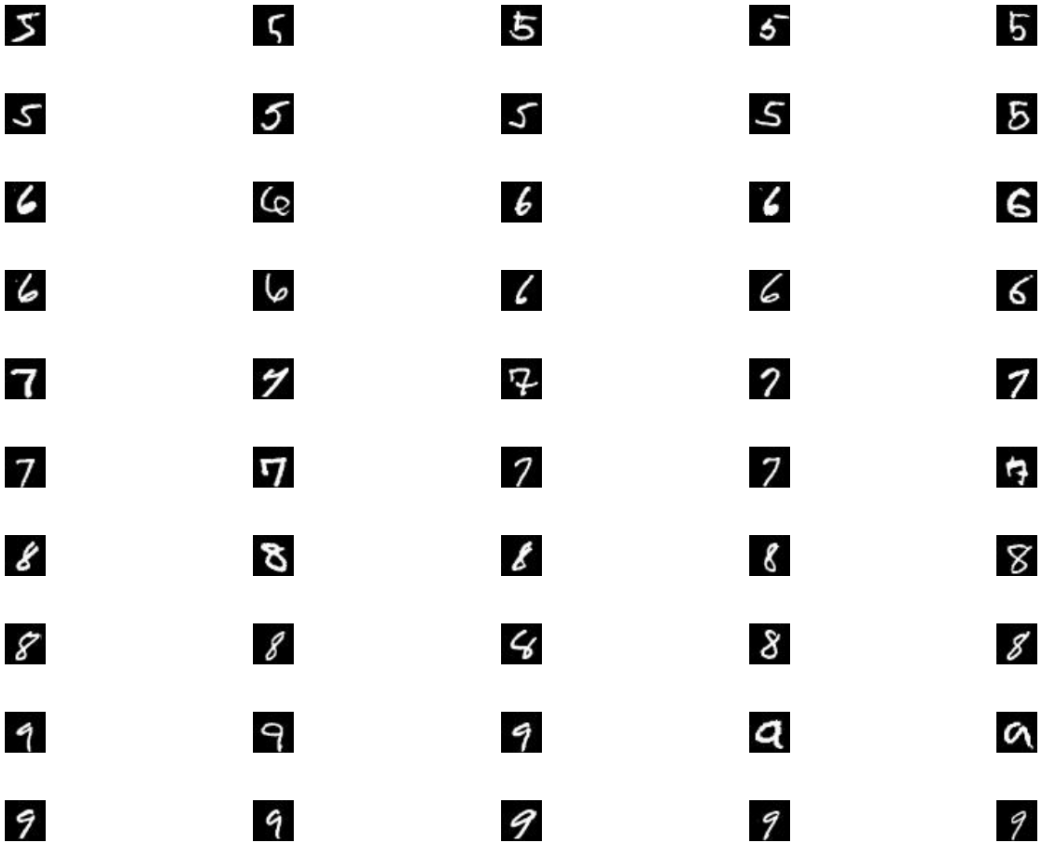
Banco de Dados

Este estudo utilizou um banco de imagens MNIST - BW&Color[3]. Esta base é composta por imagens de algarismos escritos à mão com caligrafias diferentes e está dividida em dois diretórios: "Train" e "Valid". Neste estudo serão consideradas somente as imagens contidas no diretório "Train", que consiste em 60.000 imagens na dimensão 28×28 pixel^2.

Dentro da pasta "Train" as imagens estão organizadas em 10 diretórios, nomeados de "0" até "9". Cada diretório contém imagens do número indicado pelo seu nome. Veja a seguir exemplos das imagens deste banco, selecionadas de forma aleatória dentro de cada categoria.

[3]https://www.kaggle.com/shadow1206/bwcolor (consultado em 09/08/2022)





Objetivo

O objetivo do estudo é criar um classificador capaz de identificar imagens em preto e branco rotuladas em muitas classes.

Resultados

Primeiro é feita a leitura e redimensionamento das imagens. Este processo foi realizado a partir da função `info` já definida neste relatório. Essa função foi chamada uma vez para cada um dos 10 diretórios nomeados de `"0"` até `"9"`, depois os objetos de saída foram concatenados em um único objeto nomeado por `banco_completo`.

