

Keras: Democratizando o Acesso ao Aprendizado Profundo

Adryel S. L. Almeida, Felipe S. de Lima, Lucas C. Junker

Orientador: Professor Ricardo Pires Mesquita

Centro Universitário Carioca – UniCarioca

2.2025

{adryel7, lima.felipesalles, lcsjunker}@gmail.com

Abstract. This study investigates the essential role of the Keras library in the Deep Learning ecosystem, emphasizing its architecture as a user-friendly, high-level API. Its evolution to Keras 3 introduced multi-backend support (JAX, TensorFlow, PyTorch), effectively democratizing Neural Network development. The methodology involved implementing a Multi-Layer Perceptron (MLP) on the MNIST dataset using Keras's intuitive workflow. The model achieved a robust 97.65% accuracy on the test set, demonstrating Keras's capability for concise yet powerful solutions. The work concludes that Keras is a cornerstone of modern AI, providing an optimal balance of simplicity, flexibility, and computational performance.

Resumo. O trabalho investiga o papel essencial do Keras no Deep Learning, destacando sua arquitetura como uma API de alto nível focada em acessibilidade. Sua evolução para a versão Keras 3 oferece suporte multiplataforma (JAX, TensorFlow, PyTorch), democratizando o desenvolvimento de Redes Neurais. A metodologia utilizou o Keras para implementar um MLP na base de dados MNIST, alcançando uma acurácia de 97,65%. A conclusão é que o Keras é um pilar da IA moderna, combinando simplicidade, flexibilidade e desempenho computacional.

1. Introdução

A Inteligência Artificial (IA) e o aprendizado profundo (*Deep Learning*) têm transformado drasticamente diversos setores, mas seu desenvolvimento exige, historicamente, o domínio de bibliotecas complexas, o que impõe uma barreira de entrada significativa. É neste contexto que a biblioteca Keras surge como uma solução fundamental. Criada por François Chollet e lançada em 2015, a biblioteca foi concebida para simplificar a experimentação e tornar o Deep Learning acessível. Sua arquitetura de alto nível, sua integração posterior como API padrão do TensorFlow e sua recente evolução para a plataforma *multi-backend* (Keras 3) foram cruciais para sua ampla adoção na academia e na indústria.

O presente trabalho tem como objetivo principal examinar a relevância do Keras como ferramenta de democratização e produtividade no desenvolvimento de modelos de aprendizado profundo, comprovando sua eficácia através de um experimento prático.

Para isso, o estudo se estrutura em duas partes: a primeira, teórica, explora na Seção 2 a Origem e o Desenvolvimento do Keras, detalhando sua arquitetura e a flexibilidade *multi-backend*. A segunda parte, aplicada, inicia com a Metodologia (Seção 3), que detalha o experimento de classificação baseado na base de dados MNIST. O trabalho culmina na Implementação do Modelo com Keras (Seção 4), onde são apresentados o desenvolvimento, o treinamento e a avaliação de desempenho da rede neural (MLP), validando a robustez da solução alcançada.

2. Origem e Desenvolvimento

O Keras foi criado por François Chollet e lançado no final de março de 2015 (CHOLLET, 2016). A ferramenta surgiu da necessidade de simplificar a experimentação com redes neurais profundas, até então marcada por elevada complexidade (CHOLLET, 2016). Em 2015, com o lançamento do TensorFlow, passou-se a adotá-lo como *backend* computacional do Keras. Posteriormente, adicionou-se suporte ao CNTK (*Microsoft Cognitive Toolkit*). Essa capacidade de operação em múltiplos *backends* constituiu um diferencial relevante, permitindo a execução de códigos em diferentes “motores de computação” sem alterações significativas. (KERAS, 2024)

O ponto de inflexão ocorreu em 2017, quando o Keras foi adotado como API de alto nível padrão do TensorFlow. A partir da versão 1.4 do TensorFlow, a biblioteca passou a integrar-se diretamente via tf.keras. Em 2019, com o TensorFlow 2.0, essa integração consolidou-se e o Keras tornou-se a principal forma de construção de modelos na plataforma, constituindo sua interface pública para a maioria dos usuários. (TENSORFLOW, 2025)

