

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Enzo Lazzarini Amorim

**Estudo de um Modelo de Rede Neural
Convolutacional Aplicado ao Reconhecimento
Facial**

Uberlândia, Brasil

2026

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Enzo Lazzarini Amorim

**Estudo de um Modelo de Rede Neural Convolucional
Aplicado ao Reconhecimento Facial**

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Bruno Augusto Nassif Travencolo

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2026

Resumo

Este trabalho avalia uma arquitetura de Rede Neural Convolucional aplicada ao reconhecimento facial com base no modelo C1-C2-H. A configuração implementada mantém 16 e 36 mapas de características nas camadas convolucionais e compara camadas ocultas com 128, 256, 512 e 1024 neurônios no conjunto de dados Olivetti Faces. Para cada configuração, foram realizadas 50 execuções separadas, com novas partições estratificadas em 90% para treinamento e 10% para teste, treinamento por 20 épocas e lotes de 32 imagens. O desempenho foi descrito pela média e pelo desvio padrão das acurácias de teste. As configurações alcançaram, respectivamente, acurácias médias de 86,75%, 93,25%, 95,40% e 95,85%; a última apresentou desvio padrão de 2,77 pontos percentuais. A maior média permaneceu abaixo dos 97,5% de teste reportados pelo artigo de referência para a configuração equivalente. Diferenças de pré-processamento e de protocolo impedem, contudo, uma comparação experimental direta. Os resultados indicam que a configuração com 1024 neurônios obteve o melhor desempenho médio entre as alternativas avaliadas, mas as partições não pareadas, a seleção do modelo com os mesmos resultados usados em sua avaliação e as limitações do conjunto de dados restringem a generalização das conclusões.

Palavras-chave: Reconhecimento facial. Redes neurais convolucionais. Aprendizado profundo. Olivetti Faces.

Lista de ilustrações

Figura 1 – Hierarquia das camadas de convolução (IBM, 2020).	11
Figura 2 – Redução de dimensionalidade (An Optimistic Future - Futurology, 2020). (Figura traduzida pela ferramenta <i>Nano Banana 2</i>)	12
Figura 3 – Representação da camada de achatamento (MCCULLUM, 2025). (Figura traduzida pela ferramenta <i>Nano Banana 2</i>)	13
Figura 4 – Camada Totalmente Conectada (Educative, 2024). (Figura traduzida pela ferramenta <i>Nano Banana 2</i>)	14
Figura 5 – Diferença ao usar <i>Dropout</i> na camada oculta (LUCA, 2024). (Figura traduzida pela ferramenta <i>Nano Banana 2</i>)	14
Figura 6 – Gráfico da função <i>ReLU</i> (TEAM, 2025). (Figura traduzida pela ferramenta <i>Nano Banana 2</i>)	15
Figura 7 – Amostras do dataset Olivetti Faces, ilustrando variações de iluminação, expressão facial e detalhes como uso de óculos para quatro indivíduos distintos (SAMARIA; HARTER, 1994a).	20
Figura 8 – Acurácia de teste por tamanho da camada oculta; as colunas e a linha representam as médias, enquanto as hastes e a faixa representam um desvio padrão entre as execuções.	31
Figura 9 – Acurácia e perda de treinamento e validação ao longo das épocas.	32

Lista de tabelas

Tabela 1 – Arquitetura base do modelo <i>CNN</i> C1-C2-H (16-36-128).	21
---	----

Lista de abreviaturas e siglas

IA	Inteligência Artificial
CNN	Redes Neurais Convolucionais

Sumário

1	INTRODUÇÃO	8
1.1	Objetivos	8
1.1.1	Objetivo geral	8
1.1.2	Objetivos específicos	8
1.2	Organização do trabalho	9
2	FUNDAMENTAÇÃO TEÓRICA	10
2.1	Redes Neurais Convolucionais	10
2.1.1	Arquitetura de uma Rede Neural Convolucional	10
2.1.1.1	Camada de Convolução	10
2.1.1.2	Camada de Agrupamento	11
2.1.1.3	Camada de Achatamento	12
2.1.1.4	Camada Totalmente Conectada	12
2.1.1.5	Regularização com <i>Dropout</i>	13
2.1.2	Funções de Ativação	13
2.1.2.1	ReLU (<i>Rectified Linear Unit</i>)	14
2.1.2.2	Softmax	15
2.2	Linguagem de programação: Python	16
2.2.1	Gerenciamento de dependências com <i>uv</i>	16
2.3	Bibliotecas de aprendizado de máquina	17
2.3.1	TensorFlow	17
2.3.2	<i>Keras</i>	17
3	TRABALHOS RELACIONADOS	18
4	MÉTODO	19
4.1	Base de dados	19
4.2	Arquiteturas	19
4.3	Configuração do Ambiente de Desenvolvimento	21
4.3.1	Ferramentas de Software e Bibliotecas	22

4.3.2	Configuração de Hardware para Treinamento	22
4.4	Manipulação e Organização do Conjunto de Dados	22
4.4.1	<i>Download</i> e configuração do Conjunto de Dados	23
4.4.2	Pré-processamento e Carregamento dos Dados	23
4.5	Desenvolvimento do modelo	25
4.5.1	Implementação da Arquitetura CNN	25
4.5.2	Treinamento e Avaliação do Modelo	27
4.5.2.1	Metodologia de Treinamento	27
4.5.2.2	Monitoramento e Agregação de Métricas	28
5	RESULTADOS E DISCUSSÃO	30
5.1	Análise de Hiperparâmetros e Convergência	30
5.1.1	Histórico de Treinamento	31
5.2	Comparação com o artigo de referência	32
5.3	Discussão à luz dos métodos tradicionais	33
5.4	Discussão sobre a aplicabilidade em cenários reais	33
5.5	Limitações encontradas e trabalhos futuros	34
5.5.1	Limitações	34
5.5.2	Trabalhos Futuros	35
6	CONCLUSÃO	36
	REFERÊNCIAS	37

1 Introdução

O reconhecimento facial é uma tecnologia biométrica utilizada para identificar ou verificar indivíduos. Suas aplicações incluem o desbloqueio de *smartphones* e o controle de acesso a ambientes privados. No Brasil, a presença dessa tecnologia pode ser observada tanto na estimativa de mais de 47 milhões de pessoas sob vigilância por reconhecimento facial ([ABDALA, 2023](#)) quanto no uso do sistema pela Polícia Militar do Estado do Rio de Janeiro ([Secretaria de Estado de Polícia Militar do Rio de Janeiro, 2025](#)).

Entre as abordagens empregadas nesse domínio, destacam-se as Redes Neurais Convolucionais (*Convolutional Neural Networks* — CNNs), capazes de aprender características diretamente das imagens. Estudos anteriores apresentam resultados promissores para essa abordagem ([SILVA, 2023](#); [XIE; LI; SHI, 2019](#)). Este trabalho concentra-se na implementação e na avaliação da arquitetura C1-C2-H descrita por [Xie, Li e Shi \(2019\)](#), usando o conjunto de dados Olivetti Faces. Os trabalhos que fundamentam essa escolha são discutidos no Capítulo 3, enquanto os detalhes técnicos da implementação são apresentados no Capítulo 4.

1.1 Objetivos

1.1.1 Objetivo geral

Avaliar o desempenho de uma arquitetura CNN aplicada ao reconhecimento facial, com base no modelo de [Xie, Li e Shi \(2019\)](#), utilizando o conjunto de dados Olivetti Faces.

1.1.2 Objetivos específicos

- implementar a arquitetura C1-C2-H com as bibliotecas TensorFlow e Keras;
- pré-processar o conjunto de dados Olivetti Faces e executar o treinamento

conforme o protocolo descrito no método;

- comparar configurações com 128, 256, 512 e 1024 neurônios na camada oculta;
- descrever o desempenho por meio da acurácia média e do desvio padrão obtidos em execuções repetidas; e
- discutir as limitações do experimento e as condições necessárias para aplicações futuras.

1.2 Organização do trabalho

Além desta introdução, o Capítulo 2 apresenta a fundamentação teórica sobre CNNs, suas camadas e funções de ativação. O Capítulo 3 discute os trabalhos relacionados. O Capítulo 4 descreve o método, incluindo a base de dados, a arquitetura, o ambiente de desenvolvimento e o protocolo de treinamento. O Capítulo 5 apresenta e discute os resultados. Por fim, o Capítulo 6 retoma os objetivos, sintetiza as conclusões e propõe trabalhos futuros.