As imagens serão mantidas na dimensão de 28 *pixels* de altura por 28 de largura. O objeto `local` apresentado no *script* é do tipo `"character"` e indica o diretório onde estão as pastas nomeadas com "0" até "9". Por tratar-se de um banco grande, o processo é demorado e por isso optamos por salvar os dados em arquivos .Rdata para que não seja necessário repetir esse processo futuramente.

```
inicio = Sys.time()
h = 28
w = 28
num_de_classes = 10

bancos_por_numeros = list()

for(i in 0:9){
  local_dir_i   = paste0(local,"/",i)
  banco_i       = info(file_path=local_dir_i,
```

```
                        image_height = h,
                        image_width= w)
  bancos_por_numeros[[i+1]] = banco_i
  save(banco_i,file=paste0("banco_",i,".Rdata"))
}

fim = Sys.time()
fim - inicio
```

```
## Time difference of 10.30643 mins
```

O código a seguir apresenta a concatenação dos 10 bancos em um único banco `banco_completo` e a criação do vetor de rótulos.

```
rotulos = NULL

i = 0
banco_i = bancos_por_numeros[[i+1]]
n = length(banco_i[[2]])
rotulos = c(rotulos,rep(i,n))
banco_completo = banco_i

for(i in 1:9){
  banco_i = bancos_por_numeros[[i+1]]
  n = length(banco_i[[2]])
  rotulos = c(rotulos,rep(i,n))
  banco_completo[[1]] = rbind(banco_completo[[1]],banco_i[[1]])
  banco_completo[[2]] = c(banco_completo[[2]],banco_i[[2]])
}
```

Em seguida serão cosntruídos os bancos de treino, validação e teste a partir da função `Treino_Teste_Valid`. Vamos construir bancos balanceados e lembrar que essas imagens são em preto e branco.

```
## Time difference of 2.855283 secs
```

Após a separação em treino, validação e teste os bancos ficaram com as seguintes tamanhos: a base de treino com 32520 imagens, sendo 3252 de cada classe; a base de validação com 10840 imagens, sendo 1084 de cada classe; e a base de teste com 10850 imagens, sendo 1085 de cada classe. Algumas imagens foram desconsideradas devido a realização do balanceamento nas três bases deste estudo.

Uma vez concluída a construção das bases de treino, teste e validação parte-se para a definição da CNN e o seu treinamento. Será usada a mesma CNN do estudo anterior. A ideia não variar a CNN e ver como esta versão simplificada se comporta nos diferentes estudos de caso.

```
model <- keras_model_sequential() %>%
  layer_conv_2d(
    kernel_size = c(3, 3),
    filter = 10,
    strides = 1,
    activation = "relu",
    padding = "same",
    input_shape = c(h, w, 1)
  ) %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_dropout(rate = 0.5) %>%
  layer_flatten() %>%
  layer_dense(units = num_de_classes, activation = 'softmax')
```

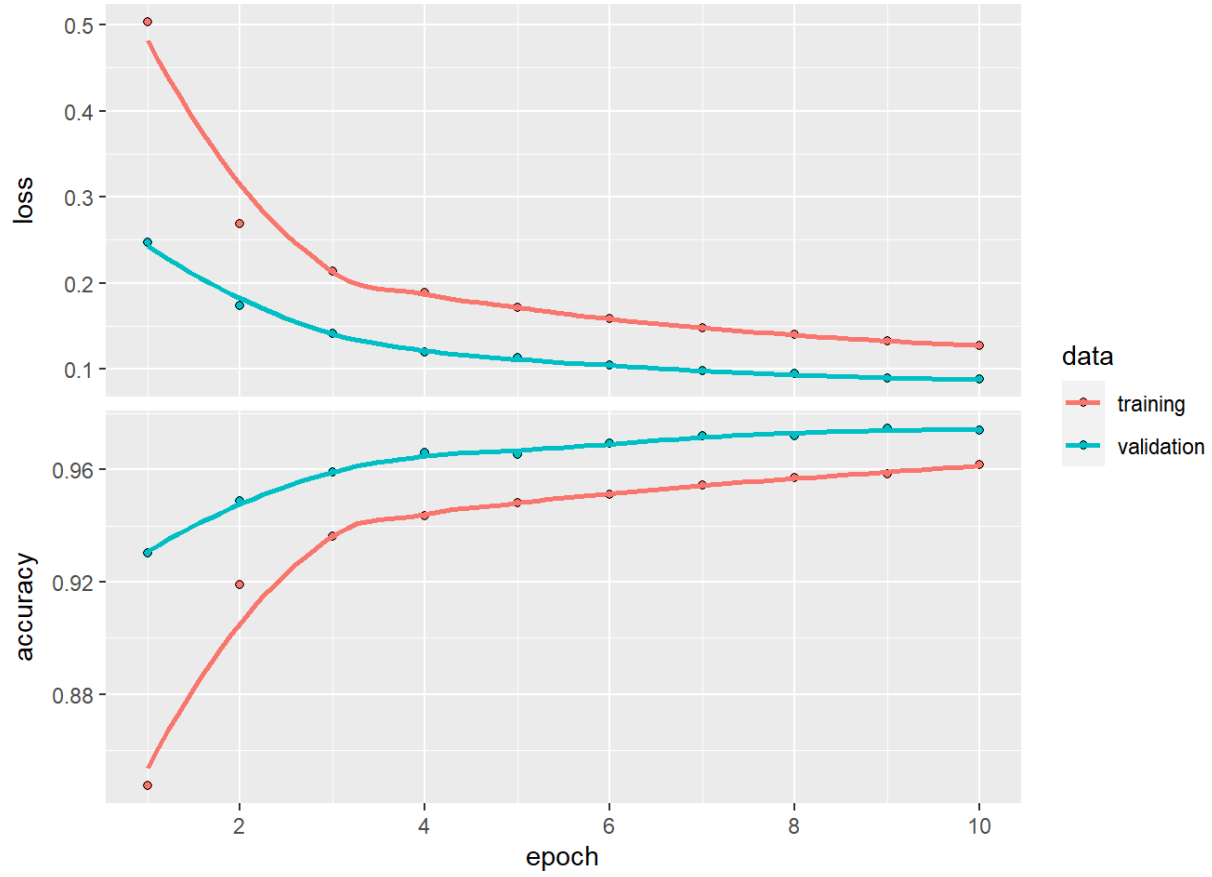
```
model %>% compile(
  optimizer = "adam",
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)
summary(model)
```

```
## Model: "sequential_3"
## _________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =================================================================================
##  conv2d_3 (Conv2D)                  (None, 28, 28, 10)              100
##  max_pooling2d_3 (MaxPooling2D)     (None, 14, 14, 10)              0
##  dropout_3 (Dropout)                (None, 14, 14, 10)              0
##  flatten_3 (Flatten)                (None, 1960)                    0
##  dense_3 (Dense)                    (None, 10)                      19610
## =================================================================================
## Total params: 19,710
## Trainable params: 19,710
## Non-trainable params: 0
## _________________________________________________________________________________
```

