REDE NEURAL CONVOLUCIONAL (CNN)

GEIA - Grupo de estudos de inteligência artificial

Aluno: Fábio Lofredo Cesar
Professor: Hygor Santiago Lara

RESUMO

Este tutorial visa ensinar sobre o funcionamento de uma Rede neural convolucional (CNN). Serão abordados os conceitos básicos, treinamento e predição. Será usada uma planilha de predição de uma CNN para mostrar de forma prática o funcionamento da mesma. Também será feito um exemplo de treinamento e predição, de CNN para classificação de dígitos numéricos, usando o banco de dados MNIST em Python. O tutorial é descrito de forma simples e intuitiva, com imagens que ajudam no entendimento.

SUMÁRIO

1.CONCEITOS BÁSICOS

1.1. INTRODUÇÃO

Rede Neural Convolucional (CNN - *Convolutional Neural Network*) é uma rede neural artificial bastante usada em imagens. Os algoritmos de CNNs usam operações matemáticas com matrizes nas suas camadas junto com redes neurais. Esse processamento permite o reconhecimento de imagens. No exemplo da figura 1, temos a imagem do “2”, ela é processada em suas várias camadas, obtendo assim uma rede neural final, que sua saída(OUTPUT) indicará qual número está na imagem.

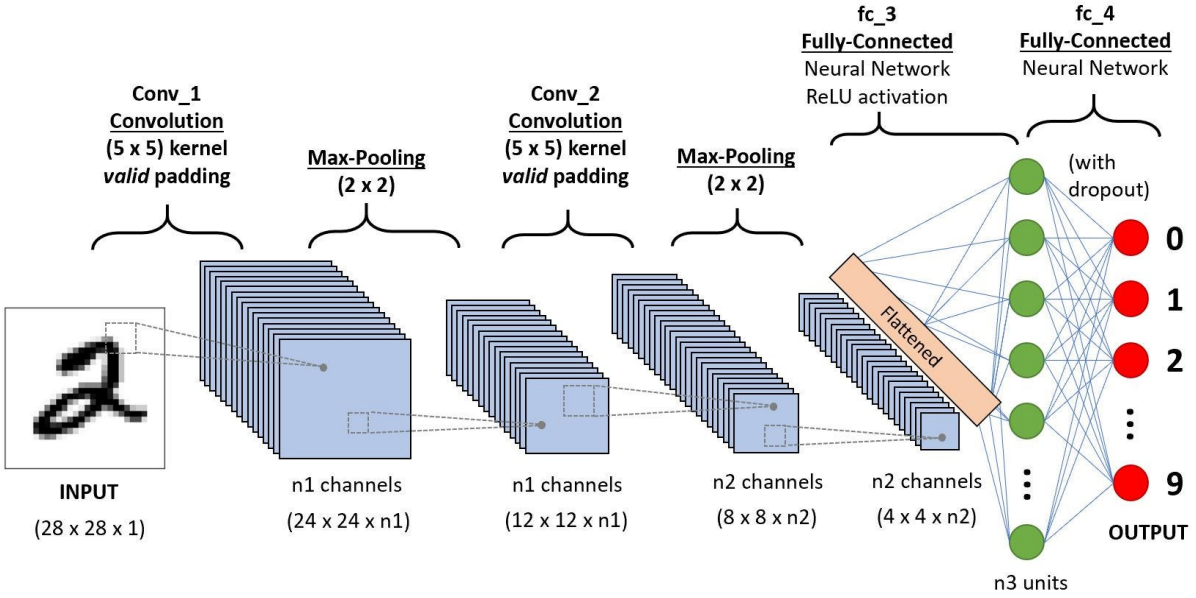


Figura 1. Rede Neural Convolucional (CNN - Convolutional Neural Network)

Acesse o simulador para visualizar melhor:
https://adamharley.com/nn_vis/cnn/2d.html

Uma rede neural precisa ser treinada, para calibrar os pesos, antes de fazer uma predição.

O algoritmo de CNN poderá utilizar **classificação** ou **regressão**. A classificação serve para obter um resultado de forma discreta, como no exemplo da figura 1, as possibilidades são de 0 a 9 usando somente números inteiros, ou seja, 0,1,2,3,4,5,6,7,8 ou 9. Já a regressão serve para obter um resultado com valor contínuo, por exemplo para estimar uma altura, o resultado poderia ser 1,7223m ou 1,9329m.

Será seguida uma estrutura simples de CNN, como na figura 2, durante esse tutorial para uma melhor visualização do processo.

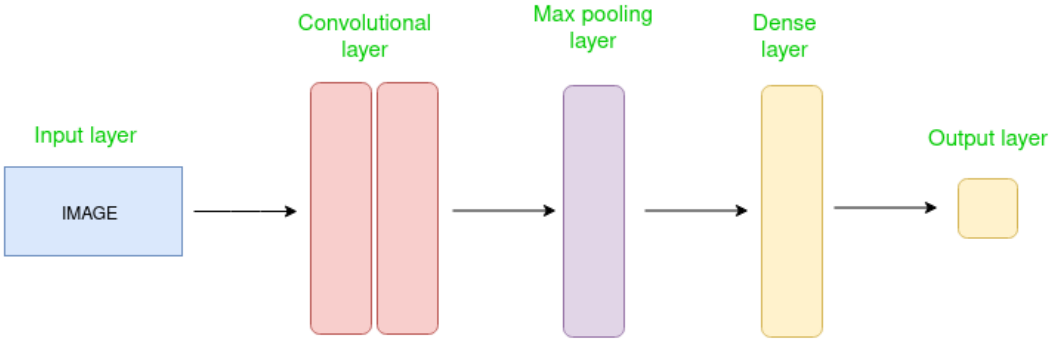


Figura 2. Estrutura simples de CNN

Também será possível acompanhar o processo pela planilha que acompanha este tutorial com um exemplo de predição de CNN.

1.2. ENTRADA DA IMAGEM

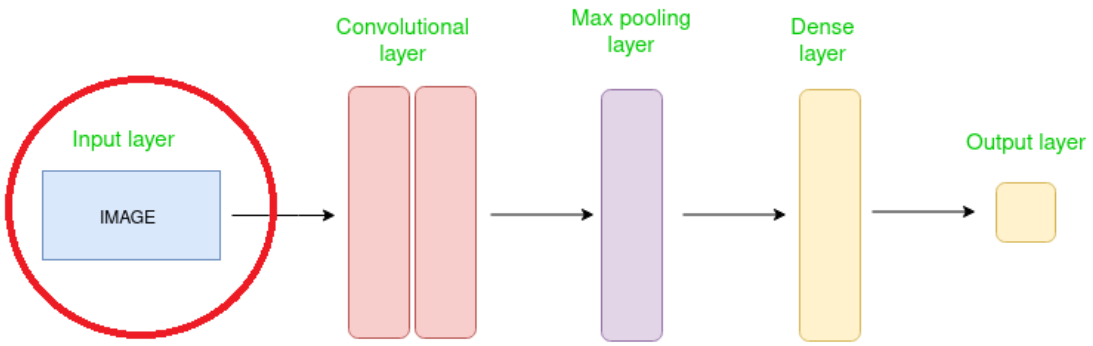


Figura 1. Estrutura simples de CNN - Input layer

Na figura 2, temos um exemplo de uma imagem RGB, com 3 canais(azul, verde e vermelho), o valor dentro de cada célula equivale a intensidade daquela cor naquele pixel. Mas existem outras representações, como por exemplo a escala de cinza, que envolve apenas um canal.

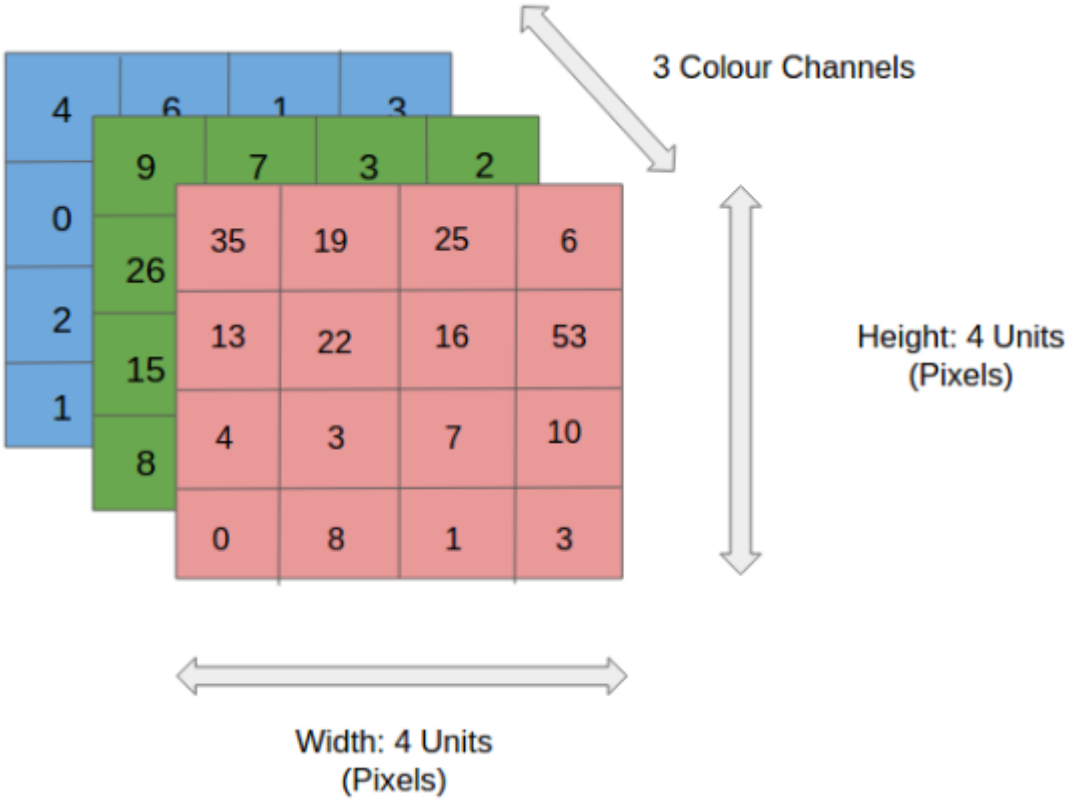


Figura 2. Imagem RGB, com 3 canais(azul, verde e vermelho), resultando uma matriz 4x4x3.