2 Fundamentação teórica

O reconhecimento facial é uma tecnologia que identifica ou verifica a identidade de indivíduos com base em características de suas faces. O processo começa pela captura de uma imagem e pela localização da face. Em seguida, extraem-se características que permitem associar a imagem a uma identidade. No modelo estudado, essa tarefa é realizada por uma Rede Neural Convolutacional ajustada com exemplos de treinamento.

2.1 Redes Neurais Convolucionais

As Redes Neurais Convolucionais são uma classe de redes neurais projetadas para processar dados organizados em grades, como imagens. Suas camadas aprendem representações hierárquicas: as primeiras tendem a responder a padrões locais simples, como bordas e texturas, enquanto as seguintes combinam essas informações em padrões mais complexos (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.1.1 Arquitetura de uma Rede Neural Convolutacional

Uma Rede Neural Convolutacional (*CNN*) é uma arquitetura de aprendizado profundo composta, em sua forma clássica, por três tipos fundamentais de camadas: a Camada de Convolução (*Convolutional Layer*), a Camada de Agrupamento (*Pooling*) e a Camada Totalmente Conectada (*Fully Connected Layer - FC*) (GOODFELLOW; BENGIO; COURVILLE, 2016). Essas camadas são tipicamente empilhadas em blocos para formar uma seção de extração de características, seguida por uma seção de classificação.

2.1.1.1 Camada de Convolução

Essa camada é responsável por aplicar filtros (*kernels*) à entrada, realizando operações de convolução para extrair características locais dos dados (bordas, cur-

vas, diagonais, etc). A ideia aqui é dividir a imagem em características específicas para classificar imagens, ou seja, extrair informações que facilitem a identificação do que se deseja aplicar, neste caso, faces. Além disso, pode-se aplicar mais de uma camada de convolução, a fim de hierarquizar as características necessárias para identificação. Seguindo o exemplo de (IBM, 2020), suponha que estamos buscando classificar bicicletas. Uma bicicleta é composta das mesmas partes, mas cada parte é diferente em cada bicicleta, e a combinação das partes é que compõe as classes das bicicletas. Assim, é possível estruturar a rede em diferentes níveis de profundidade para melhor identificar e extrair as características da imagem, como apontado na Figura 1.



Figura 1 – Hierarquia das camadas de convolução (IBM, 2020).

2.1.1.2 Camada de Agrupamento

A Camada de Agrupamento (*pooling*) agrega informações e torna a representação mais compacta, o que reduz o custo computacional e pode contribuir para diminuir o sobreajuste (*overfitting*), situação em que o modelo se ajusta excessivamente aos dados de treinamento e perde capacidade de generalização. Aqui, o modelo conduz a redução de dimensionalidade, varrendo filtros pelas saídas geradas na camada anterior (Convolução). Nessa camada, o *kernel* aplica uma função

de agregação aos valores, reduzindo as dimensões da saída e preservando as informações mais relevantes para a rede, como mostra a Figura 2.

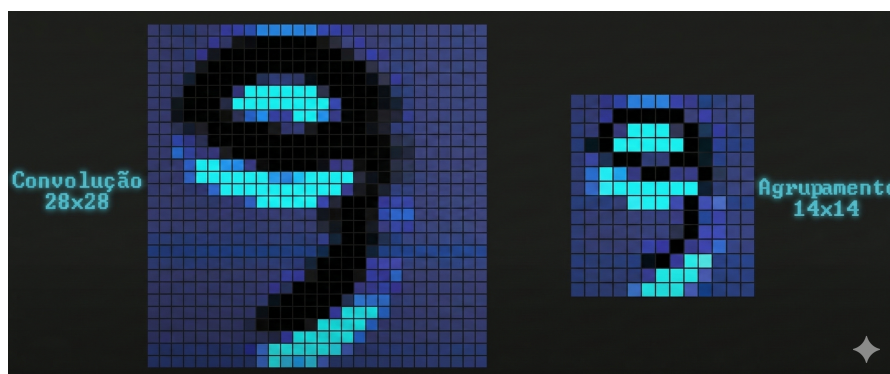


Figura 2 – Redução de dimensionalidade (An Optimistic Future - Futorology, 2020). (Figura traduzida pela ferramenta Nano Banana 2)

2.1.1.3 Camada de Achatamento

A saída da última camada convolucional ou de agrupamento é um tensor tridimensional, representando altura, largura e profundidade. Para conectar essa estrutura a uma camada densa tradicional, é necessário convertê-la em um vetor unidimensional. Essa transformação é realizada pela camada de *Flatten*, que achata o tensor e preserva os valores numéricos obtidos na etapa anterior (ZHAO et al., 2024), como representa a Figura 3.

2.1.1.4 Camada Totalmente Conectada

A camada densa, também conhecida como *fully connected layer*, é responsável pela fase de classificação propriamente dita. Cada neurônio dessa camada está conectado a todos os neurônios da camada anterior, conforme exemplificado na Figura 4, o que permite combinar de forma global as características extraídas anteriormente (ZHAO et al., 2024). Essa camada aprende a mapear o espaço de características para o espaço de saída, representando a relação entre os padrões detectados e as classes a serem previstas. Em problemas de classificação, a camada densa final contém um número de neurônios igual ao número de classes do modelo,

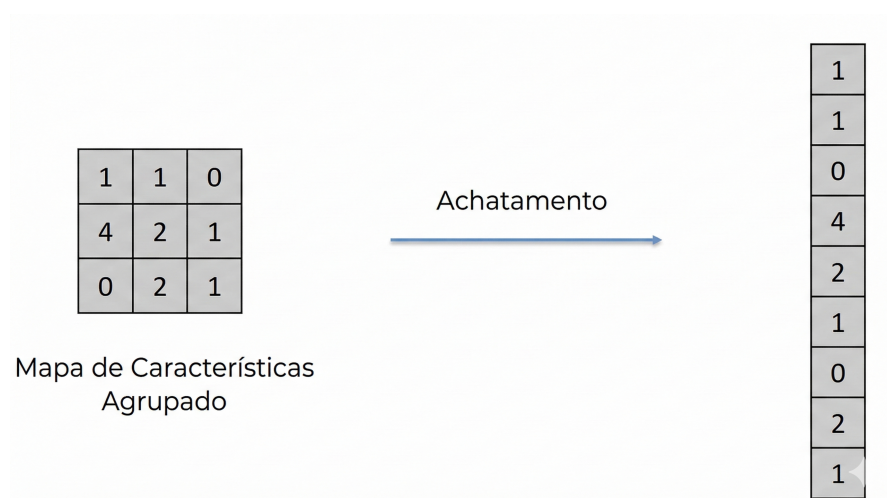


Figura 3 – Representação da camada de achatamento (MCCULLUM, 2025). (Figura traduzida pela ferramenta *Nano Banana 2*)

geralmente utilizando a função de ativação *softmax*, que converte os valores de saída em probabilidades normalizadas (ZHAO et al., 2024).

2.1.1.5 Regularização com *Dropout*

Para reduzir o sobreajuste, pode-se aplicar uma técnica de regularização. O *Dropout* consiste em desativar aleatoriamente uma fração dos neurônios durante o treinamento, reduzindo a coadaptação entre eles e favorecendo representações mais gerais (SRIVASTAVA et al., 2014). A Figura 5 mostra o funcionamento da camada.