O modelo construído contém 19710 parâmetros para serem estimados.

```
inicio = Sys.time()
history <- model %>%
  fit(
    x = base_numeros$train$x,
    y = base_numeros$train$y,
    epochs = 10,
    validation_data = unname(base_numeros$valid)
  )
fim = Sys.time()
fim - inicio
```

```
## Time difference of 1.801159 mins
```



A função `evaluate` usada da forma a seguir retorna o desempenho do classificador para os dados de teste.

```
evaluate(model,base_numeros$test$x, base_numeros$test$y, verbose = 0)
```

```
##       loss   accuracy
## 0.08994225 0.97481549
```

A seguir são definidas as matrizes de confusão em cada banco de dados: treino, validação e teste.

```
##           Reference
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0 3233    2    4    1    0    1    3    3    5    0
##          1    0 3214   19    2    3    1    1    4    7    1
##          2    4   11 3169    9   12    1    4   17   20    5
##          3    2    1   25 3171    0   11    0   16   19    7
##          4    2    8    8    0 3199    0    8    3    4   20
##          5    6    1    3   24    2 3184   12    2   13    5
##          6   10    6    1    0    1    8 3210    0   16    0
##          7    3    8   15    4   16    1    0 3167    9   29
##          8   15   12   11    6    6   19    5    5 3160   13
##          9    8    5    2   15   27   10    0   26   20 3139
```

```
##           Reference
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0 1073    0    3    0    2    1    3    0    2    1
##          1    0 1070    6    1    2    0    1    4    1    0
##          2    2    4 1062    3    3    0    1    7    2    1
```

```
##          3    2    0    9 1052    0    7    0    5    4    6
##          4    0    1    0    0 1064    0    5    0    2   13
##          5    2    2    1   12    1 1051    6    1    9    0
##          6    6    2    0    0    2    3 1070    0    2    0
##          7    3    5   13    1    7    2    0 1034    4   16
##          8    4    8    2    4    0    3    4    1 1051    8
##          9    1    3    1    8    4    1    1   10    6 1050
```

```
##            Reference
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0 1075    0    2    0    1    2    1    0    3    0
##          1    0 1070    7    0    3    0    1    1    2    0
##          2    0    2 1053    6    4    0    0   11    8    0
##          3    1    2    8 1053    0    4    1    1   10    4
##          4    1    3    3    0 1066    0    5    1    1    4
##          5    2    3    1    6    0 1058    6    0    8    0
##          6    3    3    3    1    1    5 1063    1    4    0
##          7    1    2    9    3    8    1    0 1045    3   12
##          8    1    9    5    3    6    4    3    0 1043   10
##          9    2    4    0    4   11    2    0   12    8 1041
```

A partir das matrizes de confusão é possível calcular algumas medidas de qualidade da classificação, como mostra a Tabela **??** a seguir.