As imagens são o "*input*", ou seja, os dados de entrada para o CNN.

1.3. CAMADA DE CONVOLUÇÃO - KERNEL

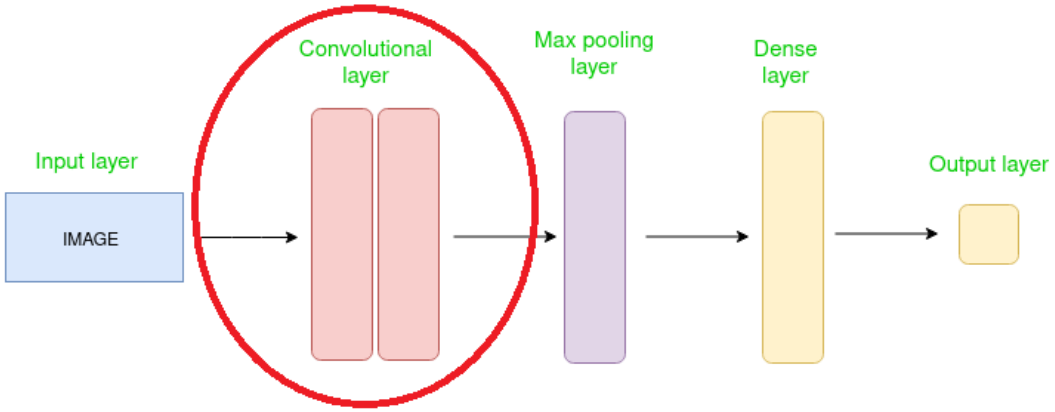


Figura 1. Estrutura simples de CNN - Convolutional layer

Kernel é uma matrix, que percorrerá a imagem fazendo cálculos matemáticos, produzindo assim uma nova imagem. Dois exemplos de *kernel* para escala de cinza seriam matrizes 3x3 ou 5x5. Já para o RGB, como possui 3 dimensões, ficariam 3x3x3 ou 5x5x3.

Faremos um exemplo de um kernel em funcionamento, segue a imagem de um kernel 3x3 na figura 2.

1	0	1
0	1	0
1	0	1

Figura 2. *Kernel* 3x3

O *kernel* irá se sobrepor no canto da imagem, multiplicando cada pixel e somando o resultado total, como podemos ver na figura

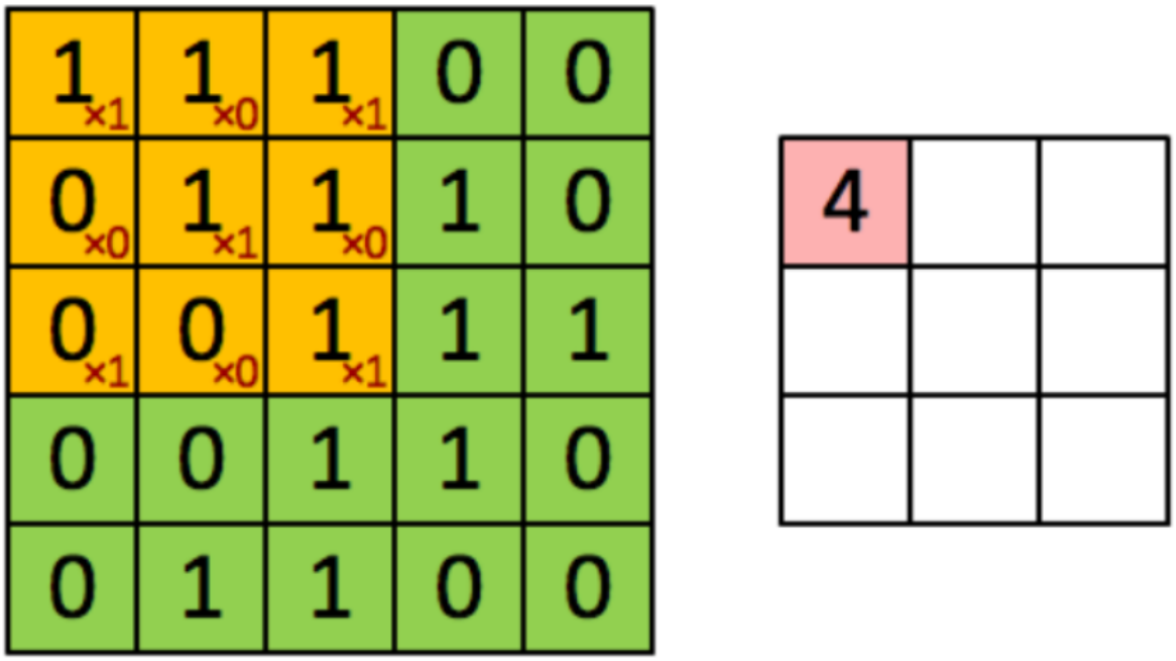


Figura 3. Usando o kernel na imagem

	1° Coluna	2° Coluna	3° Coluna
1° Linha	1x1 =1	1x0 = 0	1x1 = 1
2° Linha	0x0 = 0	1x1 = 1	1x0 = 0
3° Linha	0x1 = 0	0x0 = 0	1x1 = 1

Tabela 1. Cálculo do primeiro passo usando o *kernel* na imagem. A soma resultará em 4.

O *kernel* irá se sobrepor no canto da imagem, multiplicando cada pixel e somando o resultado total, como podemos ver na figura 4 e na tabela 1.

Para o segundo passo, o *kernel* se moverá para produzir a próxima célula da imagem.

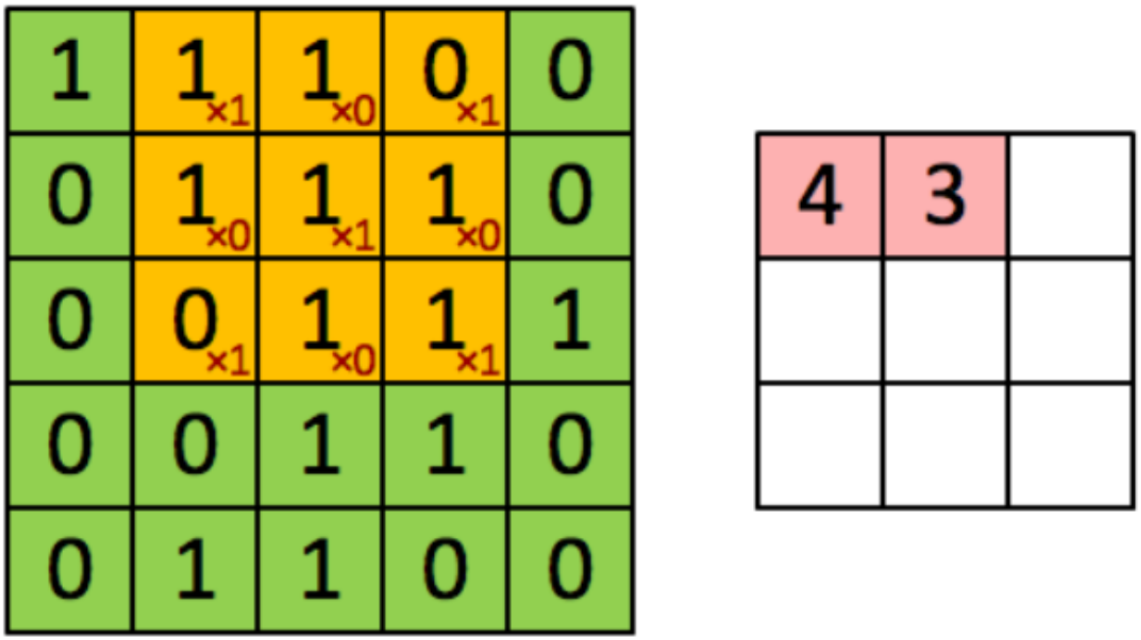


Figura 4. Usando o *kernel* na imagem, pela segunda vez

	1° Coluna	2° Coluna	3° Coluna
1° Linha	1x1 =1	1x0 = 0	0x1 = 0
2° Linha	1x0 = 0	1x1 = 1	1x0 = 0
3° Linha	0x1 = 0	1x0 = 0	1x1 = 1

Tabela 2. Cálculo do segundo passo usando o *kernel* na imagem. A soma resultará em 3.

O *kernel* percorrerá toda a imagem, produzindo assim a imagem sucedente.

