

Universidade la salle

Trabalho IA – Treinando uma CNN para reconhecer letras de A-Z em estilo Handwritted

Natanael Ferreira

Canoas, 2025

Whitepaper - Machine Learning with Kaggle Database A-Z

Recursos de entrada e recursos gerados:

- Para treinamento: Base de dados de Imagens já convertidos em Vetorial (Pode ser encontrado no seguinte link: <https://www.kaggle.com/datasets/sachinpatel21/az-handwritten-alphabets-in-csv-format>)
- Conjunto de imagens 28x28px em escala de cinza (pode ser encontrado no seguinte link: <https://medium.com/analytics-vidhya/optical-character-recognition-using-tensorflow-533061285dd3>)
- Código para converter a imagem completa em pequenas imagens 28x28 com apenas um caractere handwritten ("separateImages.py")
- Código para testar o modelo ("TestarOModelo.py")
- Código que implementa e roda o modelo de Deep Learning ("__main__.py")
- Modelo gerado pelo código ("modelo_letras.keras")
- Referência de performance do código, acurácia por época ("./results")

Descrição completa do código (__main__.py):

1-16: Cabeçalho e importação de bibliotecas

22: Carrega a base de dados

25: Define a coluna que representará as letras. Ex: 0=A, 1=B...

26: DATASET.drop exclui somente a coluna 0 e mantém os demais para vetores de letra

29: Normalização, para evitar convergência demorada ou que não aprender, também evita explosão de valores que crescem demais, como IA trabalho com Pesos normalizar é ideal

```
22 DATASET = pd.read_csv("./dataset/A_Z-Handwritten-Data.csv")
23
24 # Separa rótulos (letras) e imagens
25 y = DATASET['0'].values
26 X = DATASET.drop(labels='0', axis=1).values
27
28 # Normaliza os pixels (0-255 → 0-1)
29 X = X / 255.0
```

33: Precisa ser convertido novamente neste formato pois Redes Convolucionais – CNNs não funcionam com vetores planos, usam estrutura espacial com altura, largura e profundidade (canais), 1 canal pois é grayscale. CNNs aplicam kernels sobre a imagem.