Table 6: Medidas de Qualidade do Ajuste nos Dados de Treino

	Class: 0	Class: 1	Class: 2	Class: 3	Class: 4	Class: 5	Class: 6	Class: 7	Class: 8	Class: 9
Sensitivity	0.985	0.983	0.973	0.981	0.979	0.984	0.990	0.977	0.965	0.975
Specificity	0.999	0.999	0.997	0.997	0.998	0.998	0.999	0.997	0.997	0.996
Pos.Pred.Value	0.994	0.988	0.974	0.975	0.984	0.979	0.987	0.974	0.972	0.965
Neg.Pred.Value	0.998	0.998	0.997	0.998	0.998	0.998	0.999	0.997	0.996	0.997
Precision	0.994	0.988	0.974	0.975	0.984	0.979	0.987	0.974	0.972	0.965
Recall	0.985	0.983	0.973	0.981	0.979	0.984	0.990	0.977	0.965	0.975
F1	0.989	0.986	0.974	0.978	0.982	0.982	0.988	0.975	0.969	0.970
Prevalence	0.101	0.100	0.100	0.099	0.100	0.100	0.100	0.100	0.101	0.099
Detection.Rate	0.099	0.099	0.097	0.098	0.098	0.098	0.099	0.097	0.097	0.097
Detection.Prevalence	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100
Balanced.Accuracy	0.992	0.991	0.985	0.989	0.989	0.991	0.994	0.987	0.981	0.986

Table 7: Medidas de Qualidade do Ajuste nos Dados de Validação

	Class: 0	Class: 1	Class: 2	Class: 3	Class: 4	Class: 5	Class: 6	Class: 7	Class: 8	Class: 9
Sensitivity	0.982	0.977	0.968	0.973	0.981	0.984	0.981	0.974	0.970	0.959
Specificity	0.999	0.998	0.998	0.997	0.998	0.997	0.998	0.995	0.997	0.996
Pos.Pred.Value	0.989	0.986	0.979	0.970	0.981	0.969	0.986	0.953	0.969	0.968
Neg.Pred.Value	0.998	0.997	0.996	0.997	0.998	0.998	0.998	0.997	0.997	0.995
Precision	0.989	0.986	0.979	0.970	0.981	0.969	0.986	0.953	0.969	0.968
Recall	0.982	0.977	0.968	0.973	0.981	0.984	0.981	0.974	0.970	0.959
F1	0.985	0.982	0.973	0.971	0.981	0.976	0.983	0.963	0.970	0.963
Prevalence	0.101	0.101	0.101	0.100	0.100	0.098	0.101	0.098	0.100	0.101
Detection.Rate	0.099	0.099	0.098	0.097	0.098	0.097	0.099	0.095	0.097	0.097

	Class: 0	Class: 1	Class: 2	Class: 3	Class: 4	Class: 5	Class: 6	Class: 7	Class: 8	Class: 9
Detection.Prevalence	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100
Balanced.Accuracy	0.990	0.988	0.983	0.985	0.989	0.990	0.990	0.984	0.983	0.978

Table 8: Medidas de Qualidade do Ajuste nos Dados de Teste

	Class: 0	Class: 1	Class: 2	Class: 3	Class: 4	Class: 5	Class: 6	Class: 7	Class: 8	Class: 9
Sensitivity	0.990	0.974	0.965	0.979	0.969	0.983	0.984	0.975	0.957	0.972
Specificity	0.999	0.999	0.997	0.997	0.998	0.997	0.998	0.996	0.996	0.996
Pos.Pred.Value	0.992	0.987	0.971	0.971	0.983	0.976	0.981	0.964	0.962	0.960
Neg.Pred.Value	0.999	0.997	0.996	0.998	0.997	0.998	0.998	0.997	0.995	0.997
Precision	0.992	0.987	0.971	0.971	0.983	0.976	0.981	0.964	0.962	0.960
Recall	0.990	0.974	0.965	0.979	0.969	0.983	0.984	0.975	0.957	0.972
F1	0.991	0.981	0.968	0.975	0.976	0.980	0.982	0.969	0.960	0.966
Prevalence	0.100	0.101	0.101	0.099	0.101	0.099	0.100	0.099	0.101	0.099
Detection.Rate	0.099	0.099	0.097	0.097	0.098	0.098	0.098	0.096	0.096	0.096
Detection.Prevalence	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100
Balanced.Accuracy	0.994	0.987	0.981	0.988	0.984	0.990	0.991	0.985	0.976	0.984

Imagens de Raças de Cachorros

Estudo de Caso realizado por Thiago Augusto Santos Lima. Neste estudo foi tratado o caso de classificação de imagens coloridas em mais de duas categorias.