2.1.2 Funções de Ativação

As funções de ativação são elementos fundamentais na arquitetura das redes neurais. Elas são aplicadas à saída de cada neurônio, ou de cada filtro no caso das camadas convolucionais, e têm como principal objetivo introduzir não-linearidade no modelo (GOODFELLOW; BENGIO; COURVILLE, 2016). Sem essa característica, mesmo uma rede com várias camadas se comportaria, do ponto de vista matemático, como uma única transformação linear. Isso limitaria sua capacidade de representar e aprender padrões complexos, como aqueles presentes em imagens,

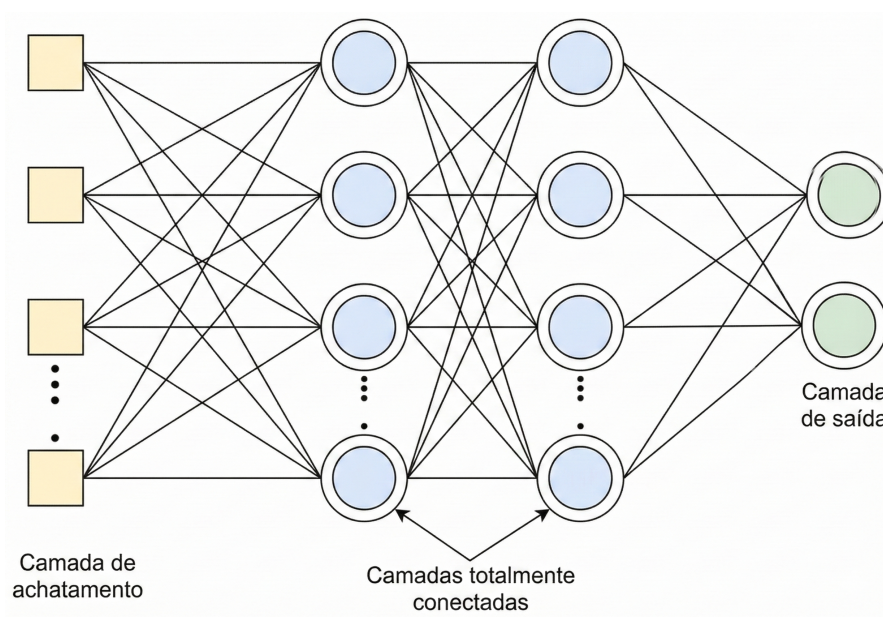


Figura 4 – Camada Totalmente Conectada ([Educative, 2024](#)). (Figura traduzida pela ferramenta *Nano Banana 2*)

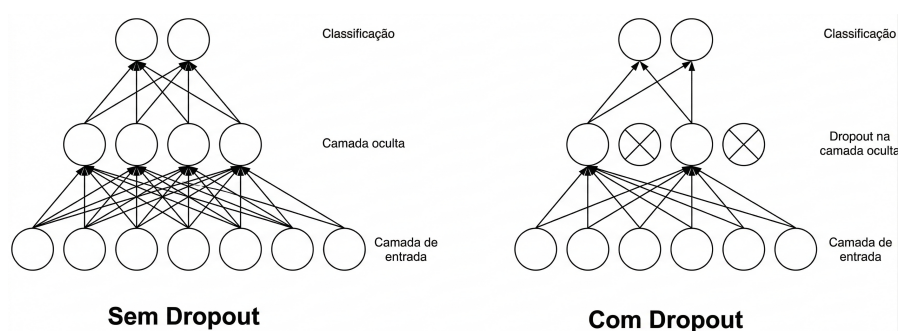


Figura 5 – Diferença ao usar *Dropout* na camada oculta ([LUCA, 2024](#)). (Figura traduzida pela ferramenta *Nano Banana 2*)

sons ou textos. Entre as funções de ativação mais utilizadas em redes convolucionais estão a ReLU e a Softmax.

2.1.2.1 ReLU (*Rectified Linear Unit*)

A Unidade Linear Retificada, ou *ReLU*, é uma das funções de ativação mais populares em redes neurais modernas, sendo utilizada tanto em camadas convolu-

cionais quanto em camadas densas (NAIR; HINTON, 2010). Sua popularidade se deve à combinação entre simplicidade computacional e eficiência durante o treinamento. A função é definida como $f(x) = \max(0, x)$, o que significa que retorna o próprio valor de entrada quando ele é positivo e zero quando é negativo, com gráfico apontado na Figura 6. Essa operação ajuda a reduzir o desaparecimento do gradiente (*vanishing gradient*) (GLOROT; BENGIO, 2010), fenômeno em que os gradientes se tornam muito pequenos durante a retropropagação e dificultam o aprendizado nas camadas iniciais da rede. Como resultado, redes mais profundas podem ser treinadas com maior estabilidade e desempenho.

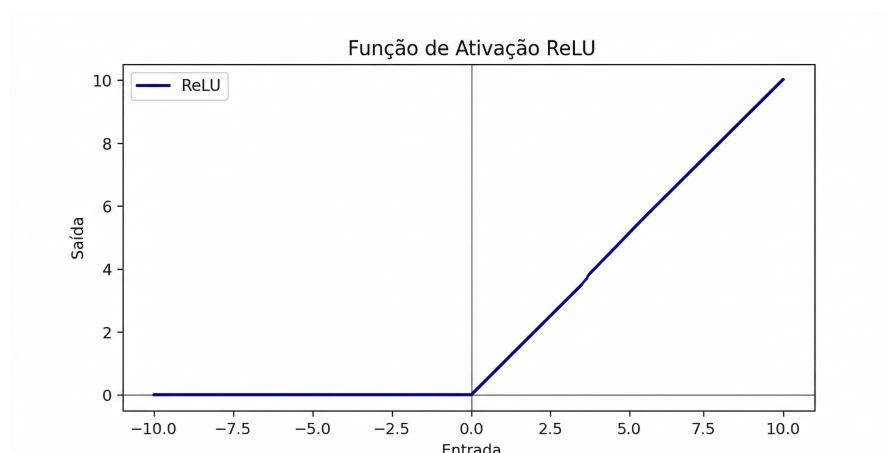


Figura 6 – Gráfico da função *ReLU* (TEAM, 2025). (Figura traduzida pela ferramenta *Nano Banana 2*)

2.1.2.2 Softmax

A função *Softmax* é amplamente utilizada na camada de saída de redes neurais voltadas para classificação multiclasse (GOODFELLOW; BENGIO; COURVILLE, 2016). Seu papel é transformar o vetor de valores brutos, conhecidos como logits, produzido pela última camada densa em uma distribuição de probabilidades. Em um problema com N classes distintas, a *softmax* apresenta duas propriedades fundamentais (GOODFELLOW; BENGIO; COURVILLE, 2016):

- Cada saída é normalizada para um valor entre 0 e 1.

- A soma de todas as N saídas é igual a 1.

Dessa forma, a saída do neurônio i pode ser interpretada como a probabilidade de a entrada pertencer à classe i , e a classe com maior probabilidade é escolhida como a predição final do modelo.

2.2 Linguagem de programação: Python

A escolha da linguagem de programação influencia diretamente a arquitetura de software, pois define os paradigmas, ferramentas e bibliotecas disponíveis para o desenvolvimento.

Para este trabalho, foi adotado o *Python* ([Python Software Foundation, s.d.](#)), linguagem interpretada e de alto nível cuja sintaxe prioriza a legibilidade. Sua biblioteca padrão e o suporte a módulos e pacotes favorecem a organização e a reutilização do código.

No contexto da Inteligência Artificial (IA), o *Python* destaca-se por sua ampla adoção e pelo suporte de bibliotecas consolidadas, como *TensorFlow*, *Keras* e *Scikit-learn* ([ADETIBA et al., 2021](#)). Essas ferramentas fornecem suporte para tarefas como aprendizado de máquina, redes neurais e processamento de linguagem natural, reforçando a escolha da linguagem para o desenvolvimento deste projeto.

Outro ponto importante é sua natureza multi-paradigma. O *Python* oferece suporte à programação orientada a objetos, procedural e funcional, conferindo flexibilidade no desenvolvimento de diferentes tipos de aplicações.

2.2.1 Gerenciamento de dependências com *uv*

O *uv*¹ foi utilizado para criar o ambiente virtual e centralizar a instalação das dependências do projeto.

¹ <<https://docs.astral.sh/uv/>>

2.3 Bibliotecas de aprendizado de máquina

Para a construção e o treinamento da CNN, foram utilizados o *TensorFlow* e o *Keras*, descritos a seguir.

2.3.1 TensorFlow

O *TensorFlow* ([ABADI et al., 2015](#)) é um *framework* de aprendizado de máquina de código aberto desenvolvido pela Google para facilitar a construção e treinamento de modelos de aprendizado de máquina e aprendizado profundo. O *TensorFlow* oferece diversas ferramentas e bibliotecas que facilitam a implementação de algoritmos de aprendizado de máquina.

2.3.2 Keras

O *Keras* ([CHOLLET et al., 2015](#)) é uma API de aprendizado profundo escrita em *Python*. Sua API sequencial, utilizada neste trabalho, permite organizar as camadas do modelo como uma pilha linear.