Em 2023, o Keras foi reestruturado como uma API multiplataforma e lançado como Keras 3 (ou Keras Core), com suporte *multi-backend*, permitindo o uso combinado de tecnologias como TensorFlow, PyTorch e JAX. Essa versão tornou o Keras mais flexível, operando em várias plataformas e suportando múltiplos *frameworks* com reaproveitamento de código. (KERAS, 2024)

Atualmente, embora o projeto Keras 3 retome a visão *multi-backend* (KERAS, 2024), a identidade mais forte no mercado e na academia permanece associada à interface padrão e acessível do ecossistema TensorFlow. (TENSORFLOW, 2025)

2.1. Arquitetura e Princípios Fundamentais

O Keras consolidou-se como uma das ferramentas proeminentes no campo da inteligência artificial, atuando como uma API de aprendizado profundo (*Deep Learning*) (KERAS, 2024). Escrito em Python, foi projetado para simplificar a construção e o treinamento de modelos de redes neurais, tornando a tecnologia mais acessível (TENSORFLOW, 2025). A versatilidade permite execução sobre JAX, TensorFlow e PyTorch, conferindo

flexibilidade ao desenvolvimento. (KERAS, 2024)

2.2. Filosofia de Design: Usabilidade e Acessibilidade

A motivação central por trás do desenvolvimento do Keras consistiu na democratização do *Deep Learning*, com foco na acessibilidade a um público amplo, para além de especialistas. Ao oferecer uma interface simples e otimizada para casos de uso comuns, o Keras reduz a barreira de entrada imposta por bibliotecas mais complexas. O objetivo principal consistiu em fornecer “blocos de Lego” que permitissem prototipação e construção de modelos de modo rápido e intuitivo. (CHOLLET, 2016)

2.3. Características e Flexibilidade *Multi-Backend*

Uma característica marcante do Keras é a facilidade de uso, com interface consistente e mensagens claras. Os modelos são modulares e compostos, permitindo a conexão de elementos configuráveis com poucas restrições. (TENSORFLOW, 2025; CHOLLET, 2016)

Inicialmente conhecido pela forte integração com o TensorFlow, tf.keras tornou-se a API de alto nível recomendada para criação e treinamento de modelos nesse ecossistema. (TENSORFLOW, 2025)

Com o lançamento do Keras 3, a ferramenta evoluiu para uma API *multi-framework*. Isso significa que modelos desenvolvidos com Keras podem ser executados em diferentes *backends* como JAX, TensorFlow ou PyTorch, possibilitando a seleção do *backend* com melhor desempenho para cada cenário (GPU ou CPU) sem alteração de código. Essa flexibilidade também maximiza o ecossistema disponível, permitindo o uso de modelos Keras com pacotes do PyTorch, ferramentas de produção do TensorFlow e a infraestrutura de treinamento em larga escala do JAX. (KERAS, 2024)

2.4. Aplicações e Importância Atual

O poder e a escalabilidade do Keras tornaram-no uma escolha popular para aplicações industriais, sendo utilizado por grandes organizações. O uso abrange desde pesquisa acadêmica até sistemas de produção, incluindo reconhecimento de imagens, processamento de linguagem natural, diagnóstico médico e condução autônoma. (KERAS, 2024)

A relevância do Keras é reforçada por sua ampla adoção e facilidade de uso, permitindo que modelos implementados em Keras sejam utilizados independentemente do *framework* preferido (TensorFlow, PyTorch, etc.) (KERAS, 2024). Isso amplia o impacto de soluções abertas, ao eliminar custos adicionais de portabilidade entre *frameworks*. (KERAS, 2024)

Essa compatibilidade estende-se também aos fluxos de dados, sendo as rotinas de treinamento do Keras compatíveis com diferentes formatos, como tf.data.Dataset (TensorFlow) e DataLoader (PyTorch). (KERAS, 2024)

3. Metodologia

Para demonstrar a capacidade de democratização e a simplicidade de implementação do *framework* Keras, adotou-se uma abordagem de pesquisa aplicada, de natureza quantitativa e experimental. A metodologia foi estruturada para validar os conceitos teóricos por meio da construção prática de uma Rede Neural Artificial (RNA), priorizando não apenas a acurácia, mas também a reproduzibilidade científica e a robustez em cenários reais.

A seguir, apresentam-se os materiais utilizados, o tratamento dos dados e os procedimentos experimentais detalhados.