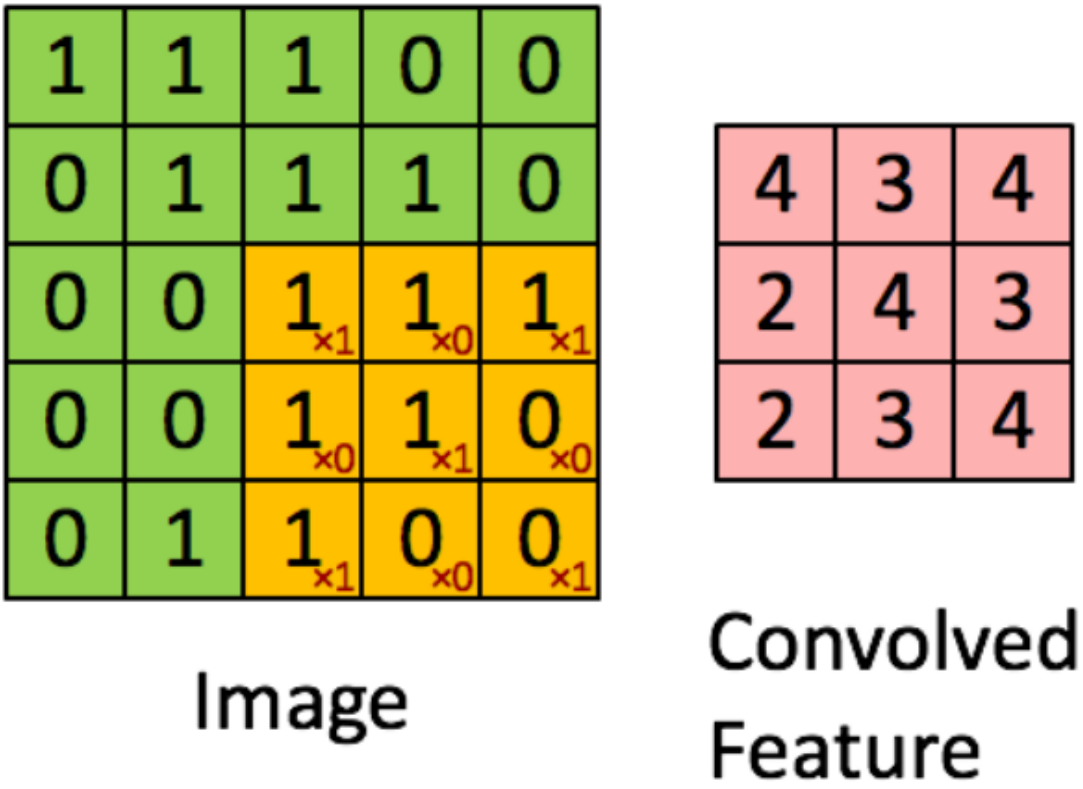


Figura 5. Usando o *kernel* na imagem, pela última vez.

Existem vários tipos de *kernel*, alguns podem detectar bordas, outros borrar a imagem. Para entender mais veja em: https://docs.gimp.org/2.8/pt_BR/plug-in-convmatrix.html

Stride é o espaçamento que o *kernel* utilizará no seu movimento. Abaixo temos um exemplo de *stride* 1 e stride 2. Nesse exemplo, stride 1 produzirá uma imagem 5x5 e com stride 2 uma imagem 3x3.

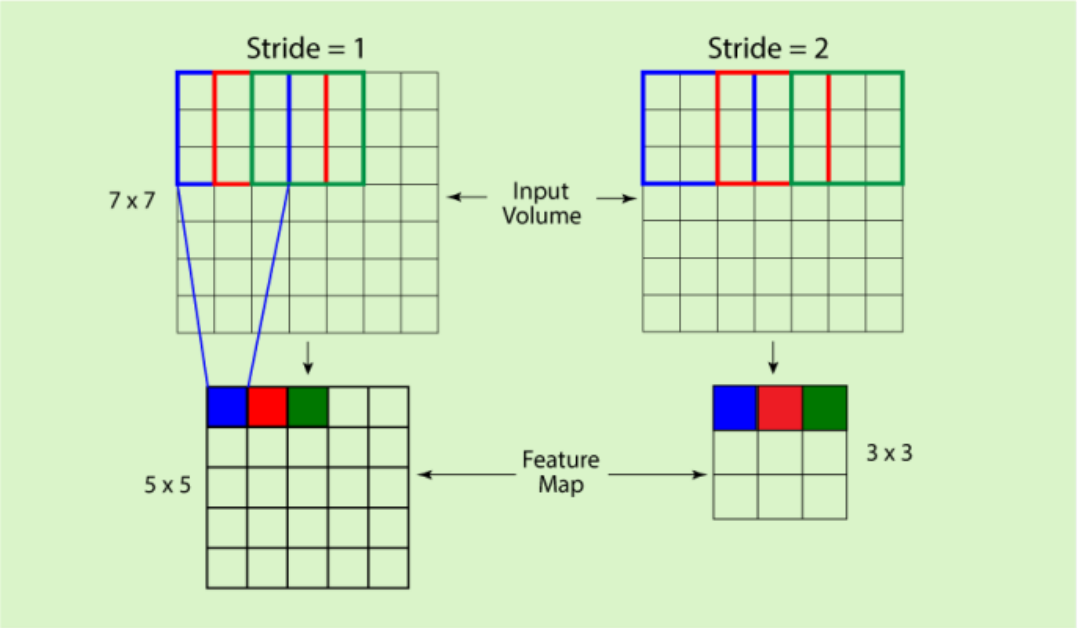


Figura 6. Stride

Padding é usado para ampliar a imagem em seus extremos e usar o *kernel* fora dos limites da imagem inicial.

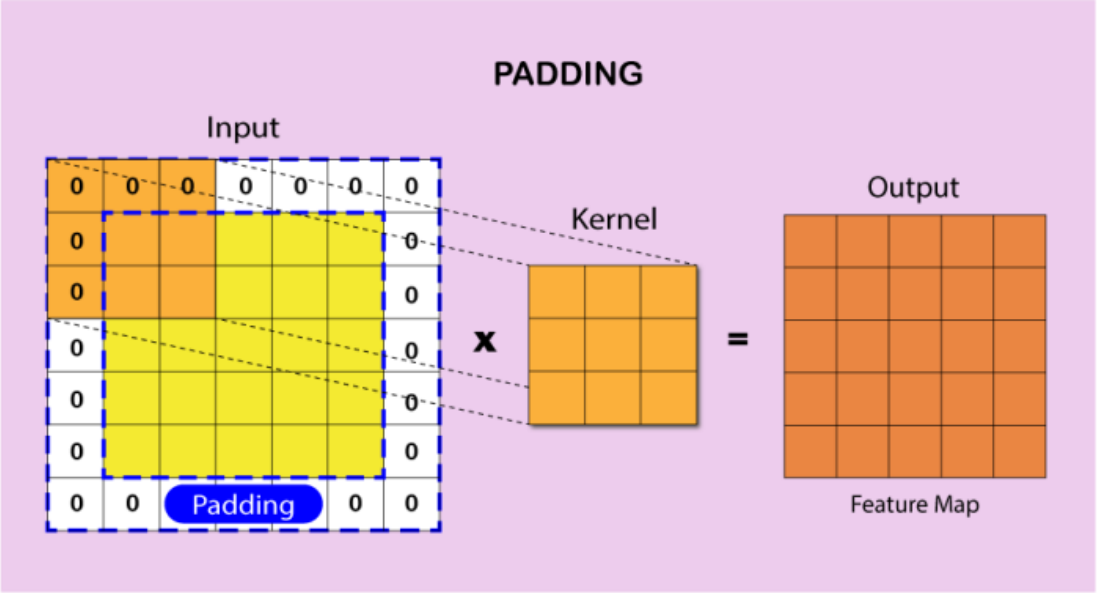


Figura 7. *Padding*

Camadas é o nome de cada processo de convolução, função de ativação e *pooling* (*pooling* é uma técnica para redução do tamanho da imagem, será explicada

em mais detalhes no próximo tópico), uma CNN pode possuir mais de uma camada. A imagem abaixo mostra um processo com 3 camadas:

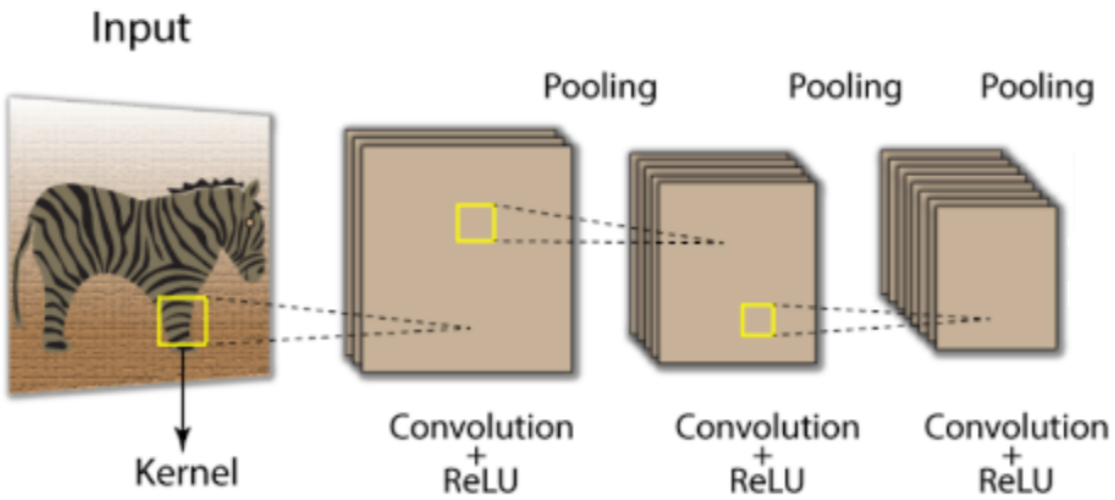


Figura 8. Camadas

Filtro, ou seja *filter*, é uma outra forma de chamar os kernels.

Feature map é a matriz gerada pelo *filter*. Portanto, a quantidade de Feature map gerada será a mesma quantidade de *filter*.

Canal é cada componente das cores que formam as imagens. RGB possui 3 canais, enquanto a escala de cinza possui apenas 1 canal.