3 Trabalhos Relacionados

A monografia de [Silva \(2023\)](#) compara LBPH, *Eigenface*, *Fisherface* e *Facenet* na autenticação de usuários. O LBPH descreve texturas locais; o *Eigenface* reduz a dimensionalidade por Análise de Componentes Principais; o *Fisherface* combina essa redução com Análise Discriminante Linear; e o *Facenet* utiliza uma CNN. Entre os métodos avaliados, o autor relata os melhores resultados de acurácia para o *Facenet*.

O artigo de [Xie, Li e Shi \(2019\)](#), adotado como referência neste TCC, descreve arquiteturas convolucionais para extrair e classificar características faciais no conjunto de dados Olivetti Faces. No experimento que mantém 16 e 36 mapas de características e varia a camada oculta, os autores registram 97,5% de acurácia de teste nas quatro configurações; os 99,69% da configuração 16-36-1024 correspondem ao conjunto de treinamento. Em um segundo experimento, que também amplia os mapas de características, a configuração 36-76-1024 atinge 100% no conjunto de teste ([XIE; LI; SHI, 2019](#)).

O estudo sistemático de [Nemavhola, Chibaya e Viriri \(2025\)](#), por sua vez, revisa bases de dados, arquiteturas e métricas empregadas em reconhecimento facial. Sua abrangência permite identificar limitações dos conjuntos de dados e desafios recorrentes na comparação entre modelos propostos por diferentes estudos.

Diferentemente de [Silva \(2023\)](#), o presente trabalho não compara famílias distintas de algoritmos. Em relação a [Xie, Li e Shi \(2019\)](#), sua contribuição consiste em implementar a configuração com 16 e 36 mapas de características e comparar camadas ocultas com 128, 256, 512 e 1024 neurônios por meio de execuções repetidas no Olivetti Faces. Enquanto [Nemavhola, Chibaya e Viriri \(2025\)](#) realiza uma revisão ampla da literatura, este TCC apresenta uma análise experimental restrita a essa configuração e a esse conjunto de dados.

4 Método

A metodologia deste trabalho baseia-se na reprodução parcial e na comparação com o artigo de referência (XIE; LI; SHI, 2019). Foi implementada a etapa do estudo que mantém 16 e 36 mapas de características nas camadas convolucionais e varia a quantidade de neurônios da camada oculta. O protocolo de avaliação, porém, utiliza partições repetidas e, por isso, não constitui uma replicação exata do experimento original.

4.1 Base de dados

O conjunto de dados Olivetti Faces, também conhecido como ORL Database of Faces, foi disponibilizado pelo AT&T Laboratories Cambridge e reúne imagens coletadas entre 1992 e 1994 (SAMARIA; HARTER, 1994b). Ele contém 400 imagens de 40 indivíduos, com dez fotografias por pessoa. As imagens apresentam variações de iluminação, expressão facial, abertura dos olhos e uso de óculos. Algumas amostras são apresentadas na Figura 7.

Todas as imagens foram capturadas sob condições controladas, com os indivíduos em posição frontal contra um fundo escuro e uniforme. Enquanto as imagens originais tinham resolução de 92×112 pixels, os arquivos utilizados neste trabalho as representam com 64×64 pixels. Originalmente, cada pixel é representado por um inteiro de 8 bits, correspondente a 256 níveis de cinza. Na versão utilizada, os valores foram normalizados para o intervalo $[0, 1]$. Essa normalização reduz a escala dos valores de entrada e favorece a estabilidade numérica durante a otimização dos pesos da rede.

4.2 Arquiteturas

A notação C1-C2-H resume as configurações avaliadas. C1 representa o número de mapas de características (*feature maps*) produzidos pela primeira camada



Figura 7 – Amostras do dataset Olivetti Faces, ilustrando variações de iluminação, expressão facial e detalhes como uso de óculos para quatro indivíduos distintos (SAMARIA; HARTER, 1994a).

convolucional. Cada mapa corresponde à resposta de um filtro e tende a representar padrões locais, como bordas, cantos e texturas. C2 representa a quantidade de mapas produzidos pela segunda camada, que combina os padrões anteriores em características mais complexas. Por fim, H indica o número de neurônios da camada oculta responsável pela classificação. O modelo base 16-36-128, apresentado na Tabela 1, foi comparado com variantes que alteram o valor de H.

A primeira etapa de extração de características consiste em um bloco convolucional. Inicialmente, uma camada de convolução (Convolução 2D) é criada com 16 filtros de tamanho 3x3. A função de ativação **ReLU** é utilizada, e um preenchimento é aplicado para garantir que os 16 mapas de características gerados mantenham as dimensões originais de 57x47. Em seguida, uma camada de agrupamento (Agrupamento Máximo 2D) realiza a primeira operação de subamostragem (*downsampling*), reduzindo pela metade as dimensões espaciais dos mapas de características, sem alterar sua quantidade. O segundo bloco aprofunda a extração de características. Uma nova camada de convolução eleva a complexidade do mo-

Tabela 1 – Arquitetura base do modelo *CNN* C1-C2-H (16-36-128).

Tipo de Camada	Formato da Saída	Nº de Parâmetros
Convolução 2D	(57, 47, 16)	160
Agrup. Máximo 2D	(28, 23, 16)	0
Convolução 2D	(28, 23, 36)	5220
Agrup. Máximo 2D	(14, 11, 36)	0
Achatamento	(5544)	0
Dropout	(5544)	0
Densa	(128)	709760
Dropout	(128)	0
Densa	(40)	5160

delo ao aplicar 36 filtros, transformando os 16 mapas de entrada em 36 novos mapas. A essa camada segue-se uma segunda camada de agrupamento, que novamente reduz as dimensões espaciais pela metade. Na seção de classificação, uma camada de achatamento (*Flatten*) converte os mapas de características em um vetor unidimensional. Em seguida, uma camada de *Dropout* desativa aleatoriamente parte dos valores para reduzir o sobreajuste. Uma camada oculta (Densa) com ativação ReLU processa o vetor, seguida por outra camada de *Dropout*. Por fim, a camada de saída, também do tipo Densa, utiliza a função de ativação **softmax** para produzir as probabilidades das 40 identidades.

4.3 Configuração do Ambiente de Desenvolvimento

Para configurações de dependências, o gerenciador de pacotes *uv* foi utilizado, mantendo as dependências centralizadas com fácil gerenciamento, reprodutibilidade e isolamento do ambiente de desenvolvimento. Assim, para a execução dos experimentos, foi configurado um ambiente de desenvolvimento que combinasse ferramentas de software amplamente utilizadas na área de aprendizado profundo e um hardware adequado ao treinamento de redes neurais convolucionais. A seguir são descritos os principais componentes empregados.

4.3.1 Ferramentas de Software e Bibliotecas

A implementação dos modelos foi realizada em *Python* ([Python Software Foundation](#), s.d.), linguagem consolidada em ciência de dados e apoiada por um amplo ecossistema de bibliotecas. O *TensorFlow* foi utilizado como base para a construção das redes neurais, oferecendo suporte para cálculos em larga escala, enquanto a API de alto nível *Keras* foi empregada para facilitar a definição e o treinamento das arquiteturas. Para a divisão dos dados, a estratificação das amostras e o cálculo das métricas de desempenho, utilizou-se a biblioteca *scikit-learn* ([PEDREGOSA et al., 2011](#)). O *tqdm* foi empregado para o monitoramento das execuções, exibindo barras de progresso durante os ciclos de treinamento. No pré-processamento das imagens, a biblioteca *scikit-image* ([WALT et al., 2014](#)) foi utilizada por meio do módulo `resize`, responsável por redimensionar cada imagem do *dataset* para o formato esperado pelo modelo. Por fim, o *Matplotlib* foi empregado na geração dos gráficos de desempenho dos modelos.

4.3.2 Configuração de Hardware para Treinamento

Apesar da aceleração reconhecida proporcionada por Unidades de Processamento Gráfico (GPUs) no treinamento de redes neurais convolucionais, os experimentos foram conduzidos em um processador **Intel(R) Core(TM) i7-9700KF CPU @ 3.60GHz**. Embora o ambiente contasse com uma GPU *NVIDIA GeForce RTX 2070* com suporte à plataforma *CUDA*, a configuração para delegação das operações ao *TensorFlow* não foi ativada. Dessa forma, todos os cálculos foram executados na CPU, sem otimização do tempo de treinamento por GPU. O ambiente também contava com 32 GB de memória RAM a 3200 MHz.

