INSTITUTO FEDERAL DO ESPIRITO SANTO
SISTEMAS DE INFORMAÇÃO

ARTHUR FRIÇO GRILLO

CLASSIFICAÇÃO DE KANJIS UTILIZANDO UMA REDE NEURAL CONVOLUCIONAL PROFUNDA

Cachoeiro de Itapemirim
2022

ARTHUR FRIÇO GRILLO

CLASSIFICAÇÃO DE KANJIS UTILIZANDO UMA REDE NEURAL CONVOLUCIONAL PROFUNDA

Trabalho de Conclusão de Curso apresentado à Coordenadoria do Curso de Sistemas de Informação do Instituto Federal do Espírito Santo, Campus Cachoeiro de Itapemirim, como requisito parcial para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Dr. Ricardo Maroquio Bernardo

Cachoeiro de Itapemirim

2022

(Biblioteca do Campus Cachoeiro de Itapemirim)

G859c Grillo, Arthur Friço.

Classificação de Kanjis utilizando uma rede neural convolucional profunda / Arthur Friço Grillo. - 2022.
66 f. : il. ; 30 cm..

Orientador: Ricardo Maroquio Bernardo

TCC (Graduação) Instituto Federal do Espírito Santo, Campus Cachoeiro de Itapemirim, Sistemas de Informação, 2022.

1. Redes neurais (Computação). 2. Reconhecimento óptico de caracteres. 3. Classificação de imagens . 4. Caracteres chineses - Japão. I. Bernardo , Ricardo Maroquio. II.Título III. Instituto Federal do Espírito Santo.

CDD: 006.32

Bibliotecário/a: Renata Lorencini Rizzi CRB6-ES nº 085

MINISTÉRIO DA EDUCAÇÃO
INSTITUTO FEDERAL DO ESPÍRITO SANTO
CAI - COORDENADORIA DO CURSO DE BACHARELADO EM SISTEMAS DE INFORMACAO

FOLHA DE APROVAÇÃO-TCC Nº 12/2022 - CAI-CCSI (11.02.18.01.08.02.13)

Nº do Protocolo: 23151.002801/2022-03

Cachoeiro De Itapemirim-ES, 05 de julho de 2022.

ARTHUR FRIÇO GRILLO

CLASSIFICAÇÃO DE KANJIS UTILIZANDO UMA REDE NEURAL CONVOLUCIONAL PROFUNDA

Trabalho de Conclusão de Curso apresentado à Coordenadoria de Informática do Instituto Federal do Espírito Santo, como requisito parcial para obtenção de título de Bacharel em Sistemas de Informação.

Aprovado em 11 de março de 2022

COMISSÃO EXAMINADORA

Ricardo Maroquio Bernardo, D.Sc.
Instituto Federal Do Espírito Santo
Orientador

Rafael Vargas Mesquita dos Santos, D.Sc.
Instituto Federal do Espírito Santo

Susana Brunoro Costa de Oliveira, D.Sc.
Instituto Federal do Espírito Santo

(Assinado digitalmente em 06/07/2022 10:24)
RAFAEL VARGAS MESQUITA DOS SANTOS
PROFESSOR DO ENSINO BASICO TECNICO E TECNOLOGICO
CAI-CCSI (11.02.18.01.08.02.13)
Matrícula: 1544937

(Assinado digitalmente em 05/07/2022 14:56)
RICARDO MAROQUIO BERNARDO
PROFESSOR DO ENSINO BASICO TECNICO E TECNOLOGICO
CAI-CCSI (11.02.18.01.08.02.13)
Matrícula: 2152606

(Assinado digitalmente em 05/07/2022 17:30)
SUSANA BRUNORO COSTA DE OLIVEIRA
PROFESSOR DO ENSINO BASICO TECNICO E TECNOLOGICO
CAI-CCSI (11.02.18.01.08.02.13)
Matrícula: 1505999

Para verificar a autenticidade deste documento entre em https://sipac.ifes.edu.br/public/documentos/index.jsp informando seu número: **12**, ano: **2022**, tipo: **FOLHA DE APROVAÇÃO-TCC**, data de emissão: **05/07/2022** e o código de verificação: **afd423a3a3**

RESUMO

Umas das partes mais complexas do aprendizado de japonês é a identificação de kanjis, um dos sistemas de escrita da língua. Muitas vezes eles são difíceis de serem pesquisados, especialmente quando são encontrados em imagens ou textos não editáveis. Neste trabalho, foi desenvolvido um novo modelo de redes neural convolucional profunda para o reconhecimento de kanjis em imagens, assim como uma interface gráfica para demonstrar o uso dessa tecnologia no aprendizado. Essa rede foi treinada para classificar um conjunto de demonstração de 222 caracteres kanjis e os seus resultados foram comparados com outros dois modelos, treinados no mesmo *dataset*.