1.4. CAMADA DE POOLING

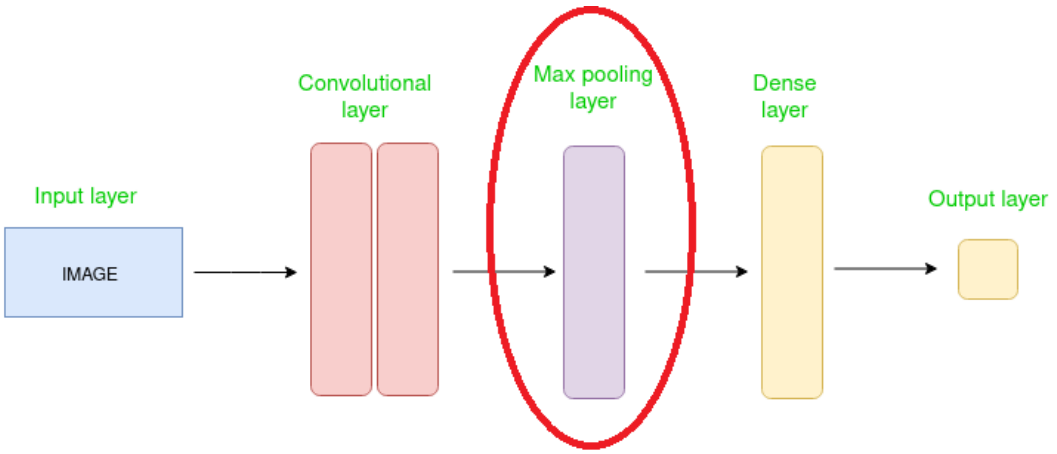


Figura 1. Estrutura simples de CNN - *Max pooling layer*

Pooling opera em cada *feature map*, ele reduz o tamanho da imagem. *Pooling* pode ser feito de duas formas:

1) ***Max-pooling***: Retorna o valor máximo na área sobreposta do kernel. Funciona como supressor de ruído.
2) ***Average-pooling***: Retorna a média na área sobreposta do kernel

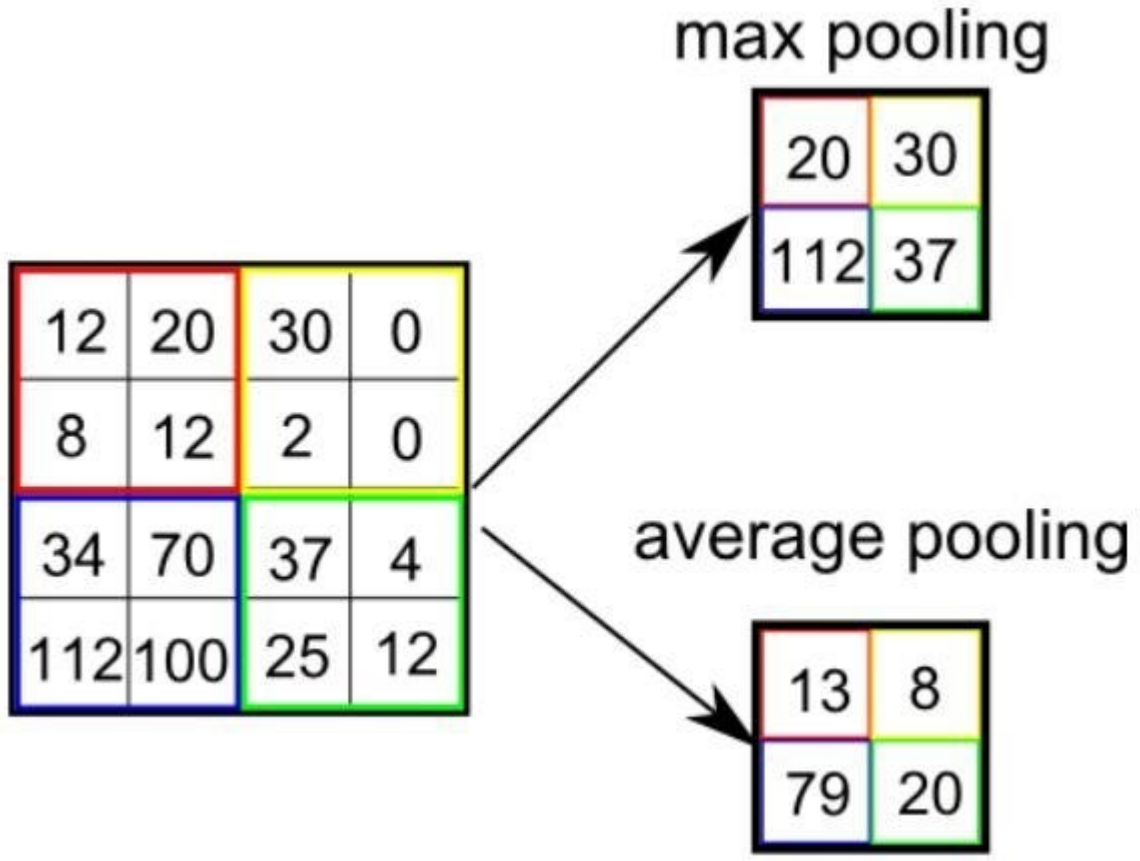


Figura 2. *Max pooling* e *Average pooling*

1.5. FLATTEN LAYER

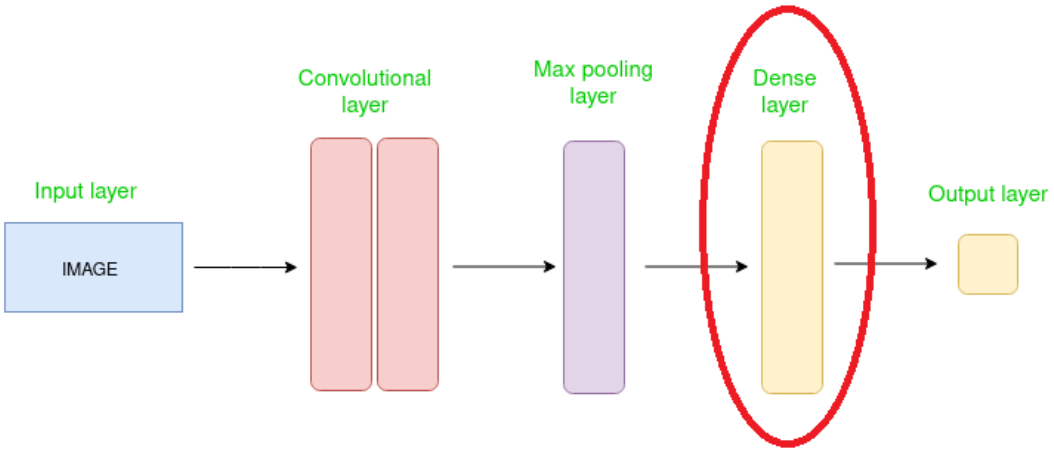


Figura 1. Estrutura simples de CNN -Dense layer

Flatten layer é um vetor linear obtido no processo de *Flattening*, consiste em transformar a imagem no formato de matrix, para um vetor linear, rearranjando os valores linearmente. Abaixo um exemplo de *flattening* para um *feature map*, uma matriz 3x3 sendo *flattening* para um *flatten layer* de 9x1.

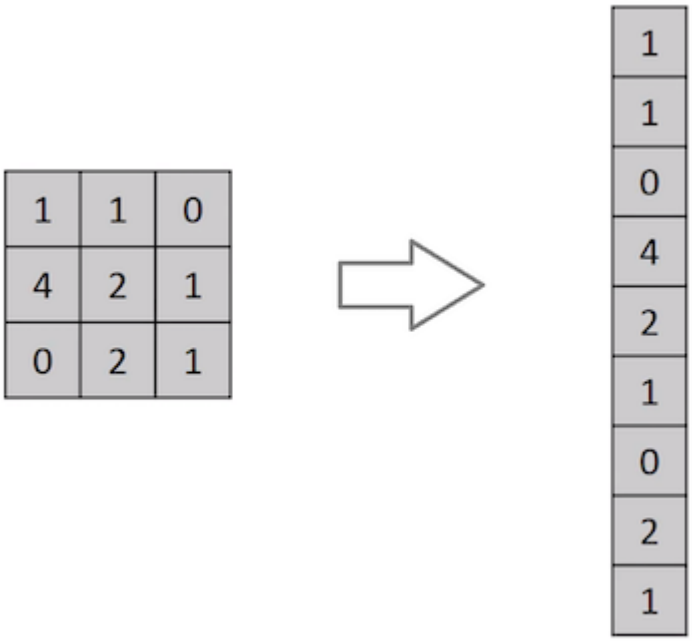


Figura 2. Transformando uma matrix 3x3 em um Flatten layer

Todas as *feature maps* passam pelo processo de *flattening*. Caso o exemplo acima possuísse 10 *feature maps*, ou seja, 10 matrizes 3x3, seria produzido um *flatten layer* de 90x1.