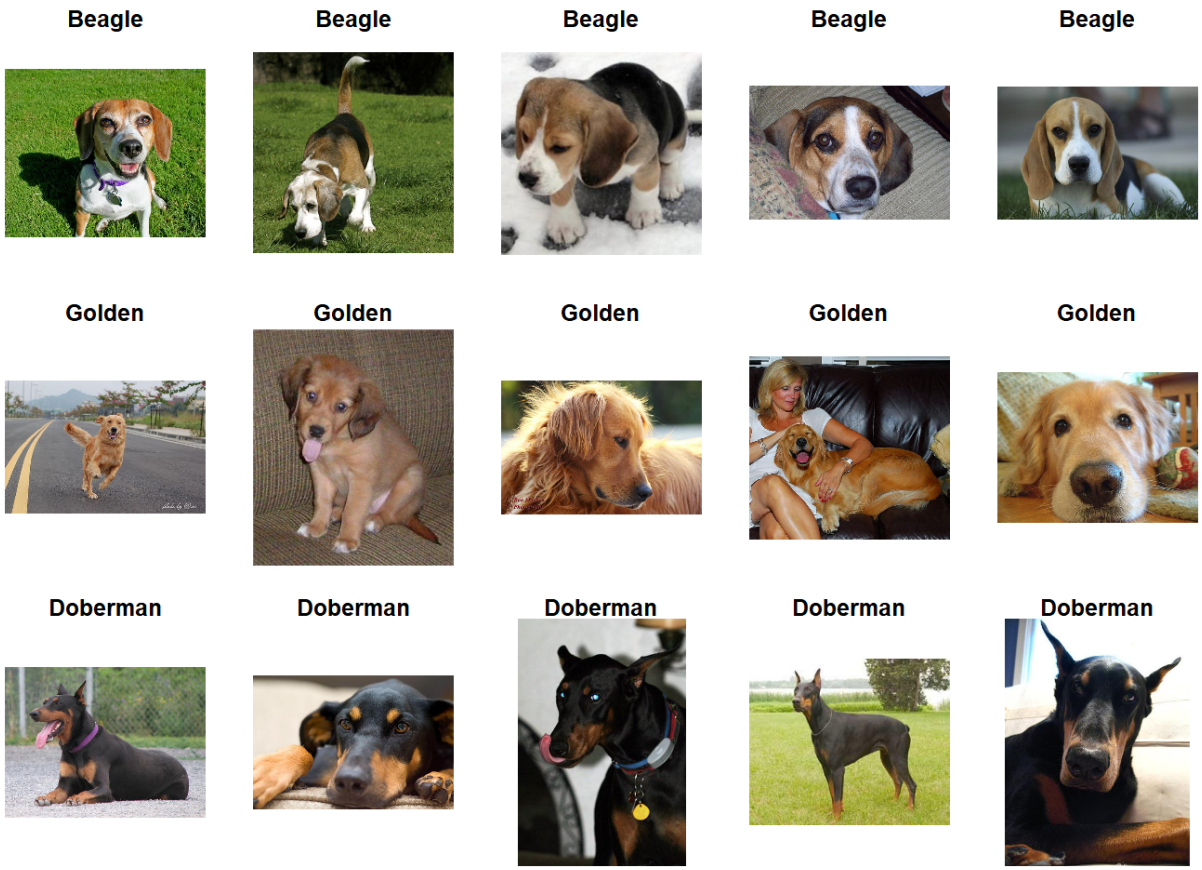
Banco de Dados

O conjunto de dados *Stanford Dogs*[4] [3] é composto por 20 mil imagens categorizadas em 120 raças de cães de todo o mundo. Cada imagem possui um rótulo que identifica a raça do cão em questão.

Para esse estudo de caso foram utilizadas as imagens de 3 raças de cães: Beagle, Golden Retriever e Doberman. O banco utilizado, depois de filtradas as três raças, é composto por 495 imagens: 195 da raça "Beagles", 150 da raça "Golden Retriever" e 150 da raça "Doberman". Os 495 arquivos de imagem estão no formato jpg, foram alocados em um diretório, aqui chamado de `Banco_Dogs`, e nomeados de forma que as raças estivessem em sequência. Os primeiros arquivos de imagem são de cães da raça "Beagles", seguidos pelos arquivos de cães da raça "Golden Retriever" e os últimos arquivos são imagens de cães da raça "Doberman".

Veja a seguir alguns exemplos de imagens deste banco, selecionadas de forma aleatória dentro de cada categoria.

[4] http://vision.stanford.edu/aditya86/ImageNetDogs/ (acessado em 02/08/2022)



Objetivo

O objetivo do estudo é criar um classificador capaz de identificar imagens coloridas rotuladas em três classes.

Resultados

O primeiro passo, como nos estudos anteriores, é a leitura e redimensionamento das imagens. Este passo foi realizado a partir da função `info` definida anteriormente. A seguir é apresentado um *script* que apresenta o código para ler as imagens deste banco e redimensioná-las para 32 *pixels* de altura por 32 de largura. O objeto `Banco_Dogs` apresentado no *script* é do tipo `"character"` que indica o diretório onde estão as imagens do estudo.