Palavras chave: kanji, Rede Neural Convolucional e Modelos MNIST.

ABSTRACT

One of the most complex aspects of learning Japanese is identifying kanji characters, one of the language's writing systems. They are often difficult to be searched, especially when located on non-editable images or text. In this work, a new model of deep convolutional neural networks was developed for the recognition of kanji characters in images, as well as a graphical interface, to demonstrate the use of this technology in learning. This network was trained to classify a demo set of 222 kanji characters and its results were compared with two other models, trained on the same dataset.

Keywords: kanji, Convolutional Neural Network and MNIST Models.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O sistema de escrita kanji não tem suas origens no Japão, mas na China. Com o crescimento das rotas comerciais do século I A.C., o sistema de escrita chinês começou a chegar no Japão através de selos, moedas, espadas, cartas, entre outros objetos importados da China. Com os anos, esses caracteres foram sendo adaptados pelos japoneses para melhor representarem a língua local, eles são usados como base para um dos três sistemas de escrita que formam a língua japonesa atualmente.

Aprender uma língua nova não é uma tarefa fácil. É necessário aprender novas palavras, novas regras gramaticais, novas pronúncias, etc. O que pode acabar sobrecarregando os estudantes, especialmente se a língua estudada não é originada da mesma proto-língua, por exemplo: um chines vai ter muita dificuldade aprendendo português, isso porque as duas línguas vieram de proto-línguas completamente diferentes. Isso é um problema ainda maior para japoneses, pelo fato do país se localizar em uma ilha isolada à milhares de anos, a língua deles é única que não tem um parentesco com outras línguas asiáticas, o que a torna especialmente difícil de se aprender.

Essa enorme dificuldade de aprendizado também pode ser vista como uma grande oportunidade de desenvolvimento. As tecnologias de informação podem ser usadas para ajudar o aprendizado de línguas, de formas nunca usadas pela humanidade. Um dos campos de tecnologias de informação que tem crescido muito nos últimos anos são as redes neurais artificiais. Elas têm permitido avanços em diversos campos, um especialmente relevante para línguas é o reconhecimento de caracteres. Uma rede convolucional pode ser treinada para receber uma imagem de entrada, como uma imagem contendo um kanji, e classificá-la com uma enorme acurácia quando comparado a outros métodos (Data Science Academy, 2022).

As redes convolucionais já foram usadas com sucesso no passado para o reconhecimento de imagens (SIMONYAN; ZISSERMAN, 2014). E também para a detecção de caracteres (WANG et al., 2012) (PTUCHA et al., 2019). Isso inclui caracteres mais complexos como os chineses e japoneses (ZHANG et al., 2017) (LY et al., 2017).

Porém, ainda existe muito a ser feito na área de redes neurais convolucionais, já que é uma área nova que tem crescido muito nos últimos anos.

1.1 PROBLEMA

Para facilitar a exemplificação do problema, é necessário primeiro um entendimento básico do que são kanjis. Kanjis são os caracteres que compõem um dos três sistemas de escrita usados no Japão. A principal diferença entre o sistema de escrita romano e o sistema de kanjis, é que as suas palavras representam sons, enquanto kanjis representam significados. Nele não é possível identificar o som de um caractere apenas olhando para ele. Isso se torna um enorme problema, porque a pronúncia dos kanjis é necessária para eles serem escritos em teclados que usam o alfabeto romano, como o famoso QWERTY.

Quando um indivíduo está tentando aprender um novo idioma, uma das coisas mais normais de acontecer é ele se deparar com uma palavra ou conceito que ele não sabe o que significa, se um usuário está navegando pela internet e encontra com uma palavra da qual não tem conhecimento, por exemplo, *TREE*, ele não vai saber o que ela significa, mas como ela está no mesmo sistema de escrita que a língua portuguesa usa, ele pode simplesmente escrevê-la em um aplicativo de tradução ou em um dicionário para descobrir o seu significado.

Porém, com kanjis essa situação aconteceria de uma forma diferente, como foi explicado, a pronúncia de um kanji é necessária para escrevê-lo, mas diferentemente do português, não é possível descobrir a pronúncia de um kanji apenas olhando para ele. Se um usuário se deparasse com o kanji mostrado na figura 1, ele não conseguiria descobrir o significado do kanji porque o aspecto visual do caractere não oferece a informação necessária para pesquisar o kanji em um dicionário.