4.4 Manipulação e Organização do Conjunto de Dados

A primeira etapa fundamental para o treinamento do modelo consiste na aquisição e organização do conjunto de dados. Neste trabalho, foi utilizado o *dataset Olivetti faces*, obtido diretamente da plataforma Kaggle por meio da biblioteca *kagglehub*. O processo foi automatizado por um script em Python, garantindo a

reprodutibilidade e a correta estruturação dos arquivos para as etapas subsequentes.

4.4.1 *Download* e configuração do Conjunto de Dados

O Código 4.1 inicia com o *download* do dataset para um diretório temporário gerenciado pela própria biblioteca. Em seguida, uma estrutura de diretórios local, definida pela variável `DATASET_DIR`, é criada para armazenar permanentemente os arquivos de imagem. A função `os.makedirs` com o parâmetro `exist_ok=True` é utilizada para evitar erros caso o diretório de destino já exista em execuções anteriores. O núcleo da manipulação é um laço de iteração que percorre todos os itens no diretório de *download* temporário. Para cada item, o script verifica se é um arquivo (e não um subdiretório) utilizando o método `item.is_file()`. Sendo um arquivo, ele é copiado para o diretório de destino `DATASET_DIR` através da função `shutil.copy`. Essa abordagem assegura que apenas as imagens sejam transferidas, resultando em uma pasta local organizada e pronta para as etapas seguintes.

```
1 DATASET_DIR = "olivetti_dataset"
2
3 path = Path(kagglehub.dataset_download("imrandude/olivetti"
4 ))
5 os.makedirs(DATASET_DIR, exist_ok=True)
6
7 for item in path.iterdir():
8     target = os.path.join(DATASET_DIR, item.name)
9     if item.is_file():
10         shutil.copy(item, target)
```

Código 4.1 – Script para *download* e organização local do dataset Olivetti.

4.4.2 Pré-processamento e Carregamento dos Dados

Após a organização dos arquivos de imagem, a próxima etapa consiste em carregá-los em memória e pré-processá-los para que se adequem ao formato esperado pela arquitetura da rede neural convolucional. Este processo é encapsulado

pela função `load_data`, detalhada no Código 4.2, que realiza quatro tarefas essenciais. Primeiramente, a função carrega os dados das imagens e seus respectivos rótulos a partir dos arquivos `.npz` utilizando a biblioteca **NumPy**. Em seguida, cada imagem, originalmente com dimensões de 64x64 pixels, é redimensionada para 57x47 pixels através da função `resize` da biblioteca **Scikit-image**. Este redimensionamento serve para alinhar as dimensões da entrada com a camada de entrada do modelo. A terceira tarefa é a conformação da dimensionalidade do array de imagens. Uma dimensão de canal é adicionada ao final de cada imagem, transformando o formato de (400, 57, 47) para (400, 57, 47, 1). Essa dimensão extra é um requisito padrão para tensores de imagem em frameworks como o TensorFlow/Keras, indicando que se trata de imagens em escala de cinza (um único canal). Finalmente, os rótulos numéricos (de 0 a 39) são convertidos para o formato *one-hot encoding* usando a função `to_categorical`. Isso transforma cada rótulo inteiro em um vetor binário de 40 posições, onde apenas o índice correspondente à classe da imagem é marcado com 1. Este formato é o padrão para treinar modelos de classificação com a função de perda de entropia cruzada categórica (*categorical cross-entropy*). Ao final, a função retorna os dados de imagem e os rótulos, ambos prontos para serem consumidos pelo modelo.

```
1 def load_data(dataset_name):
2     data = np.load(f"{DATASET_DIR}/{dataset_name}_faces.npz")
3     target = np.load(f"{DATASET_DIR}/{dataset_name}_faces_target.npz")
4
5     data_resized = np.array([resize(img, (57, 47),
6     anti_aliasing=True) for img in data])
7     data_resized = data_resized[..., np.newaxis].astype("float32")
8
9     return data_resized, to_categorical(target, num_classes=40)
```

Código 4.2 – Função para carregar e pré-processar o dataset.

Essa função é utilizada para separar as imagens do *dataset* de suas respectivas classes, possibilitando o treinamento do modelo e o uso dessas classes como

referência na melhoria das características extraídas pela rede.

4.5 Desenvolvimento do modelo

Com os dados devidamente pré-processados e formatados, a etapa seguinte consiste na definição e implementação da arquitetura da rede neural convolucional responsável por realizar a tarefa de reconhecimento facial. Nesta fase, são especificadas as camadas que compõem o modelo, suas funções e parâmetros, bem como o método de compilação que orienta o processo de aprendizado. A utilização das bibliotecas **TensorFlow** e **Keras** permite uma construção modular e eficiente da rede, facilitando tanto a experimentação de diferentes configurações quanto a integração com rotinas de treinamento e avaliação. Após a definição da arquitetura, o modelo é treinado e suas métricas de desempenho são monitoradas ao longo das épocas, permitindo a análise da evolução do aprendizado e da capacidade de generalização da rede.

4.5.1 Implementação da Arquitetura CNN

A implementação da rede neural convolucional foi realizada utilizando as bibliotecas **TensorFlow** e **Keras**, que oferecem uma interface de alto nível para construção, treinamento e avaliação de modelos de aprendizado profundo. O modelo adotado segue uma estrutura inspirada na arquitetura proposta por [Xie, Li e Shi \(2019\)](#), projetada para reconhecimento facial com imagens em escala de cinza de dimensões reduzidas (57x47 pixels). A definição do modelo está encapsulada na função `build_model`, apresentada no Código 4.3. Essa função utiliza a API sequencial do Keras, que permite empilhar camadas de forma linear, facilitando a implementação e leitura da arquitetura. A rede é composta por duas camadas convolucionais seguidas de camadas de *pooling*, responsáveis por extrair e condensar as características espaciais das imagens. Cada camada convolucional aplica filtros 3x3 com ativação *ReLU* e preenchimento (*padding*) `same`. Esse preenchimento adiciona valores às bordas da entrada para preservar suas dimensões espaciais antes da operação de *pooling*. O uso da ativação *ReLU* favorece o treinamento e reduz o problema do desaparecimento do gradiente, descrito na fundamentação teórica.

Após as etapas convolucionais, o tensor tridimensional é achatado (*flattened*) em um vetor unidimensional, possibilitando a conexão com camadas densas totalmente conectadas. São adicionadas camadas de *dropout* com taxa de 50% após o achatamento e após a camada densa intermediária, como forma de regularização para evitar o sobreajuste (*overfitting*). A camada totalmente conectada (*Dense*) intermediária utiliza ativação *ReLU* e serve como extratora de padrões de alto nível. Já a camada de saída contém 40 neurônios, correspondentes às classes do conjunto de dados, e emprega ativação *softmax*, adequada para problemas de classificação multiclasse, pois normaliza as saídas em probabilidades somando 1.

Por fim, o modelo é compilado com o otimizador *Adam* e a função de perda *categorical cross-entropy*, métricas padrão para tarefas de classificação supervisionada. O Código 4.3 ilustra a implementação completa dessa arquitetura.

```
1 def build_model(self, input_shape=(57, 47, 1)):
2     model = Sequential()
3
4     # Input layer
5     model.add(Input(shape=input_shape))
6
7     # Convolution layer 1
8     model.add(Conv2D(self.convolution_11, (3, 3),
9                       activation="relu", padding="same"))
10    model.add(MaxPooling2D(pool_size=(2, 2)))
11
12    # Convolution layer 2
13    model.add(Conv2D(self.convolution_12, (3, 3),
14                      activation="relu", padding="same"))
15    model.add(MaxPooling2D(pool_size=(2, 2)))
16
17    # Flatten + Dropout
18    model.add(Flatten()) # Output: 14*11*36 = 5544
19    model.add(Dropout(0.5))
20
21    # Fully connected + Dropout
22    model.add(Dense(self.hidden_1, activation="relu"))
23    model.add(Dropout(0.5))
24
```

```
25     # Output layer
26     model.add(Dense(40, activation="softmax"))
27     model.compile(optimizer="adam",
28                   loss="categorical_crossentropy",
29                   metrics=["accuracy"])
30
31     return model
```

Código 4.3 – Função para construção do modelo convolucional.

A estrutura resultante recebe imagens de 57×47 pixels em escala de cinza e produz probabilidades para as 40 classes do conjunto de dados.