O primeiro passo foi criar um vetor de rótulos. Os rótulos usados foram: `0` para a raça "Beagles", `1` para a raça "Golden Retriever" e `2` para a raça "Doberman". O objeto `rotulos` criado ao final do código a seguir é do tipo `array` ou `matrix` e contém 495 linhas e 1 coluna.

```
dogs      = as.integer(c(rep(0,195), rep(1,150), rep(2,150)))
rotulos   = array(dogs, c(495,1))
num_de_classes = 3
```

Usando as funções `info` e `Treino_Teste_Valid` já apresentadas foi criado o objeto `base_dogs`.

```
h = 32
w = 32
tabela_info <- info(file_path = local,
                    image_height = h,
                    image_width = w)
```

```
base_dogs <- Treino_Teste_Valid(Tabela_info = tabela_info,
                                Vetor_class = rotulos,
                                Prop = c(0.6,0.2,0.2),
                                COR = T,
                                BALANCEAMENTO = T)
```

O objeto `base_dogs` é uma lista de tamanho 3, que guarda em cada uma das posições o banco de treino, teste e validação. Para esse exemplo o banco de treino tem tamanho 270, sendo 90 de cada classe; o banco de teste tem tamanho 90, sendo 30 de cada classe; e o banco de validação tem tamanho 90, sendo 30 de cada classe.

Uma vez os bancos de treino, teste e validação separados vamos a CNN a partir da adição de filtros e funções de ativação. O código a seguir realiza essa função.

```
model <- keras_model_sequential()  %>%
  layer_conv_2d(kernel_size = c(3, 3),
                filter = 10,
                strides = 1,
                activation = "relu",
                input_shape = c(h, w, 3)) %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_dropout(rate = 0.5) %>%
  layer_flatten() %>%
  layer_dense(units = num_de_classes, activation = "softmax")
```

A função de perda utilizada será `"sparse_categorical_crossentropy"` e o otimizador `"adam"`. O banco de validação será usado para ajustar o modelo.

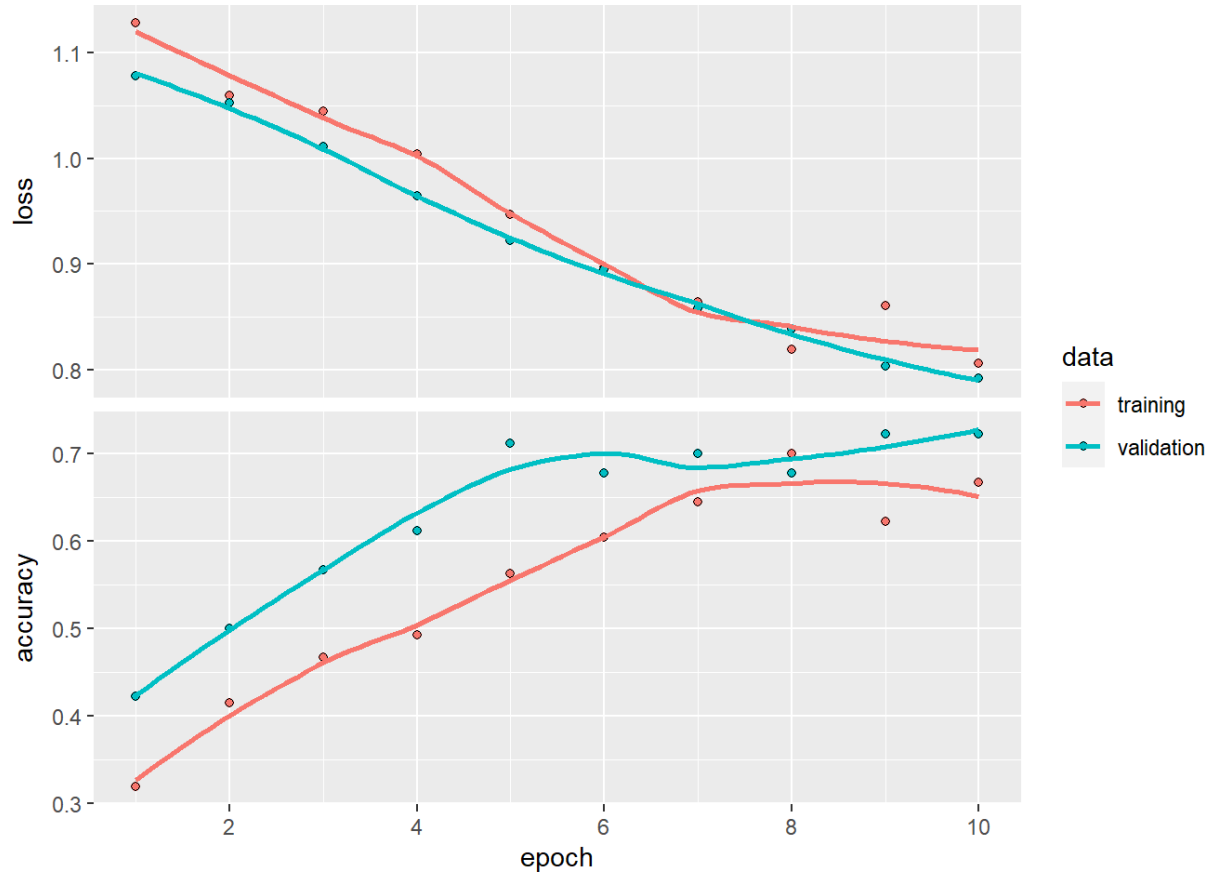
```
model %>% compile(
  optimizer = "adam",
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)
summary(model)
```

```
## Model: "sequential_4"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  conv2d_4 (Conv2D)                  (None, 30, 30, 10)              280
##  max_pooling2d_4 (MaxPooling2D)     (None, 15, 15, 10)              0
##  dropout_4 (Dropout)                (None, 15, 15, 10)              0
##  flatten_4 (Flatten)                (None, 2250)                    0
##  dense_4 (Dense)                    (None, 3)                       6753
## =========================================================================
## Total params: 7,033
## Trainable params: 7,033
## Non-trainable params: 0
## _________________________________________________________________________
```