Figura 1 – Kanji que representa uma árvore

Se esse kanji estiver em um texto selecionável, como um PDF ou um texto HTML5, o estudante poderia simplesmente copiar o kanji e usá-lo para descobrir seu significado, porém se ele estiver em uma forma não selecionável, como em uma imagem ou um vídeo, a situação se torna mais complicada, pois o usuário é forçado a descobrir o significado do kanji de uma forma indireta, que normalmente é mais trabalhosa e mais demorada.

1.2 OBJETIVOS

Para solucionar esse problema, uma rede neural convolucional recebe uma imagem e classifica o kanji presente nela em tempo real. A imagem será criada por um aplicativo de computador, que permitirá aos usuários selecionarem a parte da tela que contém o kanji desejado utilizando o mouse, o programa então geraria uma imagem e usaria essa imagem como a entrada da rede neural de classificação, mostrando o kanji para os usuários de uma forma fácil, rápida e eficiente, o que levaria a um melhor aprendizado do idioma japonês.

Esse trabalho tem como objetivo geral desenvolver o aplicativo descrito no parágrafo anterior, assim como um modelo de rede neural convolucional profunda que consiga classificar caracteres kanji, demonstrando a viabilidade de redes neurais para solucionar esse problema e facilitar o aprendizado do idioma.

Para alcançar o objetivo geral, os seguintes objetivos específicos devem ser alcançados:

a) Encontrar e modificar um *dataset* que seja grande o suficiente para demonstrar a efetividade da rede no reconhecimento de kanjis;

b) Construir um modelo de rede neural convolucional com base em uma rede de uma arquitetura já existente, que consiga classificar imagens de kanjis;

c) Treinar a rede usando o *dataset* de kanjis;

d) Construir um programa que permita o usuário inserir uma imagem de um caractere kanji, e receber uma resposta do programa;

e) Analisar os resultados para determinar se a rede teve um bom desempenho na classificação de caracteres kanji.

1.3 JUSTIFICATIVAS

O aprendizado de línguas estrangeiras permite a um indivíduo descobrir novas experiências, se encantar com novas visões do mundo e se comunicar com pessoas que convivem em uma cultura totalmente diferente da dele. O caminho é longo e bem inclinado, mas a recompensa por essa jornada é algo muito gratificante para ser descrito. Por isso, qualquer projeto que permita mais pessoas a aprenderem uma nova língua é importante, não só para esses indivíduos, mas como para a sociedade globalizada do mundo atual.

Aprender japonês também é algo muito útil para o mercado de trabalho, especialmente se o objetivo for trabalhar com uma companhia japonesa. De acordo com um estudo de 2016, mais de 70% dos japoneses não se consideram habilidosos em inglês, com apenas 7% se considerando habilidosos na língua (RAKUTEN, 2016). O Japão também tem o terceiro maior produto interno bruto (PIB) do planeta e um enorme mercado de trabalho, além de ser um excelente ponto turístico.

1.4 ORGANIZAÇÃO DO TRABALHO

Neste capítulo foi feita a introdução ao tema do trabalho, o problema que ele está tentando resolver, os objetivos do seu desenvolvimento e como ele tentará alcançar esses objetivos. O segundo capítulo foca na parte teórica do trabalho. Ele introduz os conceitos de kanjis e explica o que são as redes neurais convolucionais, como elas surgiram, como elas funcionam e como elas vão ser usadas par tentar alcançar os objetivos deste trabalho.

O terceiro capítulo foca no desenvolvimento do modelo de rede neural usado neste trabalho, assim como seu *dataset* de treinamento e a sua interface visual. Esse capitulo também apresenta explicações dos outros dois modelos, que serão usados como base de comparação para o modelo deste trabalho.

Por fim, o quarto capítulo apresenta os resultados deste trabalho, como o modelo

desenvolvido se compara com os outros dois apresentados e as possíveis direções que trabalhos futuros podem seguir, a partir dos sistemas desenvolvidos neste.

2 REFERENCIAL TEÓRICO

Esse capítulo vai abranger a parte teórica do trabalho. Ele vai apresentar os conceitos principais do sistema de escrita kanji, o funcionamento de redes neurais convolucionais, como essas redes podem ser usadas para classificar caracteres kanji e outros artigos relevantes.

2.1 TRABALHOS RELACIONADOS

Essa seção apresenta trabalhos que contribuirão para a produção deste artigo, alguns deles são artigos científicos que explicam os conceitos usados, enquanto outros projetos mostram como outros grupos tentaram resolver o problema de classificação de caracteres kanjis em tempo real.