4.5.2 Treinamento e Avaliação do Modelo

O treinamento da CNN é executado por um *script* que repete a partição dos dados, o ajuste do modelo e o cálculo das métricas. A função `train_model` reúne essas etapas e agrega as acurácias observadas.

O *script* principal itera sobre um conjunto predefinido de hiperparâmetros, como o número de neurônios na camada densa (`hidden_neurons_list`), permitindo a comparação de diferentes configurações da arquitetura.

4.5.2.1 Metodologia de Treinamento

Para cada configuração da camada oculta, foram realizadas 50 execuções separadas, cada uma com uma nova partição dos dados e uma nova instância do modelo. Em todas essas execuções, o treinamento utilizou 20 épocas e lotes de 32 imagens. Nessa etapa comparativa, não foram definidas sementes globais para a inicialização dos pesos nem o parâmetro `random_state` nas partições; portanto, as divisões variaram entre as execuções e não foram pareadas entre as configurações.

Em cada execução, o processo segue os seguintes passos:

1. **Particionamento dos Dados:** O *dataset* completo, carregado pela função `load_data`, é novamente dividido em conjuntos de treinamento (90%) e teste (10%) com a biblioteca **Scikit-learn**. A estratificação (`stratify=y`) mantém a proporção das classes em ambos os conjuntos.


```
1 X_train, X_test, y_train, y_test = train_test_split(  
2     X, y, test_size=0.1, stratify=y  
3 )
```

Código 4.4 – Particionamento dinâmico dos dados a cada *run*.

2. **Instanciação do Modelo:** Uma nova instância do modelo (`XieModel`) é criada e compilada para que cada execução comece com pesos reinicializados. O *script* utiliza uma função `lambda` para passar o construtor do modelo para `train_model`.

```
1 model = model_fn()
```

Código 4.5 – Criação de nova instância do modelo via `lambda`.

3. **Treinamento (Fit):** O modelo recém-criado é treinado com o método `model.fit` sobre os dados de treinamento (`X_train`, `y_train`) durante 20 épocas, com tamanho de lote igual a 32.

4.5.2.2 Monitoramento e Agregação de Métricas

O desempenho é monitorado durante o treinamento e agregado ao final das execuções:

1. Monitoramento por Época (Validação):

Durante a execução do `model.fit`, o **Keras** reserva 10% dos dados de treinamento como conjunto de validação interna. As métricas de perda (*loss*) e acurácia (*accuracy*) nesse conjunto são calculadas ao final de cada época, permitindo observar a convergência e indícios de sobreajuste.

```
1 model.fit(  
2     X_train,  
3     y_train,  
4     epochs=epochs,  
5     batch_size=batch_size,  
6     validation_split=0.1,  
7     verbose=0,  
8 )
```

Código 4.6 – Uso de `validation_split` no `model.fit`.

2. **Avaliação por Execução (Teste):** Após o término das épocas de uma execução, o modelo é avaliado no conjunto de teste (`X_test`, `y_test`), que não foi usado no ajuste dos pesos daquela execução. A acurácia observada é registrada.

```
1 _, acc = model.evaluate(X_test, y_test, verbose=0)
2 accuracies.append(acc)
```

Código 4.7 – Avaliação no conjunto de teste e registro da métrica.

3. **Agregação Final (Média e Desvio Padrão):** Ao final das execuções de uma configuração, as acurácias armazenadas em `accuracies` são resumidas pela média e pelo desvio padrão. Essas medidas descrevem, respectivamente, o desempenho médio observado e sua variabilidade entre as repetições; não constituem, por si sós, um teste de significância estatística.

```
1 mean_acc = np.mean(accuracies)
2 std_acc = np.std(accuracies)
3 log.status(f"Mean Accuracy: {mean_acc*100:.2f}% \pm {
    std_acc*100:.2f}%")
```

Código 4.8 – Cálculo da média e desvio padrão das acurácias.

Esse procedimento aplica a mesma estratégia de partição repetida às quatro configurações, mas não utiliza partições pareadas. A Seção 5.2 confronta esses resultados com os do artigo de referência, considerando as diferenças entre os protocolos experimentais.

5 Resultados e Discussão

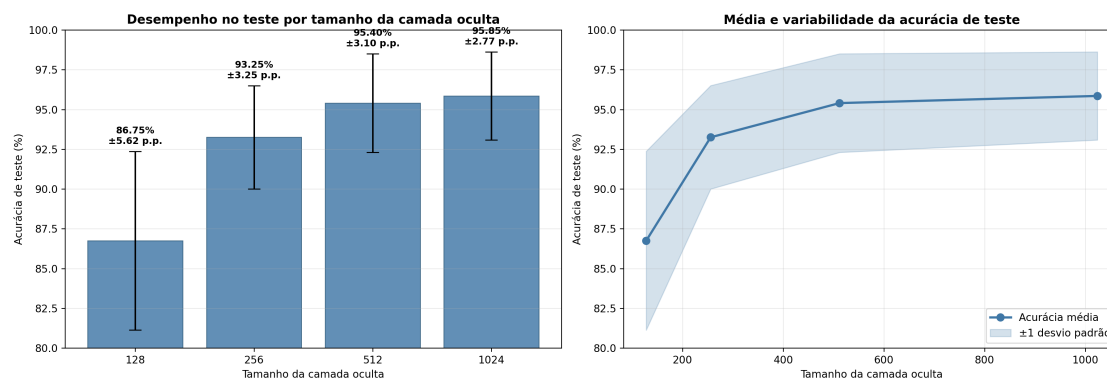
Este capítulo apresenta e discute as acurácias obtidas pelas configurações avaliadas, a variação observada entre as execuções e o histórico de treinamento. Em seguida, relaciona os resultados ao artigo de referência (XIE; LI; SHI, 2019) e à literatura discutida, delimitando as conclusões permitidas pelo protocolo experimental.

5.1 Análise de Hiperparâmetros e Convergência

Antes da definição do modelo final, foi realizado um estudo comparativo variando o número de neurônios na camada oculta (*hidden layer*), conforme metodologia descrita na Seção 4.5.2.1. O objetivo foi identificar a configuração que maximizasse a acurácia média e minimizasse o desvio padrão entre as múltiplas execuções. Os valores agregados correspondem às acurácias calculadas no conjunto de teste ao final de cada execução.

A Figura 8 ilustra o desempenho de teste do modelo com 128, 256, 512 e 1024 neurônios. As acurácias médias foram, respectivamente, 86,75%, 93,25%, 95,40% e 95,85%, com desvios padrão de 5,62, 3,25, 3,10 e 2,77 pontos percentuais. Observa-se uma tendência de aumento da acurácia média entre as configurações avaliadas, com a maior média na configuração de 1024 neurônios.

Figura 8 – Acurácia de teste por tamanho da camada oculta; as colunas e a linha representam as médias, enquanto as hastes e a faixa representam um desvio padrão entre as execuções.



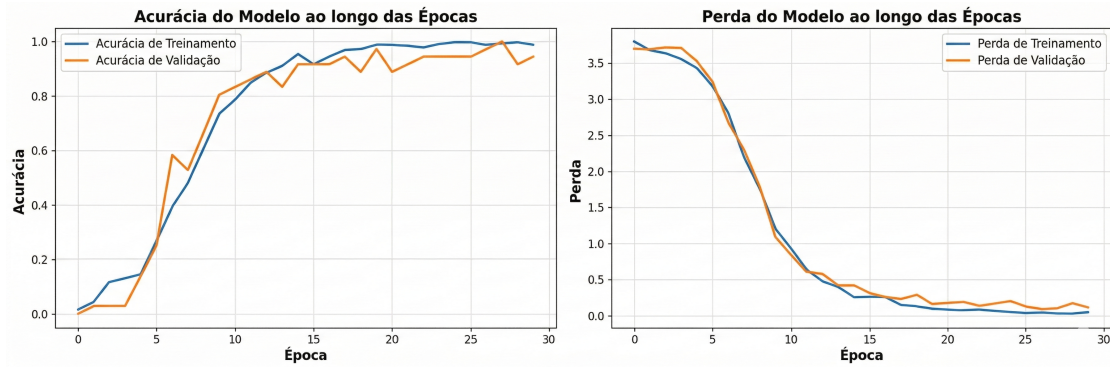
Fonte: Elaborado pelo autor.

Com base nesses resultados, a arquitetura com 1024 neurônios na camada oculta foi a configuração de melhor acurácia média entre as avaliadas e foi adotada nas análises subsequentes.