O modelo construído contém 7033 parâmetros para serem estimados.

```
history <- model %>%
  fit(
    x = base_dogs$train$x,
    y = base_dogs$train$y,
    epochs = 10,
```

```
    validation_data = unname(base_dogs$valid),
    verbose = 2
  )
```



A acurácia na base de treino foi de 0.6814815, já na base de validação a acurácia ficou em 0.7444444. O comando a seguir calcula a acurácia para os dados de teste com o modelo ajustado, que resultou em uma acurácia de 0.5555556 para os dados de teste.

```
evaluate(model,base_dogs$test$x, base_dogs$test$y, verbose = 0)
```

```
##      loss  accuracy
## 0.8988479 0.5555556
```

```
predictions_train = predict(model, base_dogs$train$x)
posicao_maior     = apply(predictions_train,1,FUN=which.max)
pred_train        = posicao_maior - 1
MC_train = confusionMatrix(data = as.factor(pred_train),
                           reference = as.factor(base_dogs$train$y))
MC_train$table
```

```
##           Reference
## Prediction  0  1  2
##          0 69  9 11
##          1  9 72  9
##          2 12  9 70
```

```
predictions_valid =  predict(model, base_dogs$valid$x)
posicao_maior     = apply(predictions_valid,1,FUN=which.max)
pred_valid        = posicao_maior - 1
MC_valid = confusionMatrix(data = as.factor(pred_valid),
                           reference = as.factor(base_dogs$valid$y))
MC_valid$table
```

```
##           Reference
## Prediction  0  1  2
##          0 22  3  8
##          1  4 26  3
##          2  4  1 19
```

```
predictions_teste <-  predict(model, base_dogs$test$x)
posicao_maior = apply(predictions_teste,1,FUN=which.max)
pred_teste    = posicao_maior - 1
MC_teste = confusionMatrix(data = as.factor(pred_teste),
                           reference = as.factor(base_dogs$test$y))
MC_teste$table
```

```
##           Reference
## Prediction  0  1  2
##          0 14 11 10
##          1  8 19  3
##          2  8  0 17
```

Table 9: Medidas de Qualidade do Ajuste nos Dados de Treino

	Class: 0	Class: 1	Class: 2
Sensitivity	0.767	0.800	0.778
Specificity	0.889	0.900	0.883
Pos.Pred.Value	0.775	0.800	0.769
Neg.Pred.Value	0.884	0.900	0.888
Precision	0.775	0.800	0.769
Recall	0.767	0.800	0.778
F1	0.771	0.800	0.773
Prevalence	0.333	0.333	0.333
Detection.Rate	0.256	0.267	0.259
Detection.Prevalence	0.330	0.333	0.337
Balanced.Accuracy	0.828	0.850	0.831

Table 10: Medidas de Qualidade do Ajuste nos Dados de Validação

	Class: 0	Class: 1	Class: 2
Sensitivity	0.733	0.867	0.633
Specificity	0.817	0.883	0.917
Pos.Pred.Value	0.667	0.788	0.792
Neg.Pred.Value	0.860	0.930	0.833
Precision	0.667	0.788	0.792
Recall	0.733	0.867	0.633
F1	0.698	0.825	0.704
Prevalence	0.333	0.333	0.333