3.1. Materiais e Ferramentas Computacionais

O experimento foi desenvolvido na linguagem Python, escolhida por sua vasta biblioteca de suporte à ciência de dados. A implementação utilizou o ambiente de desenvolvimento em nuvem Google Colab, aproveitando a disponibilidade de recursos computacionais gratuitos (CPU/GPU) para o treinamento.

As principais bibliotecas empregadas foram:

- TensorFlow/Keras: Para construção da arquitetura neural, compilação e treinamento.
- Pandas e NumPy: Para manipulação algébrica de tensores e estruturação de logs de treinamento (arquivos CSV e JSON).
- Matplotlib e Seaborn: Para a geração de visualizações gráficas de desempenho e matrizes de confusão.
- Scikit-learn: Para o cálculo de métricas analíticas detalhadas (*Precision*, *Recall*, *F1-Score*).

3.2. Fonte de Dados e Preparação

O objeto de estudo foi a base de dados MNIST (*Modified National Institute of Standards and Technology*), contendo 70.000 imagens de dígitos manuscritos em escala de cinza (28×28 pixels).

Para garantir a integridade do experimento, os dados foram submetidos a um pipeline rigoroso de preparação:

- Segregação: Divisão em conjunto de treinamento (60.000 amostras) e teste (10.000 amostras).
- Normalização: Reescalonamento dos valores de pixel do intervalo inteiro [0, 255] para o intervalo flutuante [0, 1], visando acelerar a convergência do gradiente descendente.

- Remodelagem (Reshape): Ajuste dimensional dos dados de entrada para o formato ($N, 28, 28, 1$), adicionando explicitamente o canal de cor (escala de cinza), requisito necessário para operações de processamento de imagem no Keras.

3.3. Procedimentos Experimentais

O fluxo de trabalho foi desenhado para mitigar problemas clássicos de redes neurais, como o *overfitting* e a variância de resultados.

3.3.1. Reprodutibilidade Científica

Diferente de abordagens padrão, este trabalho implementou o controle estrito de aleatoriedade. Foram fixadas sementes (seeds) pseudoaleatórias (SEED = 42) para as bibliotecas **random**, **numpy**, **os** e **tensorflow**. Isso garante que a inicialização dos pesos e o embaralhamento dos dados sejam determinísticos, permitindo que os resultados obtidos sejam auditáveis e reproduzíveis. (TENSORFLOW, 2025)

3.3.2. Estratégia de Generalização (Data Augmentation)

Para superar a limitação de variância espacial (onde a rede aprende apenas dígitos centralizados), implementou-se a técnica de Aumento de Dados (*Data Augmentation*) utilizando a classe `ImageDataGenerator`. (TENSORFLOW, 2024) Durante o treinamento, as imagens originais sofreram transformações dinâmicas e aleatórias a cada época, incluindo:

- Rotação de até 15°;
- Translação horizontal e vertical de 10%;
- Zoom de 10%.

Essa estratégia obriga a rede a aprender as características estruturais invariantes do dígito, e não apenas sua posição fixa em pixels.

3.3.3. Arquitetura da Rede Neural

A topologia foi definida como um *Perceptron Multicamadas* (MLP) sequencial, estruturado para equilibrar complexidade e desempenho:

- Camada de Entrada: Flatten, convertendo a matriz (28, 28, 1) em vetor unidimensional.
- Camada Oculta: Densa (Dense) com 128 neurônios e função de ativação ReLU (Rectified Linear Unit), responsável pela não linearidade.
- Camada de Saída: Densa com 10 neurônios e ativação Softmax, gerando a distribuição de probabilidade para as classes (0 a 9).

3.3.4. Configuração e Treinamento

O modelo foi compilado com o otimizador Adam (taxa de aprendizado adaptativa) e a função de perda *Sparse Categorical Crossentropy*. O treinamento ocorreu por 15 épocas, utilizando o fluxo de dados aumentados (*augmented flow*) com lotes (*batch size*) de 32

amostras. O histórico de métricas (acurácia e perda) de treino e validação foi persistido em arquivos de log para análise posterior.