5.1.1 Histórico de Treinamento

O comportamento do modelo durante o processo de aprendizagem pode ser visualizado na Figura 9. Após a seleção da configuração com 1024 neurônios, foi realizada uma execução separada com a semente 42 na partição de teste, 30 épocas e lotes de 32 imagens. Os gráficos mostram a evolução da acurácia e da função de perda (*loss*) no treinamento e na validação dessa execução.

Figura 9 – Acurácia e perda de treinamento e validação ao longo das épocas.



Fonte: Elaborado pelo autor.

Nessa execução, a acurácia de validação aproxima-se de 90% por volta da 15ª época e se estabiliza nas épocas finais. As curvas de validação e treinamento seguem tendências semelhantes, sem divergência acentuada. Essa observação descreve apenas a execução representada e não substitui uma análise agregada do sobreajuste entre todas as repetições.

5.2 Comparação com o artigo de referência

Os dois estudos utilizam as 400 imagens do Olivetti Faces, entradas de 57×47 pixels e a configuração com 16 e 36 mapas de características para avaliar camadas ocultas de 128, 256, 512 e 1024 neurônios. Os protocolos de avaliação, entretanto, são distintos. Xie, Li e Shi (2019) separam 320 imagens para treinamento, 40 para validação e 40 para teste, sem descrever partições repetidas. Neste trabalho, cada execução reserva 10% das imagens para teste e 10% das 360 imagens restantes para validação, resultando em 324 imagens efetivamente usadas no ajuste, 36 na validação e 40 no teste. As acurácias de teste são então agregadas entre as execuções. O pré-processamento também difere: o artigo de referência obtém as faces de 57×47 pixels a partir de uma imagem composta, enquanto este trabalho carrega arquivos de 64×64 pixels e os redimensiona com a biblioteca *scikit-image*.

Na Tabela 3 do artigo de referência, as quatro configurações com 16 e 36

mapas alcançam 97,5% no conjunto de teste. O valor de 99,69% associado à configuração 16-36-1024 refere-se ao conjunto de treinamento (XIE; LI; SHI, 2019). Neste TCC, as médias de teste para 128, 256, 512 e 1024 neurônios foram, respectivamente, 86,75%, 93,25%, 95,40% e 95,85%. Portanto, a maior média obtida permaneceu abaixo dos 97,5% reportados no teste pelo artigo.

O resultado de 100% no teste apresentado por Xie, Li e Shi (2019) pertence à configuração 36-76-1024, obtida em uma segunda etapa que também altera a quantidade de mapas das camadas convolucionais. Essa configuração não foi avaliada neste TCC, que manteve os valores 16 e 36. Assim, o experimento realizado reproduz apenas a primeira etapa de variação da camada oculta. Como o artigo também não informa todos os hiperparâmetros, as sementes ou o procedimento de repetição, os dados disponíveis não permitem atribuir a diferença de desempenho a um único fator.

5.3 Discussão à luz dos métodos tradicionais

Os experimentos deste trabalho avaliaram somente variações da arquitetura CNN e não reimplementaram métodos tradicionais sob as mesmas partições de dados. Portanto, os resultados não constituem uma comparação experimental direta com LBPH, *Eigenface* ou *Fisherface*.

Na literatura revisada, Silva (2023) relata desempenho superior do *Facenet* em relação aos métodos tradicionais avaliados. A acurácia média de 95,85%, com desvio padrão de 2,77 pontos percentuais, obtida neste TCC é compatível com a relevância das CNNs indicada por esse estudo. Entretanto, diferenças de base de dados, protocolo e arquitetura impedem atribuir superioridade aos modelos deste trabalho sem uma avaliação controlada dos métodos sob as mesmas condições.

5.4 Discussão sobre a aplicabilidade em cenários reais

Os experimentos foram executados em uma CPU Intel Core i7-9700KF, sem aceleração por GPU. Esse resultado demonstra que o treinamento das configurações avaliadas foi possível no computador descrito, mas não permite concluir

que o modelo tenha baixa demanda computacional. Não foram medidos tempo de treinamento, latência de inferência, consumo de memória ou energia.

O controle de acesso cooperativo, como em portarias e catracas, é uma possível aplicação futura porque suas condições podem se aproximar das imagens frontais e da iluminação controlada do Olivetti Faces. A implantação nesse cenário, porém, depende de testes com captura real, variações ambientais e métricas de desempenho operacional.

A principal contribuição do trabalho é experimental: implementar a configuração com 16 e 36 mapas da arquitetura C1-C2-H, comparar quatro tamanhos de camada oculta em execuções repetidas e discutir os resultados no contexto de Visão Computacional e Aprendizado Profundo. Essa análise oferece uma base para pesquisas futuras sobre autenticação facial, sem validar, por si só, o uso do modelo em sistemas reais.

5.5 Limitações encontradas e trabalhos futuros

Os resultados devem ser interpretados à luz das limitações técnicas e metodológicas do estudo.

5.5.1 Limitações

A principal restrição identificada refere-se à base de dados utilizada. O *Olivetti faces dataset*, embora consolidado na literatura acadêmica, possui um número reduzido de amostras (400 imagens) e baixa variabilidade de pose e expressões. As imagens, convertidas para escala de cinza e redimensionadas para 57×47 pixels, suprimem detalhes de textura e cor que poderiam ser cruciais para a distinção de faces muito semelhantes em um universo maior de indivíduos.

As 50 partições de cada configuração foram geradas sem sementes fixas e não foram pareadas entre as configurações. Assim, as diferenças observadas combinam o efeito do tamanho da camada oculta com a variação das partições e da inicialização dos pesos. Além disso, as avaliações repetidas foram usadas tanto para comparar as configurações quanto para escolher a camada oculta com 1024

neurônios. Embora uma partição definida com a semente 42 tenha sido usada posteriormente para gerar o histórico de treinamento, nenhum conjunto permaneceu intocado durante todo o processo de seleção. Por isso, a média reportada não constitui uma estimativa independente do desempenho da configuração escolhida.

Outra limitação foi a ausência de aceleração por GPU. Embora o ambiente contasse com uma NVIDIA GeForce RTX 2070, a delegação das operações via CUDA não foi ativada. Isso restringiu a velocidade de treinamento e a abrangência da busca de hiperparâmetros. Além disso, a ausência de medições de tempo, latência e consumo de recursos impede conclusões sobre eficiência operacional.

5.5.2 Trabalhos Futuros

Com base nas dificuldades encontradas e nas oportunidades de melhoria, propõem-se as seguintes direções para a continuidade da pesquisa:

- **Otimização do Ambiente de Treinamento:** Configurar as bibliotecas TensorFlow e CUDA para utilizar a aceleração por GPU. Essa configuração pode reduzir o tempo de cada época e ampliar a quantidade de arquiteturas avaliadas.
- **Expansão e Enriquecimento dos Dados:** Avaliar a arquitetura proposta em conjuntos de dados mais desafiadores e modernos, como o *Labeled Faces in the Wild* (LFW) (HUANG et al., 2007). Adicionalmente, recomenda-se a aplicação de técnicas de aumento de dados (*Data Augmentation*) durante o pré-processamento, introduzindo rotações, ruídos e variações de brilho artificiais para aumentar a robustez do modelo a condições adversas.
- **Implementação de Protótipo em Tempo Real:** Desenvolver uma aplicação que integre o modelo treinado à captura de vídeo ao vivo. Isso permitiria medir a latência de reconhecimento e o consumo de recursos durante o processamento contínuo de vídeo.

6 Conclusão

Este trabalho avaliou uma arquitetura CNN aplicada ao reconhecimento facial com base no modelo de [Xie, Li e Shi \(2019\)](#). A configuração com 16 e 36 mapas da arquitetura C1-C2-H foi implementada em Python com TensorFlow e Keras e treinada com o conjunto de dados Olivetti Faces. Também foram comparadas camadas ocultas com 128, 256, 512 e 1024 neurônios por meio de execuções repetidas.

Entre as configurações avaliadas, a de 1024 neurônios apresentou a maior acurácia média, de 95,85%, com desvio padrão de 2,77 pontos percentuais. A média obtida ficou abaixo dos 97,5% de teste apresentados por [Xie, Li e Shi \(2019\)](#) para a configuração 16-36-1024. Os 100% de teste do artigo correspondem à configuração 36-76-1024, que não foi avaliada neste trabalho, conforme discutido na Seção 5.2.