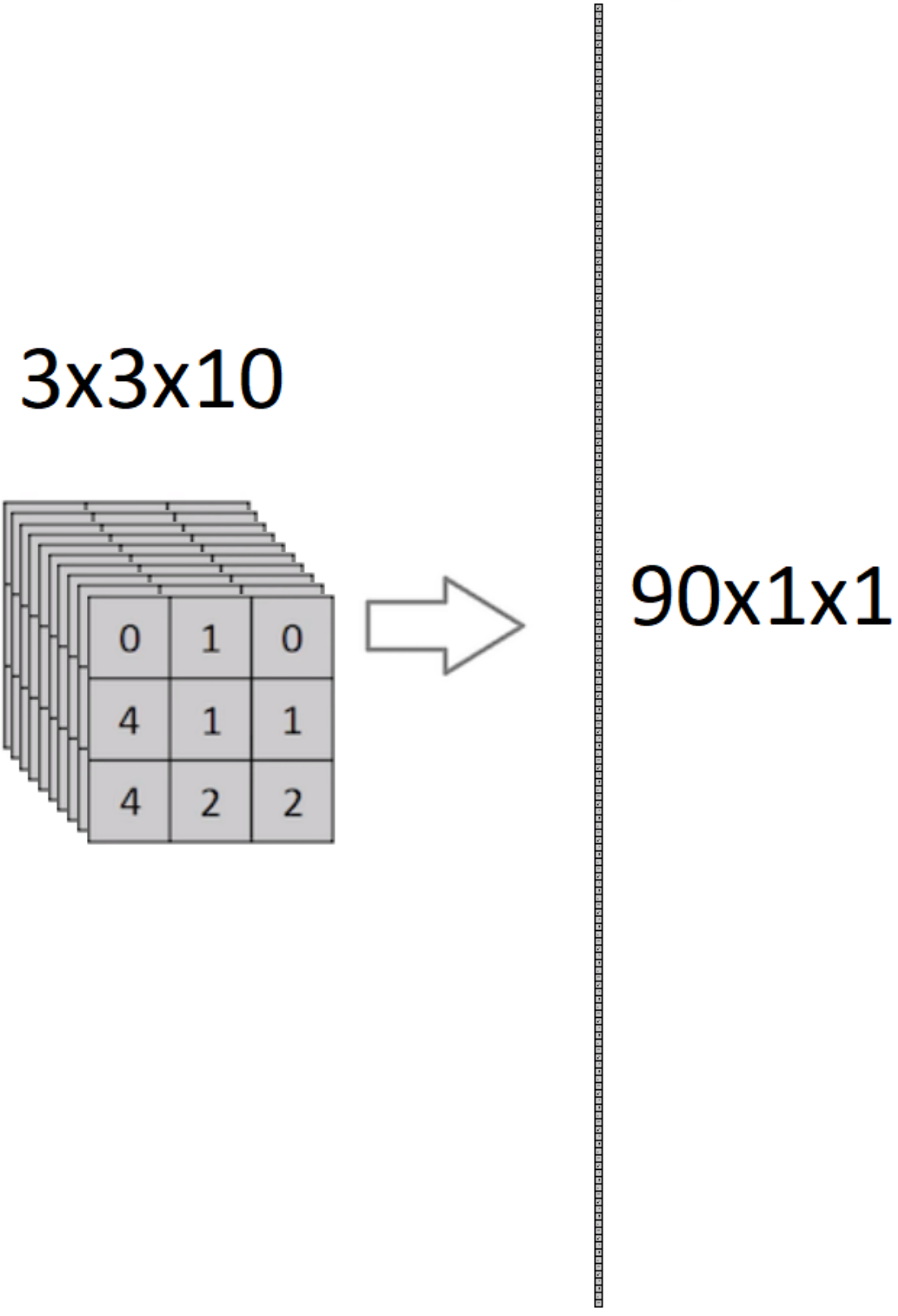


Figura 3. Transformando 10 matrizes 3x3 em um *Flatten layer*

1.6. HIDDEN E OUTPUT LAYERS

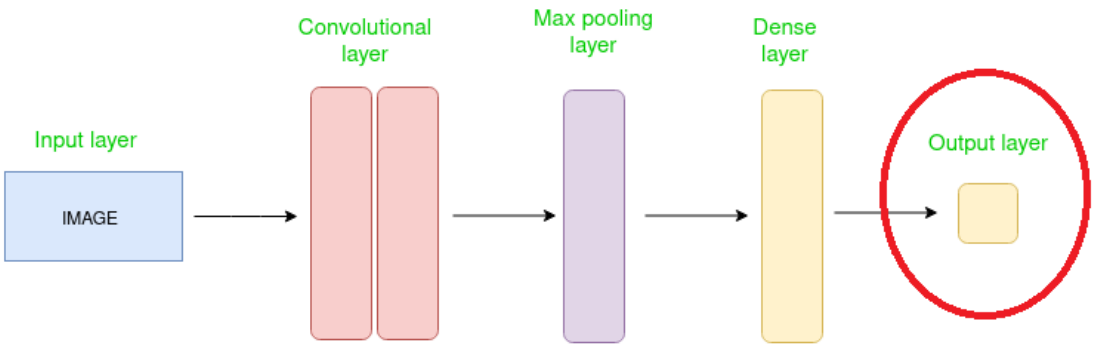


Figura 1. Estrutura simples de CNN - *Output layer*

Um algoritmo de CNN utiliza um modelo de rede neural densa no seu final, rede neural densa é uma rede neural em que todos os neurônios estão conectados entre eles mesmos. Essa rede neural densa é composta por pelo menos 3 camadas: *Input layer*, *Hidden layers* e *Output layer*. Sendo que podem existir mais de uma *Hidden layer*.

Input Layer: é o Flatten layer, a entrada dos dados.

Hidden Layers: onde será calculado(com os inputs do neurônio anterior e os pesos e bias definidos no treinamento) um valor de output para o próximo neurônio.

Output Layer: cada neurônio representa uma classificação definida, onde o neurônio com maior valor definirá o resultado.

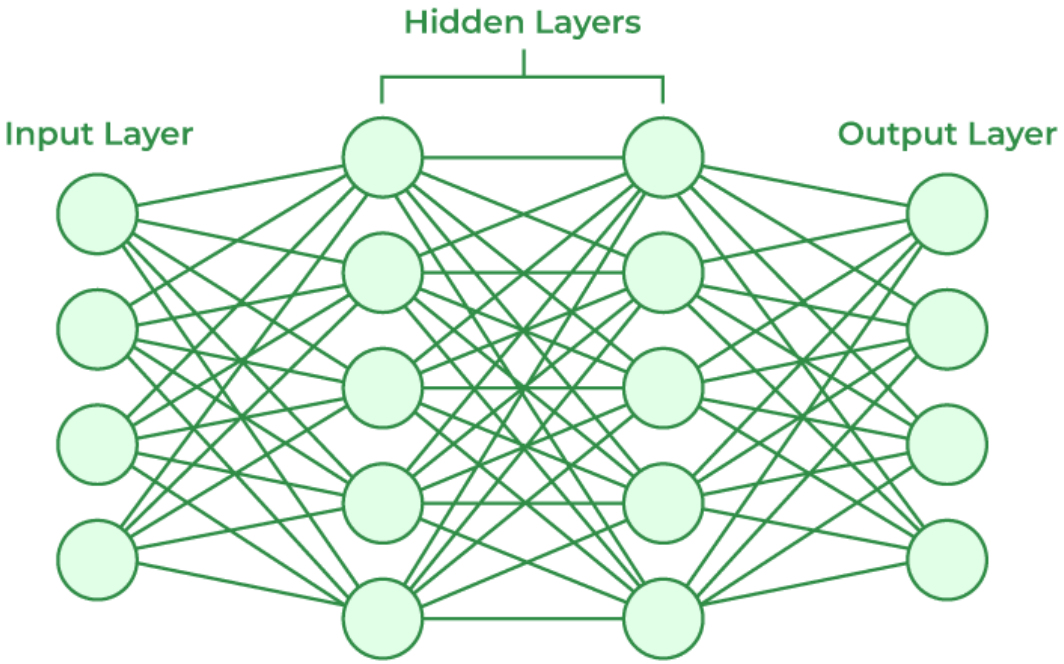


Figura 2. Rede neural densa

O ***flatten layer*** substituirá o ***Input layer***. No ***Hidden layer***, será calculado, com os inputs do neurônio anterior e os pesos e bias definidos no treinamento, um valor de output

para o próximo neurônio.No ***Output layer*** cada neurônio representa uma classificação definida, onde o neurônio com maior valor definirá o resultado.

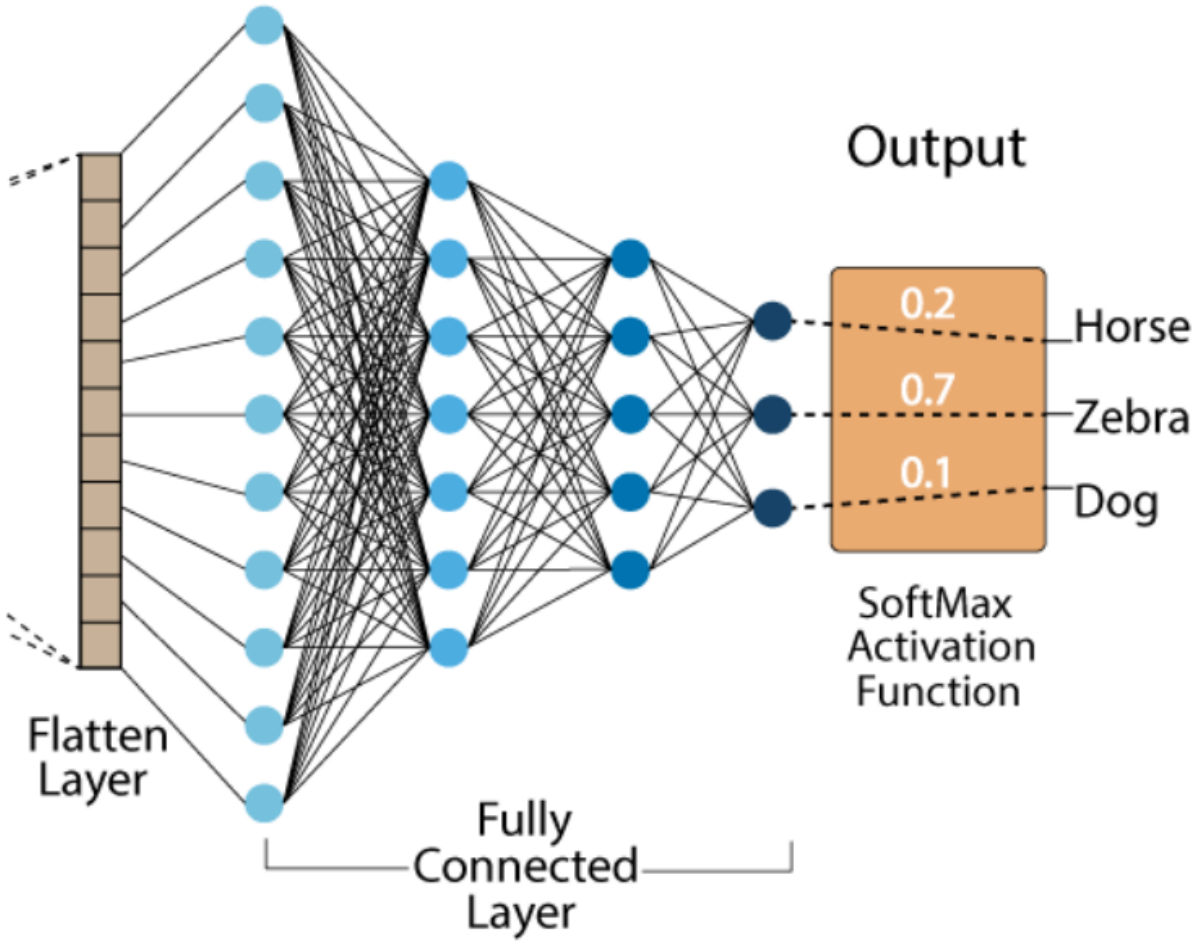


Figura 3. Predição após a rede neural densa. A imagem foi classificada como Zebra, pois obteve o maior valor de 0,7.