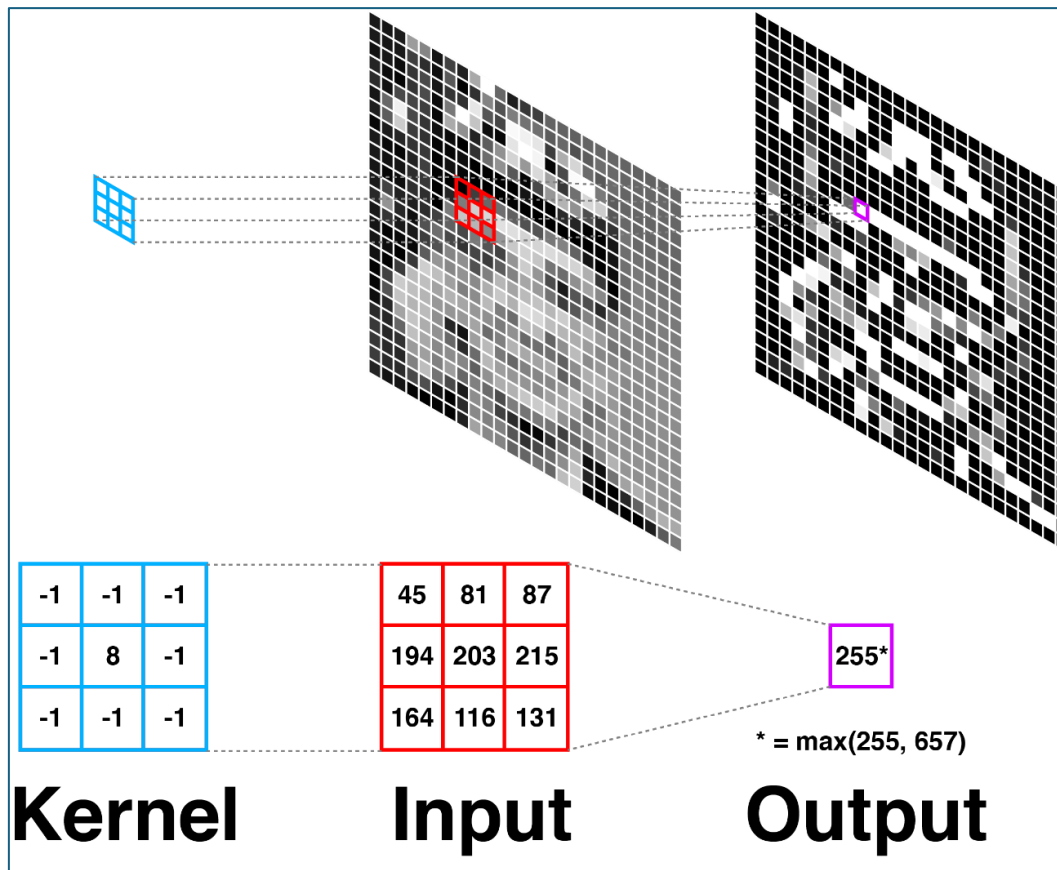



Figura 1 - <https://gregorygundersen.com/image/cnns/41847277995.png>

35: Transforma os rótulos (y) — que originalmente são **números inteiros representando letras** — em **vetores one-hot**.

 **O que é codificação one-hot?**

Imagine que temos 4 classes (A, B, C, D), codificadas como:

Letra	Label original (inteiro)	One-hot
A	0	[1, 0, 0, 0]
B	1	[0, 1, 0, 0]
C	2	[0, 0, 1, 0]
D	3	[0, 0, 0, 1]

Figura 2 -Gerado por IA - Chat GPT

Pela função de saída da rede ser Softmax ela espera produzir um vetor com 26 probabilidades: [0.01, 0.00, 0.05, ..., 0.91, ..., 0.00]

 **Comparação com outras ativações**

Função	Aplicação típica	Faixa de saída	Somam 1?	Uso principal
<code>sigmoid</code>	Saída binária	(0, 1)	✗	Classificação binária
<code>tanh</code>	Camadas intermediárias	(-1, 1)	✗	Captura valores positivos e negativos
<code>ReLU</code>	Camadas intermediárias	[0, ∞)	✗	Rápido, evita gradientes nulos
<code>softmax</code>	Última camada (multiclasse)	(0, 1)	✓	Classificação multiclasse exclusiva (ex: letras)

Figura 3 - Gerado por IA - ChatGPT

37:Separa a porcentagem de dados de treino(80%) e de teste(20%), para que a rede treine com dados não vistos ainda.

```
33 X = X.reshape(-1, 28, 28, 1)
34
35 y = to_categorical(y, num_classes=26)
36
37 X_train, X_test, y_train, y_test = train_test_split(*arrays: X, y, test_size=0.2)
```

51:Cria o modelo que será usado para treinar
O que cada atributo da configuração faz:

Etapa	Função	Entrada	Saída	Descrição
Conv2D(32, 3x3)	Detector de bordas e texturas	(28, 28, 1)	(26, 26, 32)	Extrai padrões locais
MaxPooling2D(2x2)	Compressão e destaque de padrões fortes	(26, 26, 32)	(13, 13, 32)	Reduz complexidade e mantém o essencial