Os resultados mostram que a arquitetura alcançou acurácia elevada no Olivetti Faces sob as condições controladas do experimento. Contudo, os mesmos resultados das 50 execuções foram usados tanto para comparar as configurações quanto para reportar o desempenho da configuração escolhida, sem um conjunto final intocado durante todo o processo de seleção. Além disso, não foram realizados testes de latência, consumo de recursos ou implantação. Assim, os dados não sustentam conclusões sobre eficiência operacional ou viabilidade em aplicações reais.

A contribuição do trabalho consiste na implementação dessa configuração, na comparação dos tamanhos da camada oculta e na descrição da variabilidade entre execuções. Estudos futuros devem avaliar bases mais diversas, reservar dados para uma avaliação final após a seleção de hiperparâmetros e medir o desempenho computacional em cenários de uso.

Referências

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANÉ, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCHE, V.; VASUDEVAN, V.; VIÉGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y.; ZHENG, X. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. 2015. Software available from tensorflow.org. Disponível em: <<https://www.tensorflow.org/>>. Acesso em: 6 mar. 2026. Citado na página 17.

ABDALA, V. **Mais de 47 milhões podem estar sob vigilância de reconhecimento facial no Brasil, aponta estudo**. Estadão, 2023. Disponível em: <<https://www.estadao.com.br/link/cultura-digital/mais-de-47-milhoes-podem-estar-sob-vigilancia-de-reconhecimento-facial-no-brasil-aponta-estudo/>>. Acesso em: 6 mar. 2026. Citado na página 8.

ADETIBA, E.; JOHN, T.; AKINRINMADE, A.; MONINUOLA, F.; AKINTADE, O.; BADEJO, J. A. Evolution of artificial intelligence languages, a systematic literature review. **CoRR**, abs/2101.11501, 2021. Disponível em: <<https://arxiv.org/abs/2101.11501>>. Acesso em: 6 mar. 2026. Citado na página 16.

An Optimistic Future - Futorology. **Convolutional Neural Networks Explained (CNN Visualized)**. 2020. Disponível em: <<https://www.youtube.com/watch?v=pj9-rr1wDhM>>. Acesso em: 6 mar. 2026. Citado 2 vezes nas páginas 3 e 12.

CHOLLET, F. et al. **Keras**. Keras, 2015. Disponível em: <<https://keras.io/>>. Acesso em: 6 mar. 2026. Citado na página 17.

Educative. **Introduction to Convolutional Neural Networks**. 2024. Disponível em: <<https://www.educative.io/blog/introduction-to-convolutional-neural-networks>>. Acesso em: 6 mar. 2026. Citado 2 vezes nas páginas 3 e 14.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: TEH, Y. W.; TITTERINGTON, M. (Ed.).

Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010. (Proceedings of Machine Learning Research, v. 9), p. 249–256. Disponível em: <https://proceedings.mlr.press/v9/glorot10a.html>. Acesso em: 6 mar. 2026. Citado na página 15.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning.** MIT Press, 2016. Disponível em: <https://www.deeplearningbook.org/>. Acesso em: 6 mar. 2026. Citado 3 vezes nas páginas 10, 13 e 15.

HUANG, G. B.; RAMESH, M.; BERG, T.; LEARNED-MILLER, E. **Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments.** Amherst, 2007. Technical Report 07-49, University of Massachusetts Amherst. Disponível em: <https://people.cs.umass.edu/~elm/papers/lfw.pdf>. Acesso em: 6 mar. 2026. Citado na página 35.

IBM. **O que são redes neurais convolucionais?** 2020. Disponível em: <https://www.ibm.com/br-pt/think/topics/convolutional-neural-networks>. Acesso em: 6 mar. 2026. Citado 2 vezes nas páginas 3 e 11.

LUCA, G. D. **How ReLU and Dropout Layers Work in CNNs.** Baeldung on Computer Science, 2024. Disponível em: <https://www.baeldung.com/cs/ml-relu-dropout-layers>. Acesso em: 6 mar. 2026. Citado 2 vezes nas páginas 3 e 14.

MCCULLUM, N. **The Flattening and Full Connection Steps of Convolutional Neural Networks.** Nick McCullum, 2025. Disponível em: <https://www.nickmccullum.com/python-deep-learning/flattening-full-connection/>. Acesso em: 6 mar. 2026. Citado 2 vezes nas páginas 3 e 13.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: FÜRNKRANZ, J.; JOACHIMS, T. (Ed.). **Proceedings of the 27th International Conference on Machine Learning (ICML-10).** Haifa, Israel: Omnipress, 2010. p. 807–814. Disponível em: <https://icml.cc/Conferences/2010/papers/432.pdf>. Acesso em: 6 mar. 2026. Citado na página 15.

NEMAVHOLA, A.; CHIBAYA, C.; VIRIRI, S. A systematic review of cnn architectures, databases, performance metrics, and applications in face recognition. **Information**, v. 16, n. 2, 2025. ISSN 2078-2489. Disponível em: <https://www.mdpi.com/2078-2489/16/2/107>. Acesso em: 6 mar. 2026. Citado na página 18.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011. Citado na página 22.

Python Software Foundation. **What is Python? Executive Summary.**

Python Software Foundation, s.d. Disponível em: <<https://www.python.org/doc/essays/blurbs/>>. Acesso em: 6 mar. 2026. Citado 2 vezes nas páginas 16 e 22.

SAMARIA, F. S.; HARTER, A. C. **Olivetti Research Laboratory Face Dataset.** 1994. Disponível em: <https://www.researchgate.net/figure/Olivetti-research-laboratory-face-dataset_fig15_326852590>. Acesso em: 6 mar. 2026. Citado 2 vezes nas páginas 3 e 20.

_____. **The ORL Database of Faces.** 1994. AT&T Laboratories Cambridge. Disponível em: <<https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabases.html>>. Acesso em: 6 mar. 2026. Citado na página 19.

Secretaria de Estado de Polícia Militar do Rio de Janeiro. **Governo do Estado chega à marca de 500 prisões realizadas com auxílio da tecnologia de reconhecimento facial.** 2025. Disponível em: <<https://sepm.rj.gov.br/2025/03/governo-do-estado-chega-a-marca-de-500-prisoas-realizadas-com-auxilio-da-tecnologia-de-reconhecimento-facial/>>. Acesso em: 6 mar. 2026. Citado na página 8.

SILVA, E. V. C. e. **Avaliação de algoritmos de reconhecimento facial para autenticação de usuários.** Monografia (Trabalho de Conclusão de Curso (Graduação em Ciência da Computação)) — Universidade Federal de Uberlândia, Uberlândia, MG, Brasil, 2023. Disponível em: <<https://repositorio.ufu.br/handle/123456789/41074>>. Acesso em: 6 mar. 2026. Citado 3 vezes nas páginas 8, 18 e 33.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, n. 56, p. 1929–1958, 2014. Disponível em: <<https://www.jmlr.org/papers/v15/srivastava14a.html>>. Acesso em: 6 mar. 2026. Citado na página 13.

TEAM, C. **Rectified Linear Unit (ReLU) Function in Deep Learning.**

Codecademy, 2025. Disponível em: <<https://www.codecademy.com/article/rectified-linear-unit-relu-function-in-deep-learning>>. Acesso em: 6 mar. 2026. Citado 2 vezes nas páginas 3 e 15.

WALT, S. Van der; SCHÖNBERGER, J. L.; NUNEZ-IGLESIAS, J.; BOULOGNE, F.; WARNER, J. D.; YAGER, N.; GOUILLART, E.; YU, T. scikit-image: image processing in python. **PeerJ**, PeerJ Inc., v. 2, p. e453, 2014. Citado na página 22.

XIE, Z.; LI, J.; SHI, H. A face recognition method based on cnn. **Journal of Physics: Conference Series**, v. 1395, n. 1, p. 012006, nov 2019. Disponível em: <<https://iopscience.iop.org/article/10.1088/1742-6596/1395/1/012006>>. Acesso em: 6 mar. 2026. Citado 8 vezes nas páginas 8, 18, 19, 25, 30, 32, 33 e 36.

ZHAO, X.; WANG, L.; ZHANG, Y.; HAN, X.; DEVECI, M.; PARMAR, M. A review of convolutional neural networks in computer vision. **Artificial Intelligence Review**, v. 57, n. 4, p. 99, mar 2024. Citado 2 vezes nas páginas 12 e 13.