3.3.5. Avaliação de Desempenho

A validação final ocorreu no conjunto de teste (dados inéditos e não aumentados). A avaliação foi multidimensional, incluindo:

- Métricas Globais: Acurácia e Perda final.
- Relatório de Classificação: Análise de *Precision*, *Recall* e *F1-Score* por classe.
- Matriz de Confusão: Diagnóstico visual para identificar quais pares de dígitos geram maior ambiguidade para o modelo.

3.3.6. Teste Prático (Prova de Conceito)

Para validar a aplicabilidade real da solução, desenvolveu-se um módulo de inferência externa. Imagens digitais de dígitos manuscritos (criadas em softwares de desenho como Paint e KolorPaint) foram inseridas no sistema.

O script de teste implementou um pré-processamento específico para compatibilizar dados do mundo real com o padrão MNIST:

- Carregamento e conversão para escala de cinza.
- Inversão de Cores: Transformação matemática ($255 - \text{pixel}$) para converter fundos brancos com traços pretos (padrão de desenho humano) para fundos pretos com traços brancos (padrão da rede neural).
- Normalização e inferência.

4. Implementação do Modelo de Classificação com Keras

Com o objetivo de demonstrar a capacidade e a simplicidade do *framework* Keras, implementou-se uma RNA para classificação de imagens. Em consonância com a metodologia estabelecida, o processo contemplou o uso de técnicas avançadas de regularização, como o Aumento de Dados (*Data Augmentation*), para garantir a robustez do modelo.

4.1. Base de Dados: MNIST

A base MNIST (*Modified National Institute of Standards and Technology*) foi adotada como referência. O conjunto total contém 70.000 imagens (28×28 pixels) de dígitos manuscritos, divididas em 60.000 para treinamento e 10.000 para teste. Devido à introdução do gerador de imagens, os dados foram redimensionados para o formato tetradiimensional ($N, 28, 28, 1$), explicitando o canal de cor em escala de cinza.

4.2. Desenvolvimento e Configuração

4.2.1. Preparação dos Dados e Reprodutibilidade

Utilizou-se o Keras em conjunto com o TensorFlow. Além da normalização dos pixels

para o intervalo $[0, 1]$, foi estabelecido um controle rigoroso de reproduzibilidade através da fixação de sementes aleatórias ($\text{SEED} = 42$), garantindo que a inicialização dos pesos e as transformações das imagens sejam determinísticas.

4.2.2. Arquitetura do Modelo (MLP)

O modelo foi construído via API *Sequential*. A arquitetura compreende:

- *Flatten*: Conversão do tensor 3D (28, 28, 1) em vetor de 784 elementos.
- *Dense (128, activation='relu')*: Camada oculta com 128 neurônios para extração de características não lineares.
- *Dense (10, activation='softmax')*: Camada de saída retornando probabilidades para as 10 classes.

4.2.3. Configuração do Treinamento (Data Augmentation)

Para mitigar o *overfitting* espacial, o treinamento não utilizou as imagens estáticas originais, mas sim um fluxo contínuo de dados gerados dinamicamente pela classe *ImageDataGenerator*, com as seguintes transformações: rotação (15°), zoom (10%) e translações laterais/verticais (10%).

4.2.4. Análise da Evolução do Treinamento

O treinamento foi executado ao longo de 15 épocas. Os dados coletados (Tabela 1, Gráfico 1, Tabela 2 e Gráfico 2) revelam um fenômeno importante decorrente do *Data Augmentation*: a acurácia de validação manteve-se consistentemente superior à acurácia de treino.

