



# Datasets - Redes Neurais

Digit Recognizer

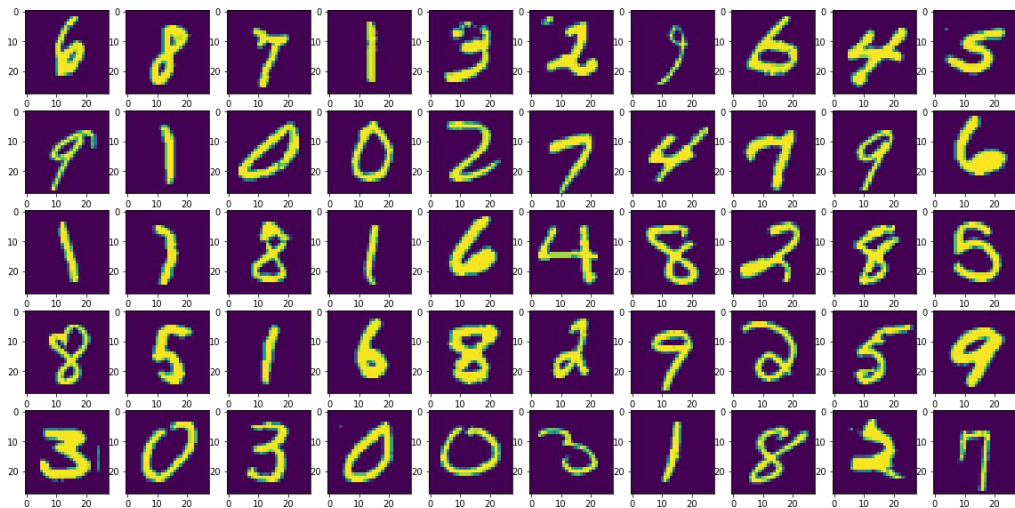
Rafael Garibaldi - 3041  
Vitor Hugo - 3049

# A competição - Identificar números escritos à mão

---


# Organização dos dados

- O dataset contém 28000 imagens de números escritos à mão.
- Os dados são 783 colunas (28x28 pixels) que variam de 0 a 255 em intensidade de preto.
- O objetivo é identificar corretamente esses números e produzir uma linha contendo o ID da imagem e o número identificado nela.



**Desenvolvendo uma rede  
neural para resolver o  
problema**

—



## Versão 1 - Loss: 0.088, acc: 0.983, kaggle: 0.97

```
def get_model(img_rows, img_cols, num_classes, x, y, x_valid=None, y_valid=None):  
  
    model = Sequential()  
  
    model.add(Conv2D(20, kernel_size=(3, 3),  
                    activation='relu',  
                    input_shape=(img_rows, img_cols, 1)))  
  
    model.add(Conv2D(20, kernel_size=(3, 3), activation='relu'))  
  
    model.add(Flatten())  
  
    model.add(Dense(128, activation='relu'))  
  
    model.add(Dense(num_classes, activation='softmax'))  
  
    model.compile(loss=keras.losses.categorical_crossentropy,  
                optimizer='adam',  
                metrics=['accuracy'])  
  
    tf.function(_fit_eval(model, x, y, x_valid, y_valid), jit_compile=True, experimental_follow_type_hints=True)  
  
    return model
```



## Versão 2 - Loss: 0.053, acc: 0.988, kaggle: 0.98567

```
def get_model(img_rows, img_cols, num_classes, x, y, x_valid=None, y_valid=None):  
    model = Sequential()  
    model.add(Conv2D(32, kernel_size=(3, 3),  
                    activation='relu',  
                    input_shape=(img_rows, img_cols, 1)))  
  
    model.add(Conv2D(64, (3, 3), activation='relu'))  
  
    model.add(MaxPooling2D(pool_size=(2, 2)))  
  
    model.add(Dropout(0.25))  
  
    model.add(Flatten())  
  
    model.add(Dense(128, activation='relu'))  
  
    model.add(Dropout(0.5))  
  
    model.add(Dense(num_classes, activation='softmax'))  
  
    model.compile(loss=keras.losses.categorical_crossentropy,  
                optimizer='adam',  
                metrics=['accuracy'])  
  
    tf.function(_fit_eval(model, x, y, x_valid, y_valid), jit_compile=True, experimental_follow_type_hints=True)  
  
    return model
```