2.1.1 A Neural Network Approach to Character Recognition

O artigo “A Neural Network Approach to Character Recognition” foi publicado em 1989, pelos pesquisadores A. Rajavelu, M. T. Musavi e M. V. Shirvaikar, da universidade de Maine, nele foi apresentada uma nova abordagem de reconhecimento de caracteres, que utiliza uma rede neural. Um dos principais problemas com outros sistemas de reconhecimento de caracteres era a inflexibilidade presente nelas. Elas tinham dificuldade de lidar com fontes de tamanhos variados ou com textos manuscritos, isso devido às pequenas variações que poderiam confundir o algoritmo. Eles resolveram esse problema de duas formas, a primeira foi usando um conjunto de técnicas de extração de característica e normalização de texto. Essas estratégias foram usadas para tornar os dados de entrada mais consistentes. A segunda foi a implementação de uma rede neural e o seu treinamento. Esse treinamento foi conduzido com diversas fontes, assim como textos manuscritos (RAJAVELU; MUSAVI; SHIRVAIKAR, 1989).

O resultado dessa nova abordagem foi excelente, para a época, com margens de acurácia de 97% e 99% para as diferentes fontes usadas no teste. A velocidade e sensibilidade da rede também foram significativas. Ela conseguiu reduzir a quantidade de dados necessários para o processo, quando comparado com as abordagens anteriores.

Esse trabalho também demonstrou a capacidade dessas redes de se adaptarem a escopos maiores, se necessário.

Sistemas como esse foram aprimorados nos anos seguintes. A principal diferença dessa abordagem, quando comparada com as atuais, é a rede utilizada. Atualmente as arquiteturas de rede neural mais usadas para análise de imagens são redes neurais convolucionais, pois essas mostram uma melhor flexibilidade, não só durante o treinamento mas também durante seu uso. Outra parte importante, é que essa rede estava trabalhando com o alfabeto romano moderno que contém apenas 26 caracteres. Redes mais modernas como a que será usada neste artigo, conseguem trabalhar com uma quantidade muito maior de classes o que a torna viável para a categorização de caracteres do sistema kanji.

2.1.2 ImageNet Classification with Deep Convolutional Neural Networks

Esse foi um trabalho extremamente influente, não só para redes neurais, mas para muitas outras tecnologias que continuam sendo usadas atualmente. Nele os engenheiros Alex Krizhevsky, Ilya Sutskever e Geoffrey E. Hinton, mostram o treinamento de uma rede neural convolucional profunda para visão computacional. O treinamento dessa rede foi feito com mais de 1,2 milhões de imagens, divididas em 1000 classes diferentes. Esse foi um dos primeiros usos modernos de tecnologias de redes neurais convolucionais para a categorização de imagens. A rede treinada nesse trabalho tem taxas de erro entre 17% e 37%, o que para a época, eram melhores que o estado da arte anterior. Esses resultados foram alcançados sem o uso de pré-treinamento não supervisionado, uma estratégia que os autores acreditavam que poderia ser usada para melhorar ainda mais os resultados da rede (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

Esse trabalho é de extrema importância para o desenvolvimento moderno de redes neurais convolucionais de visão digital. A popularidade desse projeto, comumente chamado AlexNet, inspirou o desenvolvimento de muitas outras abordagens para o problema de classificação de imagens nos anos seguintes, muitas delas até utilizando o mesmo *dataset* utilizado da competição ImageNet, onde esse trabalho foi construído.

2.1.3 MobileNets: Efficient Convolutional Neural Networks for Mobile Vision

Depois que a AlexNet venceu o desafio ImageNet ILLSVRC em 2012, (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) a popularização de redes neurais como uma solução para visão computacional tem se tornado uma estratégia cada vez mais popular. Em 2017, um grupo de programadores da Google desenvolveu um novo modelo para redes neurais, chamado MobileNets. MobileNets é um modelo de rede neural convolucional para visão de aplicações que tem um foco em dispositivos mais fracos, como celulares ou outros dispositivos mobile. Os pesquisadores desse projeto introduzem dois hiper-parâmetros para permitir que o construtor do modelo possa escolher o tamanho correto para a aplicação, baseando-se nas constantes do problema. Assim, a rede consegue trabalhar de uma forma equilibrada, balanceando eficiência e velocidade mesmo quando ela estiver sendo usada em dispositivos mais fracos. Por fim, o artigo também apresenta testes de desempenho em diversas aplicações, para demonstrar a rede sendo usada na prática (HOWARD et al., 2017).