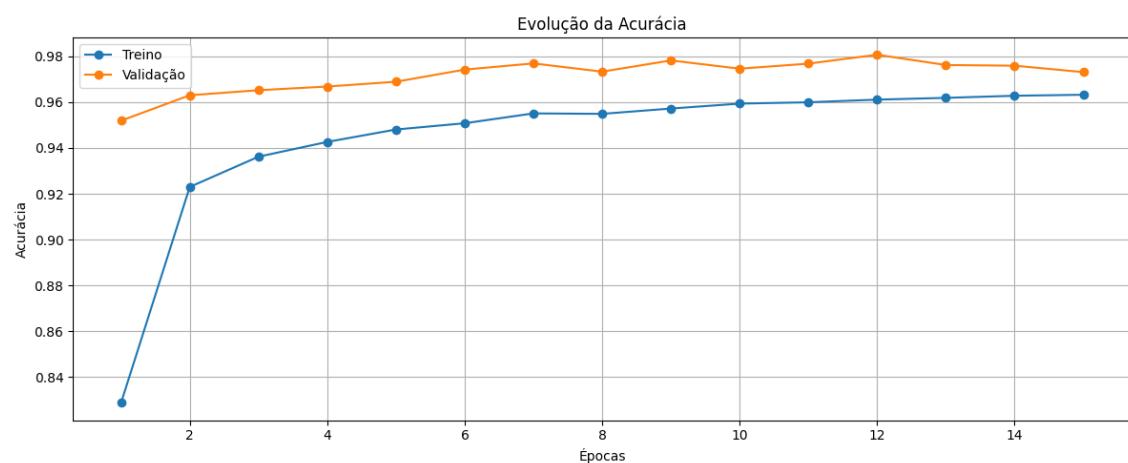


Figura 1. Evolução da Acurácia.

Tabela 1. Evolução da Acurácia.

Época	Acurácia (Treino)	Acurácia (Validação)
1	82,87%	95,19%
2	92,30%	96,30%
3	93,62%	96,52%
4	94,26%	96,68%

5	94,81%	96,89%
6	95,08%	97,42%
7	95,51%	97,69%
8	95,49%	97,33%
9	95,72%	97,82%
10	95,93%	97,46%
11	95,99%	97,68%
12	96,11%	98,07%
13	96,19%	97,62%
14	96,28%	97,59%
15	96,32%	97,31%

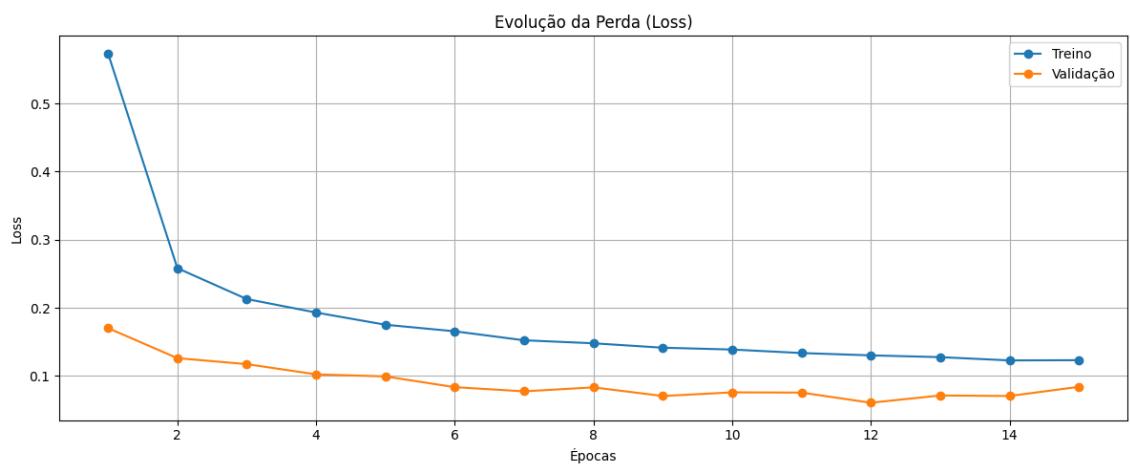


Figura 2. Evolução da Perda (Loss).

Tabela 2. Evolução da Perda (Loss).

Época	Perda (Treino)	Perda (validação)
1	0,5738	0,1702
2	0,2582	0,1261
3	0,2129	0,1173
4	0,1931	0,1023
5	0,1751	0,0992
6	0,1655	0,0834
7	0,1523	0,0773
8	0,1479	0,0831
9	0,1414	0,0704
10	0,1387	0,0758
11	0,1336	0,0755
12	0,1301	0,0606
13	0,1275	0,0712
14	0,1227	0,0705

15	0,1231	0,0839
----	---------------	---------------

Isso ocorre porque o conjunto de treino é artificialmente "dificultado" pelas rotações e distorções, enquanto a validação é realizada em imagens originais "limpas". Este comportamento indica uma forte regularização, confirmando que o modelo aprendeu as estruturas fundamentais dos dígitos e não apenas memorizou pixels.