## Versão 3 - Loss: 0.4, acc: 0.89

```
def get_model(img_rows, img_cols, num_classes, x, y, x_valid=None, y_valid=None):  
  
    img_rows, img_cols = 28, 28  
    num_classes = 10  
  
    model = models.Sequential()  
  
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(img_rows, img_cols, 1)))  
    model.add(Conv2D(64, (3, 3), activation='relu'))  
    model.add(Dropout(0.25))  
    model.add(Flatten())  
    model.add(Dense(128, activation='relu'))  
    model.add(Dropout(0.5))  
    model.add(Dense(num_classes, activation='softmax'))  
  
    model.compile(loss=keras.losses.categorical_crossentropy, optimizer='Adadelta', metrics=['accuracy'])  
    tf.function([fit_eval(model, x, y, x_valid, y_valid), jit_compile=True, experimental_follow_type_hints=True])  
  
    return model
```



## Versão 4 - Loss: 0.07, acc: 0.98, kaggle: 0.98628(822)

```
def get_model(img_rows, img_cols, num_classes, x, y, x_valid=None, y_valid=None):
    model = Sequential()

    model.add(Conv2D(200, kernel_size=(2, 2),
                    activation='relu',
                    input_shape=(img_rows, img_cols, 1)))

    model.add(AveragePooling2D(pool_size=2))

    model.add(DepthwiseConv2D(kernel_size=(2, 2), strides=(1, 1), padding='same', depth_multiplier=1,
                              input_shape=(img_rows, img_cols, 1)))

    model.add(Conv2D(100, kernel_size=(6, 6),
                    activation='relu',
                    padding='same'))

    model.add(Conv2D(num_classes, kernel_size=(8, 2),
                    activation='softmax',
                    padding='same'))

    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))

    model.add(Masking(mask_value=0))

    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    tf.function(_fit_eval(model, x, y, x_valid, y_valid), jit_compile=True, experimental_follow_type_hints=True)
```





## Versão 5 - Loss: 0.084, acc: 0.98, kaggle: 0.98939(661)

```
def get_model(img_rows, img_cols, num_classes, x, y, x_valid=None, y_valid=None):
    model = Sequential()

    model.add(Conv2D(200, kernel_size=(2, 2),
                     activation='relu',
                     input_shape=(img_rows, img_cols, 1)))

    model.add(DepthwiseConv2D(kernel_size=(1, 5), strides=(1, 1), padding='same', depth_multiplier=1,
                              input_shape=(img_rows, img_cols, 1)))

    model.add(SeparableConv2D(filters=100, kernel_size=(4, 4),
                               activation='relu', padding='same'))

    model.add(Conv2D(50, kernel_size=(5, 1), activation='relu'))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))

    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss=keras.losses.categorical_crossentropy,
                  optimizer='adam',
                  metrics=['accuracy'])

    tf.function(_fit_eval(model, x, y, x_valid, y_valid), jit_compile=True, experimental_follow_type_hints=True)

    return model
```



## Versão ???

Apesar de não termos conseguido nenhum avanço, durante a primeira reorganização da rede neural, conseguimos atingir um *loss* de mais de 6 octilhões ( $6 \cdot 10^{27}$ ), e isso só nos colocou no top 97%.

**1,000,000,000,000,000,000,000,000,000,000**

# Conclusão e melhor Submissão

---



## Posição

A melhor posição que conseguimos foi por meio da versão 3 do algoritmo, o que nos colocou em top 26,7% da competição com um score de 98.9% de acerto.

661

**Vítor Hugo**



0.98939

5

1s



Your Best Entry!

Your most recent submission scored 0.98939, which is an improvement of your previous score of 0.98628. Great job!

**Tweet this**



## Conclusão

Através do trabalho pudemos perceber como é difícil otimizar uma rede neural para que ela atinja a “perfeição”, pois nossas melhores tentativas sempre resultaram em redes com acertos de ~98%, mas nunca era possível quebrar esse recorde, apenas se distanciar dele.

Percebemos, também, que redes neurais podem ser extremamente voláteis e exigem muita experiência para ajustes; por mais que seja fácil mudar as camadas, inicializadores e demais técnicas que irão guiar a rede, suas escolhas exigem conhecimento para fazerem sentido. Escolhas erradas ou mal pensadas irão impactar negativamente o desempenho do algoritmo, podendo torná-lo inviável para a classificação de um determinado problema.