	Class: 0	Class: 1	Class: 2
Detection.Rate	0.244	0.289	0.211
Detection.Prevalence	0.367	0.367	0.267
Balanced.Accuracy	0.775	0.875	0.775

Table 11: Medidas de Qualidade do Ajuste nos Dados de Teste

	Class: 0	Class: 1	Class: 2
Sensitivity	0.467	0.633	0.567
Specificity	0.650	0.817	0.867
Pos.Pred.Value	0.400	0.633	0.680
Neg.Pred.Value	0.709	0.817	0.800
Precision	0.400	0.633	0.680
Recall	0.467	0.633	0.567
F1	0.431	0.633	0.618
Prevalence	0.333	0.333	0.333
Detection.Rate	0.156	0.211	0.189
Detection.Prevalence	0.389	0.333	0.278
Balanced.Accuracy	0.558	0.725	0.717

Conclusão

Neste trabalho foi estudado com usar o R para realizar classificação de imagens a partir de uma Rede Neural Convolucional. Para esse fim foi preciso antes adquirir alguns conhecimentos, como por exemplo, como as imagens são processadas pelo computador, como redimensionar imagens para que todas tenham a mesma quantidade de pixel, e como criar `arrays` no R que representem tanto as imagens em preto e branco quanto coloridas.

Também foi estudado a metodologia de redes neurais e em particular as redes neurais convolucionais (CNN). Foi definida uma única CNN e esta utilizada em três estudos de casos distintos.

O primeiro estudo de caso utilizou a CNN para criar classificadores para imagens de Raio x. Em uma primeira etapa criou-se um classificador para classificar imagens de Raio X de pacientes com COVID em relação às imagens de outros pacientes sem problema algum. Em uma segunda etapa criou-se um classificador para classificar imagens de Raio X de pacientes com COVID em relação às imagens de outros pacientes com pneumunia comum. E por fim criou-se um classificador para classificar as imagens de Raio X em uma entre as três classes: NORMAL, pneumonia e COVID.

Nas três etapas deste estudo os classificadores criados obtiveram resultados bons tanto no banco de treino, quanto no banco de validação ou teste. Para os classifciadores de duas classes,etapas 1 e 2, a precisão nos dados de teste ficou acima de 85% na base de teste. Para o classificador de três classes (etapa 3) a precisão na base de teste ficou acima de 90% para as classes 1 e 2 e ficou próximo de 65% para a classe 0.

O segundo estudo de caso trabalhou com um banco de imagens em preto e branco e 10 classes possíveis. O objetivo era identificar números de 0 até 9 manuiscritos. Este estudo, com a mesma rede CNN de todos os estudos, apresentou resultados muito bons, tais que a precisão em todas as classes não foi menor que 95%.

O último estudo de caso trabalhou com imagens de cachorros e o objetivo era classificar a raça de cães. Este estudo não apresentou resultados tão bons quanto os anteriores. Mas vale destacar que neste estudo de caso foram usadas imagens coloridas, o que triplica o número de parâmetros, e o banco de dados foi o menor deles. Analisando a matriz de confusão deste estudo pode-se perceber que o erro mais comum foi na previsão da classe `0` (“Beagles”). Ou seja, o classificador recebe a imagem de um “Beagles” e entende que é um “Doberman” ou um “Golden Retriever”.

Para finalizar a conclusão segue alguns trabalhos futuros que podem seguir deste projeto de pesquisa. Um assunto muito interessantes que infelizmente não houve tempo para abordar é a utilização de CNN para localização de objetos. Neste caso o que a CNN retorna não é uma classe, e nem a probabilidade de pertencer a ela, e sim as coordenadas dos pontos que definem as coordenadas de um retângulo que contenha o objeto da imagem. Uma aplicação deste trabalho futuro é solucionar problemas como o de identificação facial ou de digital.

References

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Suesstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34:2274–2282, 2012.

[2] M. Haghighat, S. Zonouz, and M. Abdel-Mottaleb. Cloudid: Trustworthy cloud-based and cross-enterprise biometric identification. *Expert Systems with Applications*, 42:7905–7916, 2015.

[3] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011.

[4] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[5] Lampros Mouselimis. *OpenImageR: An Image Processing Toolkit*, 2022. R package version 1.2.2.

[6] Gregoire Pau, Florian Fuchs, Oleg Sklyar, Michael Boutros, and Wolfgang Huber. Ebimage—an r package for image processing with applications to cellular phenotypes. *Bioinformatics*, 26(7):979–981, 2010.

[7] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2021.

[8] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[9] Hadley Wickham. *stringr: Simple, Consistent Wrappers for Common String Operations*, 2019. R package version 1.4.0.