1.7 FUNÇÃO DE ATIVAÇÃO

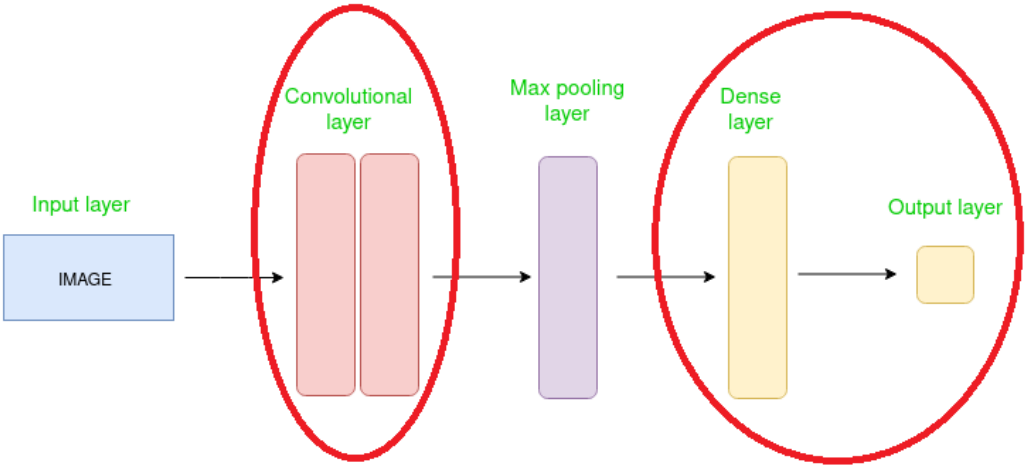


Figura 1. Estrutura simples de CNN - Função de ativação

Função de ativação é uma função matemática, em que se entra com valor do resultado do neurônio, e sai o resultado dessa função. Explicando de forma rústica ela ativa, desativa ou modifica o resultado do neurônio baseado em seus resultados. Existem várias funções de ativação, veja algumas na figura 2.

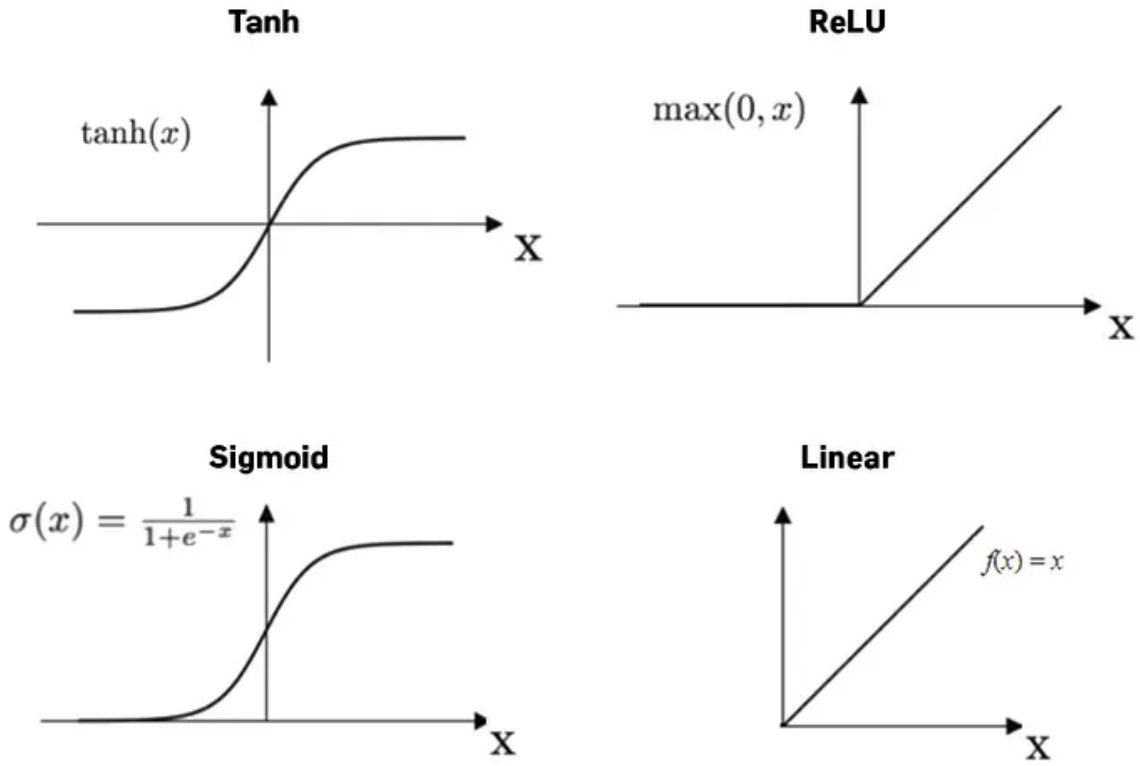


Figura 2. Funções de ativação

:

Neurônio 1	max(0,3) = 3
Neurônio 2	max(0,1) = 1
Neurônio 3	max(0,-2) = 0

Tabela 1. Exemplo de uma rede neural com 3 neurônios, em que as suas saídas obtiveram os seguintes resultados: 3, 1 e -2. Utilizando Relu iremos obter os seguintes resultados 3, 1 e 0.

Neurônio 1	$\sigma(3)= \frac{1}{1+e^{-3}}$ = 0,9525741268224332
Neurônio 2	$\sigma(1)= \frac{1}{1+e^{-1}}$ = 0,7310585786300049
Neurônio 3	$\sigma(-2)= \frac{1}{1+e^{-(-2)}}$ = 0,1192029220221176

Tabela 2. Se ao invés de Relu aplicarmos Sigmoid.

1.8. VISÃO GERAL

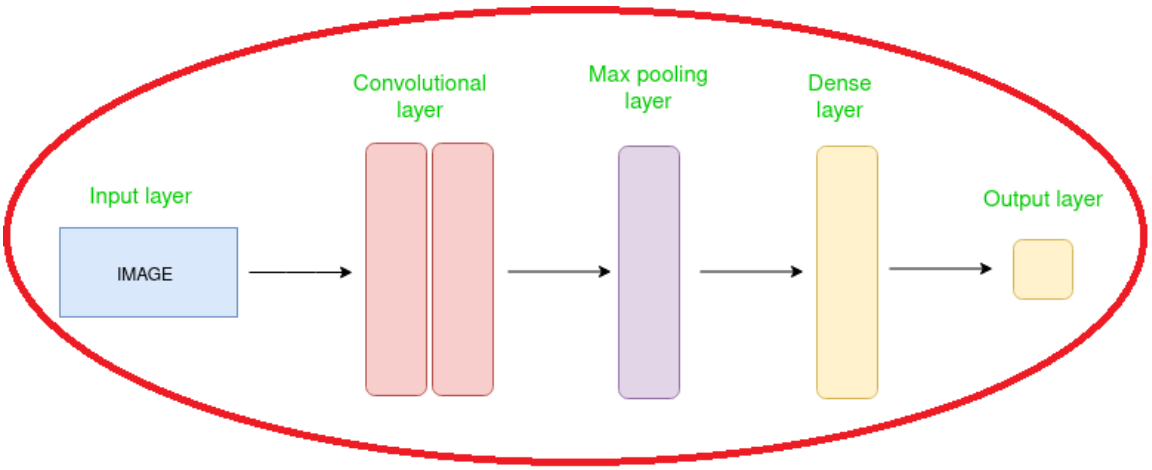


Figura 1. Estrutura simples de CNN - Visão geral

A CNN passa por um processo de treinamento, para calibrar seus pesos e após isso é possível fazer predições com o modelo feito. Ela passa por diversas camadas, como as camadas convolucionais, *pooling* e rede neural densa.