4.2.5. Avaliação de Desempenho

Após o treinamento, o modelo foi submetido ao conjunto de testes de 10.000 imagens. O desempenho final registrou uma Acurácia Global de 97,31% e uma Perda de 0,0839.

4.2.5.1. Análise do Relatório de Métricas

A Tabela 3 detalha o desempenho por classe. O modelo demonstrou excelência nos dígitos 0 e 1, com F1-Scores próximos a 99%.

No entanto, observa-se uma queda de desempenho específica no dígito 4, que apresentou o menor *Recall* (92,97%) e no dígito 9, com a menor *Precision* (94,32%) e menor F1-Score (95,65%).

Tabela 3. Relatório de Classificação

Dígito	Precision	Recall	F1-Score	Support
0	98,38%	98,98%	98,68%	980
1	99,46%	98,15%	98,80%	1135
2	98,44%	97,87%	98,15%	1032
3	93,22%	99,41%	96,21%	1010
4	99,24%	92,97%	96,00%	982
5	98,07%	96,97%	97,52%	892
6	98,53%	97,70%	98,11%	958
7	97,93%	96,69%	97,31%	1028
8	95,94%	97,13%	96,53%	974
9	94,32%	97,03%	95,65%	1009
Acurácia	-	-	97,31%	10000

4.2.5.2. Análise da Matriz de Confusão

A análise granular dos erros (Figura 3) permitiu identificar as causas das métricas inferiores nos dígitos 4 e 9.