Conv2D(64, 3x3)	Captura de padrões mais abstratos	(13, 13, 32)	(11, 11, 64)	Reconhece formas mais complexas
MaxPooling2D(2x2)	Nova compressão	(11, 11, 64)	(5, 5, 64)	Garante invariância espacial
Flatten()	Prepara para camadas densas	(5, 5, 64) = 1600	vetor(1600)	Transforma imagem em vetor
Dense(128)	Classificador parcial	(1600)	(128)	Combina padrões extraídos
Dropout(0.5)	Regularização	(128)	(128)	Evita overfitting desativando alguns perceptrons
Dense(26, softmax)	Classificador final	(128)	(26)	Gera probabilidades para cada letra (A-Z)

```

51 model = models.Sequential([
52     layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
53     layers.MaxPooling2D((2, 2)),
54
55     layers.Conv2D(64, (3, 3), activation='relu'),
56     layers.MaxPooling2D((2, 2)),
57
58     layers.Flatten(),
59     layers.Dense(128, activation='relu'),
60     layers.Dropout(0.5), # Ajuda a evitar overfitting
61     layers.Dense(26, activation='softmax') # 26 letras
62 ])

```

65: ADAM - Adaptive Moment Estimation, Imagine que o modelo previu errado a letra "A" como "C", o modelo ajusta todos os pesos conectados ao padrão da letra "A" para tentar diminuir esse erro na próxima época, de forma adaptativa e inteligente

66: A função de perda (loss) mede o quão errado o modelo está nas previsões. O treinamento tenta minimizar esse valor

67: Mede a porcentagem de acertos (predição com maior probabilidade bate com o índice da label real)

```

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

```

70-75: Inicia o treinamento da CNN, define a quantidade de épocas, define os dados em lotes de 128 amostras/imagens para treinar por etapa e define os dados que serão usados para medir a performance de cada época.

```

70     history = model.fit(
71         X_train, y_train,
72         epochs=10,
73         batch_size=128,
74         validation_data=(X_test, y_test)
75     )

```

78: Salva o modelo em formato .keras

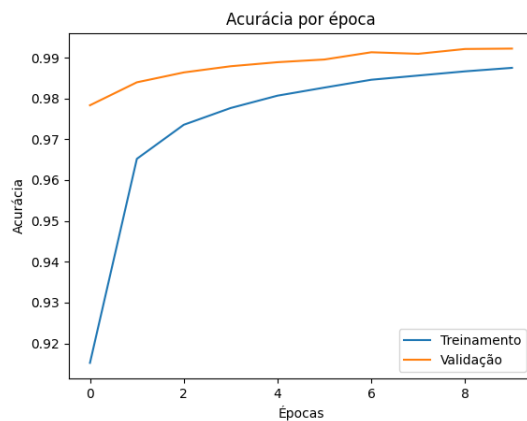
```

78     model.save("modelo_letras.keras")

```

82-88: Gera um gráfico apenas informativo para mostrar a acurácia do treinamento.

Resultado:



```

82     plt.plot(*args: history.history['accuracy'], label='Treinamento')
83     plt.plot(*args: history.history['val_accuracy'], label='Validação')
84     plt.title("Acurácia por época")
85     plt.xlabel("Épocas")
86     plt.ylabel("Acurácia")
87     plt.legend()
88     plt.show()

```

91-92: Executa uma predição completa sobre todos os dados **X_test**, depois compara as predições com os rótulos verdadeiros **y_test**.

```

91     loss, accuracy = model.evaluate(X_test, y_test)
92     print(f"Acurácia final: {accuracy:.4f}")

```

Descrição completa do código(TestarOModelo.py):

Overview Geral:

O código busca uma imagem armazenada na pasta “./ImageToTest” de tamanho 28x28 pixels com o nome “imageTest.png” converte em formato vetorial seguindo o padrão da base dados e a partir do modelo gerado, faz a predição, converte o resultado para letra e gera uma saída da letra predita.

Como treinar o modelo:

1. **Baixe a base de dados nesse link:**
<https://www.kaggle.com/datasets/sachinpatel21/az-handwritten-alphabets-in-csv-format>
2. Rode o código

Como testar o modelo:

1. Coloque uma imagem com 28x28 pixels, em formato png dentro da pasta “./ImageToTest/ imageTest.png” e rode o código.
2. Há uma série de imagens para treinamento na pasta “./dataset”