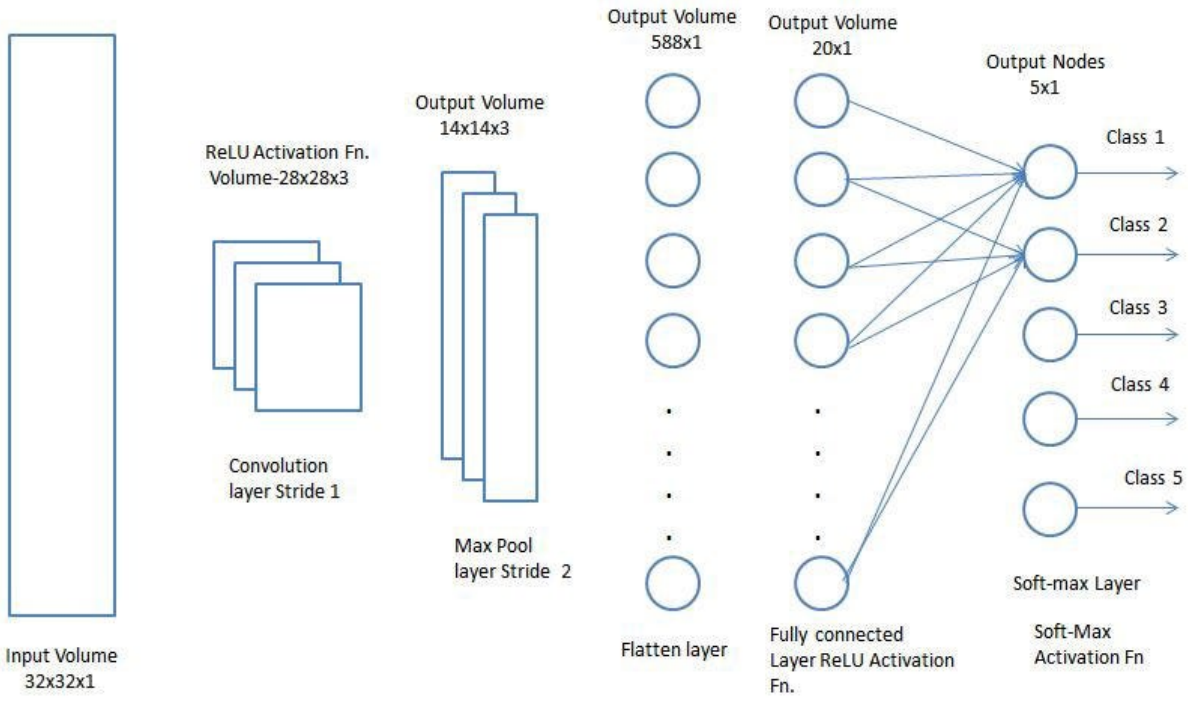


Figura 2. Exemplo de uma CNN e as dimensões dos objetos

:

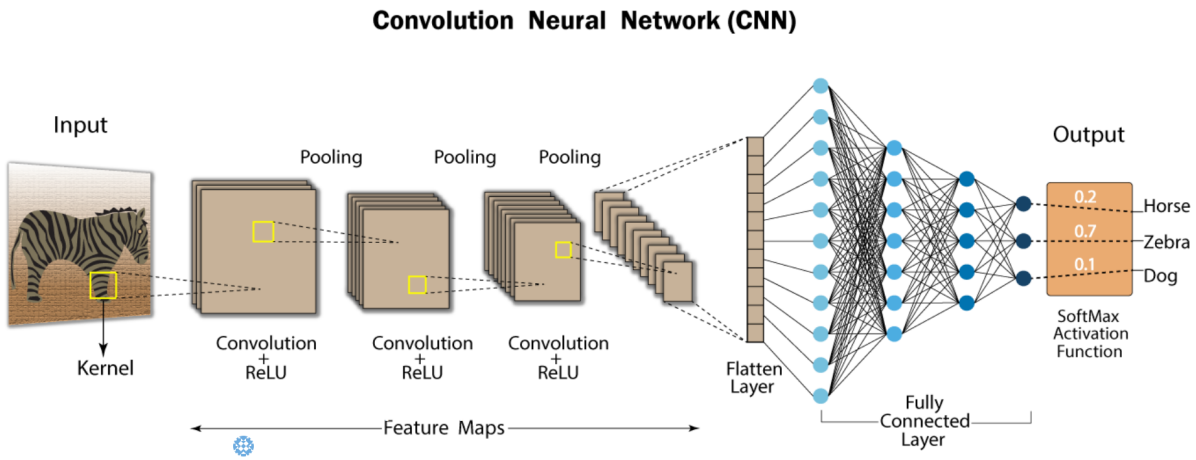


Figura 3. Exemplo de CNN classificando uma imagem como Zebra

1.9. TREINAMENTO

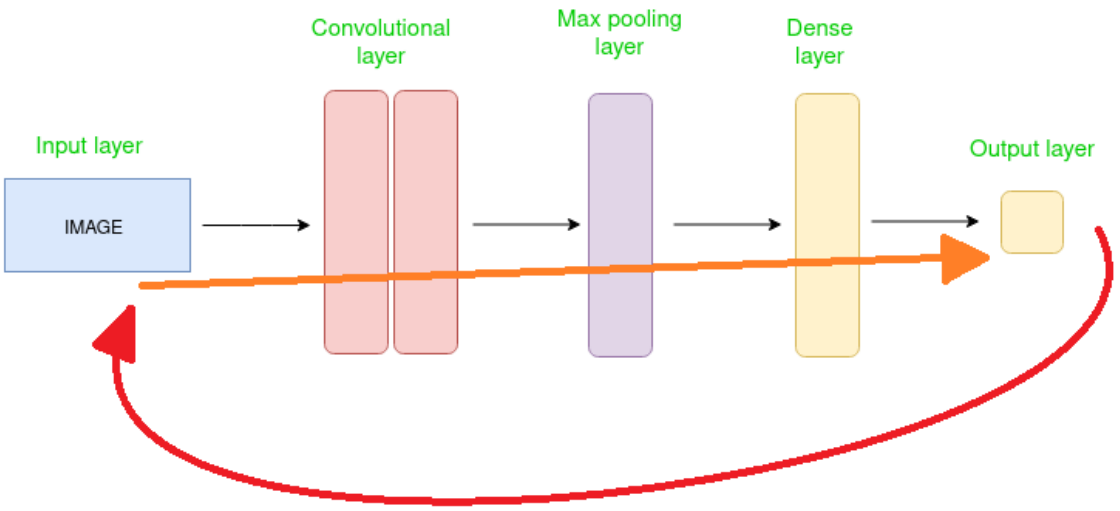


Figura 1. Estrutura simples de CNN - Treinamento

O processo de treinamento ajusta os valores dos pesos dos *kernels* e dos neurônios. Para futuramente utilizar a CNN treinada, com os pesos adequados, para a predição.

Backpropagation é o processo para ajustar os pesos do CNN com o objetivo de minimizar o erro. O processo de atualizar os pesos se repete até um valor desejado.

Época, ou *epoch*, é cada passagem pelo processo de atualizar os erros no backpropagation.

Learning Rate é um valor que define o passo ou velocidade de aprendizagem, se o valor for alto chegará mais rapidamente no resultado, mas possivelmente terá dificuldades em convergir. E valores muito baixos acontecerá o oposto.

Função de custo, ou *Loss Function*, é uma função matemática que define o quão distante se está do resultado desejado, para regressão geralmente se usa a *Mean Square Error (MSE)* e para classificação se usa geralmente *Cross-Entropy*.

Gradiente descendente, ou *Gradient Descent*, é um algoritmo que optimiza a busca pelo erro mínimo usando a função de custo. Gradient Descent usa derivada para aumentar ou diminuir o passo para achar o erro mínimo.

Batch size é a quantidade de amostras que é utilizada antes de atualizar os pesos. Temos três definições: *Stochastic* (atualiza os erros a cada amostra), *Batch* (atualiza os erros depois de toda amostra) e *Mini-Batch* (é o intermediário entre os dois, atualiza os erros depois de uma quantidade definida de amostras)

1.10. PREDIÇÃO

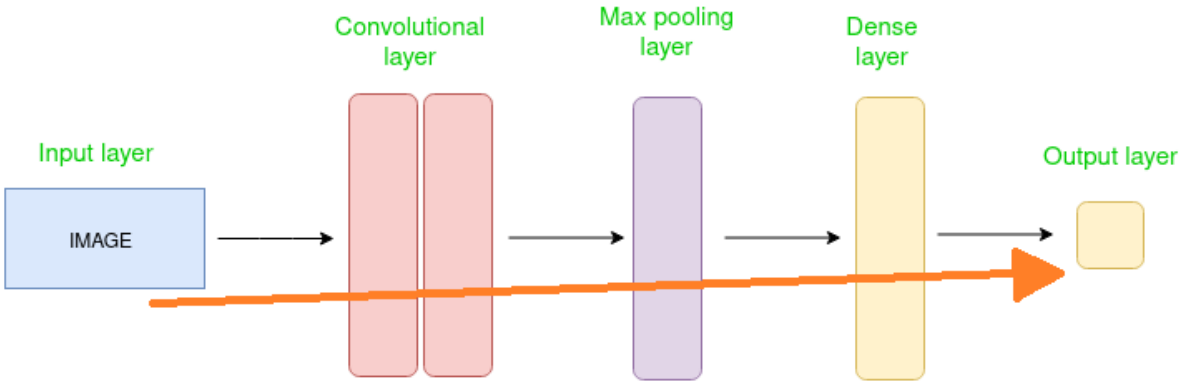


Figura 1. Estrutura simples de CNN - Predição

Após o treinamento com os ajustes dos pesos, é possível usar a CNN para prever casos. A predição irá utilizar de uma imagem para Input, passará por todo processo de convolução, pooling, rede neural densa, e finalmente irá computar dizendo, caso seja um modelo de classificação, a que classe essa imagem pertence.

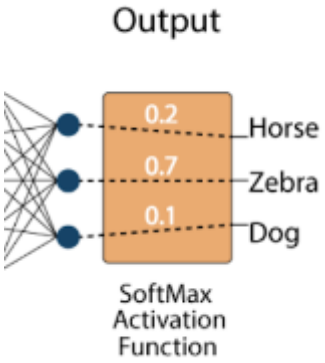


Figura 2. Predição de Zebra

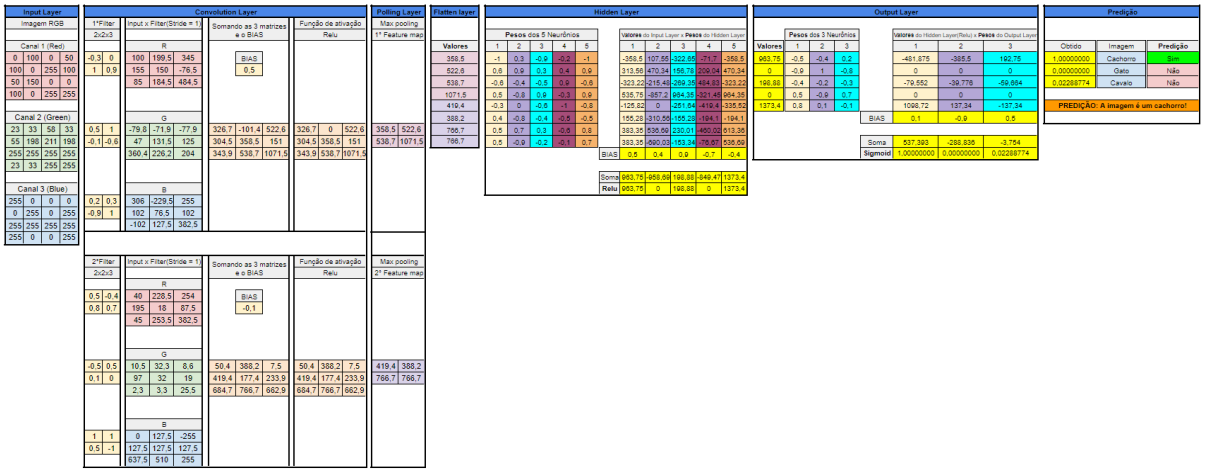


Figura 3. Foi criado uma planilha para acompanhar todo processo de predição de um algoritmo de CNN junto com este material, confira para mais detalhes.