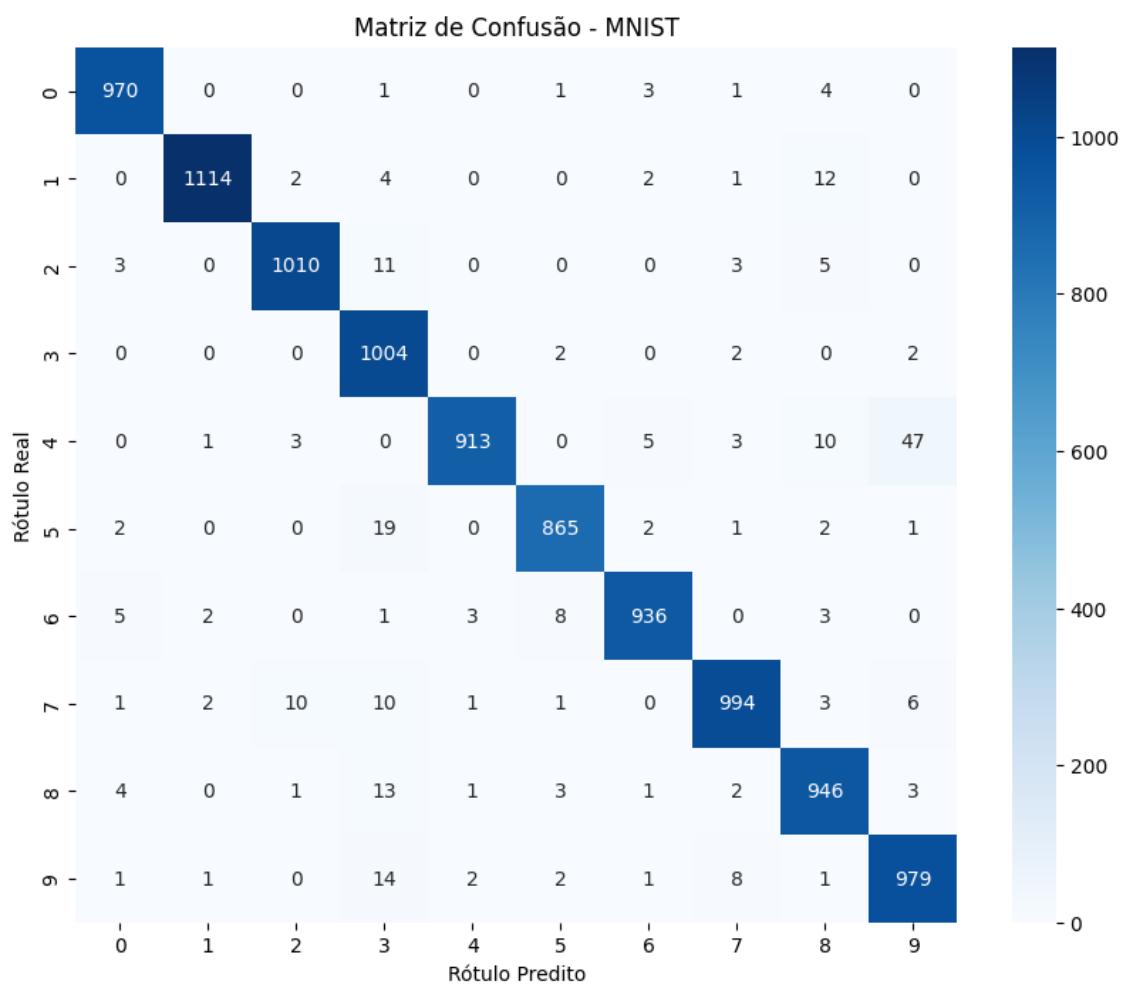


Figura 3. Matriz de Confusão.

1. O Colapso Visual entre 4 e 9

O erro mais significativo do modelo ocorreu na classificação do dígito 4. A matriz revela que o modelo classificou incorretamente o número 4 como sendo um 9 em 47 ocasiões.

2. O Dígito 3 como "Atrator de Erros"

O dígito 3 funcionou como um polo de confusão para diversos outros números curvilíneos:

- 5 => 3 (19 erros)
- 8 => 3 (13 erros)
- 9 => 3 (14 erros)

3. Conclusão da Análise

Apesar das confusões específicas concentradas nos pares (4,9) e (5,3), a média harmônica (F1) superior a 96% em quase todas as classes valida a robustez da arquitetura MLP aprimorada com *Data Augmentation*. O modelo demonstrou capacidade de generalização superior, evitando a memorização simples e aprendendo padrões morfológicos

complexos.

4.2.6. Teste Prático de Inferência (Prova de Conceito)

Para transcender a validação puramente estatística e demonstrar a aplicabilidade da solução em um cenário real de uso (*deploy*), realizou-se uma prova de conceito prática. Nesta etapa, o modelo foi submetido a dados completamente externos ao *dataset* MNIST, consistindo em dígitos desenhados manualmente em *software* de edição de imagens, simulando a entrada de dados por um usuário final.

4.2.6.1. Metodologia do Teste Prático.

O *pipeline* de inferência desenvolvido exigiu uma etapa crítica de pré-processamento para compatibilizar os dados do "mundo real" com o padrão da rede neural:

- Redimensionamento: Ajuste da imagem original para a matriz 28 x 28 pixels.
- Conversão Cromática: Transformação para escala de cinza (canal único).
- Inversão de Polaridade: Como os desenhos digitais padronizados são feitos com traço preto sobre fundo branco (o oposto do MNIST), aplicou-se uma inversão matemática para gerar o fundo preto com traço branco esperado pelo modelo.

4.2.6.2. Resultados Observados

Dos desenhos submetidos ao experimento prático, a grande maioria foi classificada corretamente pelo modelo, apresentando índices de confiança frequentemente superiores a 90%.

A Figura 4 e a Figura 5 ilustram um exemplo de sucesso, onde o sistema identificou corretamente um dígito desenhado fora de centro e com traço irregular.



Figura 4. Imagem a qual o modelo foi submetido.

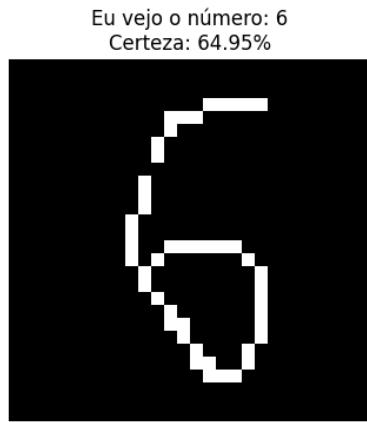


Figura 5. Resultado de inferência.

Este alto índice de acertos em imagens não padronizadas valida a hipótese de que a técnica de Aumento de Dados (*Data Augmentation*) foi eficaz. Ao treinar a rede com variações de rotação e translação, o modelo tornou-se robusto o suficiente para abstrair a essência do dígito, independentemente de pequenas variações de posição ou estilo de escrita inseridas pelo usuário no teste prático.

5. Conclusão

O Keras se consolidou como ferramenta essencial para o desenvolvimento de modelos de aprendizado profundo (*Deep Learning*). Desde sua criação, a evolução foi guiada por uma filosofia de simplicidade, acessibilidade e integração eficiente com diferentes frameworks e plataformas computacionais, o que contribuiu para sua ampla adoção tanto na academia quanto na indústria.

Ao longo do trabalho, foram apresentados seus princípios fundamentais, a estrutura modular e o papel das camadas (*layers*) na construção de redes neurais, bem como o ciclo completo de desenvolvimento de modelos no Keras — da definição da arquitetura ao treinamento, validação, avaliação, inferência e persistência — evidenciando um fluxo de trabalho intuitivo, robusto e consistente.

A implementação prática com o conjunto de dados MNIST reforçou a capacidade da ferramenta de simplificar tarefas complexas, permitindo que modelos relativamente simples alcancem bons resultados em problemas reais de classificação. Tal característica confirma que o Keras não apenas democratiza o acesso ao aprendizado profundo (*Deep Learning*), como também oferece recursos suficientes para aplicações robustas e escaláveis.

Dessa forma, conclui-se que o Keras permanece como um dos pilares no desenvolvimento de soluções baseadas em inteligência artificial, proporcionando equilíbrio adequado entre facilidade de uso, flexibilidade e desempenho. Seu papel segue relevante tanto para iniciantes quanto para profissionais que buscam produtividade e eficiência no desenvolvimento de modelos de *Deep Learning*.

6. Referências

- CHOLLET, François. **Deep Learning with Python**. 2. ed. Shelter Island: Manning Publications, 2021.
- CHOLLET, François. **Introducing Keras 1.0**. [S.l.]: Keras Blog, 30 mar. 2016. Disponível em: <https://blog.keras.io/introducing-keras-1.0.html>. Acesso em: 27 set. 2025.
- CHOLLET, François et al. **Keras**. [S.l.]: GitHub, 2015. Disponível em: <https://github.com/fchollet/keras>. Acesso em: 1 dez. 2025.
- DE CÁSSIA, Aline. **TensorFlow e Keras**: do princípio aos fundamentos essenciais - módulo 2. [S.l.]: Wikki Brasil, 13 jan. 2025. 1 vídeo. Disponível em: <https://www.youtube.com/watch?v=wYjYpVKFhmQ>. Acesso em: 1 dez. 2025.
- GÉRON, Aurélien. **Hands-On machine learning with Scikit-Learn, Keras, and TensorFlow**: concepts, tools, and techniques to build intelligent systems. 2. ed. Sebastopol: O'Reilly Media, 2019.
- GULLI, Antonio; PAL, Sujit. **Deep Learning with Keras**. Birmingham: Packt Publishing, 2017.
- KERAS. **About Keras**. [S.l.]: Keras, 2024. Disponível em: https://keras.io/getting_started/about/. Acesso em: 15 set. 2025.
- KINGMA, Diederik P.; BA, Jimmy. **Adam**: a method for stochastic optimization. [S.l.]: arXiv, 2014. Disponível em: <https://arxiv.org/abs/1412.6980>. Acesso em: 28 nov. 2025.
- LECUN, Yann; CORTES, Corinna; BURGES, C. J. C. **MNIST handwritten digit database**. [S.l.]: AT&T Labs, 2010. Disponível em: <http://yann.lecun.com/exdb/mnist/>. Acesso em: 4 out. 2025.
- TENSORFLOW. **Guia Keras**. [S.l.]: TensorFlow, 2025. Disponível em: <https://www.tensorflow.org/guide/keras?hl=pt-br>. Acesso em: 14 set. 2025.
- TENSORFLOW. **MNIST | TensorFlow Datasets**. [S.l.]: Google, 2025. Disponível em: <https://www.tensorflow.org/datasets/catalog/mnist?hl=pt-br>. Acesso em: 4 out. 2025.
- TENSORFLOW. **tf.keras.preprocessing.image.ImageDataGenerator**. [S.l.]: Google, 2024. Disponível em: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator. Acesso em: 30 nov. 2025.
- TENSORFLOW. **tf.random.set_seed**. [S.l.], [s.d.]. Disponível em: https://www.tensorflow.org/api_docs/python/tf/random/set_seed. Acesso em: 1 dez. 2025.