2. CÓDIGO COMPLETO

Segue um exemplo de código de CNN utilizando a base de dados do Mnist, para identificar dígitos numéricos de 0 a 9.

Código de treinamento:

```
# Carregamento das bibliotecas
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical

# Carrega o dataset MNIST que contém dígitos escritos à mão
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Pré-processamento dos dados:
# Redimensiona as imagens de treinamento e teste para um tensor 4D (amostras, altura,
largura, canais)
# Normaliza os valores dos pixels para o intervalo [0, 1]
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

# Transforma os rótulos de treinamento e teste em vetores binários (one-hot encoding)
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Constrói o modelo da Convolutional Neural Network (CNN):
model = Sequential()
# Adiciona uma camada de convolução com 32 filtros 3x3 e função de ativação ReLU
# A entrada tem o formato (28, 28, 1) correspondente a imagens em escala de cinza de 28x28
pixels
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
# Adiciona uma camada de MaxPooling 2x2 para redução de dimensões
model.add(MaxPooling2D((2, 2)))
# Adiciona outra camada de convolução com 64 filtros 3x3 e função de ativação ReLU
model.add(Conv2D(64, (3, 3), activation='relu'))
# Adiciona outra camada de MaxPooling 2x2 para redução de dimensões
model.add(MaxPooling2D((2, 2)))
# Adiciona outra camada de convolução com 64 filtros 3x3 e função de ativação ReLU
model.add(Conv2D(64, (3, 3), activation='relu'))
# Transforma os dados 2D em um vetor 1D para a etapa de classificação
model.add(Flatten())
# Adiciona uma camada densa com 64 neurônios e função de ativação ReLU
model.add(Dense(64, activation='relu'))
# Adiciona a camada de saída com 10 neurônios correspondentes aos 10 dígitos (0-9)
# A função de ativação softmax é usada para transformar as saídas em probabilidades
model.add(Dense(10, activation='softmax'))

# Compila o modelo:
# Configura o otimizador "Adam", a função de perda "categorical_crossentropy" e a métrica
"accuracy"
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
# Treina o modelo usando os dados de treinamento:
# Realiza 5 épocas de treinamento, usando um tamanho de lote (batch size) de 64
model.fit(train_images, train_labels, epochs=5, batch_size=64)

# Avalia o modelo no conjunto de teste:
# Calcula a perda e a acurácia usando os dados de teste
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
print(f'Acurácia no conjunto de teste: {test_accuracy}')
```

Resultados do treinamento:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/5
938/938 [==============================] - 61s 63ms/step - loss: 0.1846 - accuracy: 0.9444
Epoch 2/5
938/938 [==============================] - 59s 63ms/step - loss: 0.0512 - accuracy: 0.9838
Epoch 3/5
938/938 [==============================] - 58s 62ms/step - loss: 0.0362 - accuracy: 0.9887
Epoch 4/5
938/938 [==============================] - 59s 63ms/step - loss: 0.0274 - accuracy: 0.9916
Epoch 5/5
938/938 [==============================] - 60s 64ms/step - loss: 0.0217 - accuracy: 0.9935
313/313 [==============================] - 3s 10ms/step - loss: 0.0297 - accuracy: 0.9906
Acurácia no conjunto de teste: 0.9905999898910522
```

Código da predição:

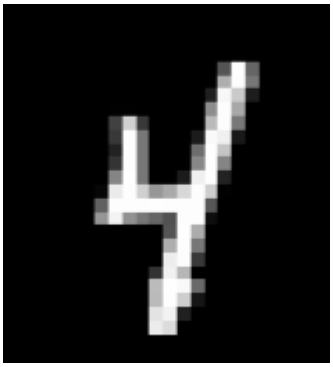
```
# Carregamento das bibliotecas
from tensorflow.keras.preprocessing.image import load_img, img_to_array

# Carrega a imagem que você deseja prever:
new_image_path = '/content/4.png'
new_image = load_img(new_image_path, color_mode='grayscale', target_size=(28, 28))
new_image_array = img_to_array(new_image)
new_image_array = new_image_array.reshape((1, 28, 28, 1)).astype('float32') / 255

# Faz a previsão usando o modelo treinado:
predictions = model.predict(new_image_array)
predicted_label = np.argmax(predictions[0])

print(f'A imagem foi classificada como o dígito: {predicted_label}')
```

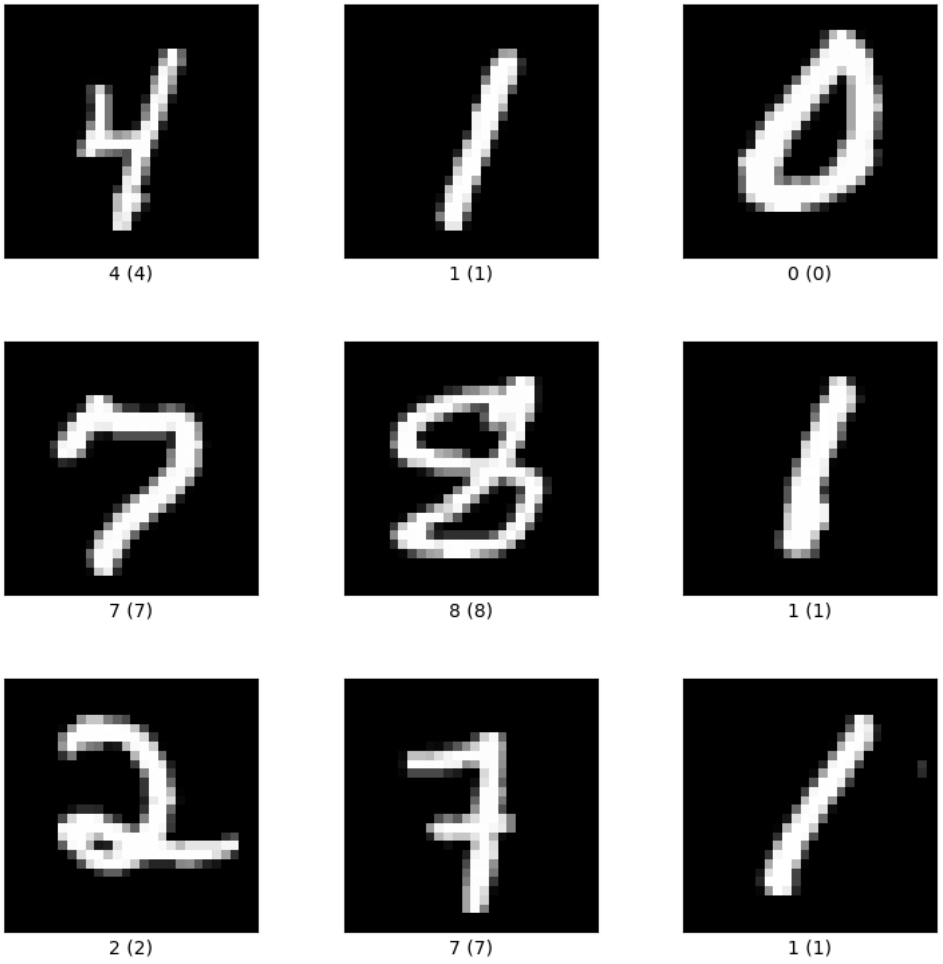
Imagem carregada como “4.png”:



Resultados da predição:

```
1/1 [==============================] - 0s 33ms/step
A imagem foi classificada como o dígito: 4
```

Outros exemplos de predição:



3. CONCLUSÃO

Foram explicados conceitos básicos de uma CNN, de forma simples com imagens. Foi feito uma planilha, explicando cada passo, com os cálculos, mostrando um exemplo de predição. Adicionalmente foi feito um código em Python, mostrando o treinamento e a predição de dígitos numéricos.

Para trabalhos futuros, pode-se indicar fazer uma planilha de treinamento com Backpropagation usando conceitos de Loss function, Gradient descent e Batch size para um melhor entendimento. Também sugere-se criar ou buscar imagens que explicam visualmente melhor alguns dos termos discutidos aqui.

4. REFERÊNCIAS

1) https://en.wikipedia.org/wiki/Convolutional_neural_network
2) https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/
3) https://www.geeksforgeeks.org/introduction-convolution-neural-network/
4) https://adamharley.com/nn_vis/cnn/2d.html
5) https://developersbreach.com/convolution-neural-network-deep-learning/
6) https://docs.gimp.org/2.8/pt_BR/plug-in-convmatrix.html
7) https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/
8) https://towardsaws.com/activation-function-f76bfc03e215
9) https://www.geeksforgeeks.org/applying-convolutional-neural-network-on-mnist-dataset/
10) https://www.tensorflow.org/datasets/catalog/mnist?hl=pt-br
11) https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/
12) https://datahacker.rs/one-layer-covolutional-neural-network/