Esse trabalho demonstra ser possível utilizar uma rede neural convolucional em dispositivos fracos mantendo um alto desempenho. Isso abre as portas para o desenvolvimento de diversos sistemas de visão computacional mais acessíveis. O baixo custo computacional permite que essa rede seja mais adaptável, servindo novos nichos de mercado que não tinham acesso a essa tecnologia. Essas são algumas das razões para ela ter sido escolhida para o programa sendo desenvolvido neste trabalho.

2.1.4 Kanji character detection from complex real scene images based on character properties

Um dos maiores desafios para reconhecimento de caracteres, são imagens de cenas reais, como fotografias ou vídeos. Para essas imagens, encontrar onde estão os caracteres a serem lidos se torna um grande problema, devido à complexidade das cenas. Esse artigo traz uma abordagem mais generalizada, que torna possível utilizar apenas as características dos caracteres para encontrá-los nas cenas. O algoritmo consiste em cinco etapas distintas: pré-processamento, classificação inicial usando regras primitivas, classificação avançada usando AdaBoost, Markov random field (MRF) clustering (uma estratégia usada para combinar componentes conectados) e o pós-processamento. De uma forma simplificada, esse artigo mostra uma estratégia

para destacar caracteres kanji em imagens. Isso, por fim, informa a localização dos caracteres na imagem, permitindo uma melhor entrada de dados na rede (XU; NAGAYOSHI; SAKO, 2008).

Uma parte importante desse artigo é que a sua abordagem tem um foco especificamente em caracteres kanji, assim como este trabalho. Ele utiliza também um modelo que pode ser aplicado em outros programas, para melhorar a entrada. Essa não é a única abordagem para a detecção de kanjis, porém ela é customizável, eficiente e eficaz.

2.1.5 Deep Learning for Classical Japanese Literature

Um dos *datasets* mais usados no treinamento de redes neurais para o reconhecimento óptico de caracteres é o MNIST (LECUN et al., 1998), esse é um *dataset* de números manuscritos, muito usado no treinamento de redes de reconhecimento e categorização de caracteres, ou para educação, já que ele é bem fácil de usar. Esse trabalho teve como objetivo criar uma versão do MNIST para caracteres dos sistemas de escrita da língua japonesa. Esse *dataset* foi chamado de Kuzushiji-MNIST. Ele foi construído a partir de livros manuscritos japoneses, trazendo uma coleção de mais de 400.000 imagens rotuladas de caracteres japoneses escritos a mão. Ele é dividido em duas partes, a primeira é chamada Kuzushiji-49 ela contém 266.407 imagem de caracteres Hiragana, um dos alfabetos silábicos usados na língua japonesas, a segunda parte se chama Kuzushiji-Kanji, e contém 140.426 imagens de mais de 3832 kanjis diferentes. Kuzushiji-49 é composto por imagens de 28×28 pixeis em escala cinza e Kuzushiji-Kanji é composto por imagens de 64×64 pixels em escala cinza (CLANUWAT et al., 2018).

A principal vantagem deste *dataset*, quando comparado a outros *datasets* da língua japonesa, é a sua consistência com redes desenvolvidas para o sistema MNIST. Ele pode ser usado com códigos desenvolvidos para o MNIST, com uma quantidade mínima de alterações. Ele também é bem denso, contendo uma abundância de imagens rotuladas, melhorando o treinamento das redes. Porém, ele não vem sem desvantagens, a principal delas está no Kuzushi-Kanji, devido a seu excesso de classes, a quantidade de imagens para cada uma dessas classes varia muito. Com os kanjis mais usados tendo milhares de imagens e os menos usados tendo pouquíssimas

imagens, alguns do kanjis realmente incomuns contém apenas uma imagem de treino, o que acaba dificultando o treinamento dessas classes.

2.1.6 Kanjitomo

KanjiTomo é um aplicativo de classificação de kanjis em tempo real, desenvolvido para Windows. Ele tem como função escanear a tela do usuário e tentar identificar se ele está apontando para um kanji com o mouse, se esse for o caso, o programa tenta descobrir qual kanji está sendo apontado. Esse sistema funciona com textos escritos em qualquer parte da tela, desde que as fontes usadas sejam padronizadas. Ele faz isso através de um algoritmo de OCR e manipulação de imagem, tentando separar kanjis em blocos distintos e comparando esses blocos a uma base de dados interna, tentando encontrar semelhanças entre o carácter da tela e o da base (KanjiTomo Team, 2012).

O KanjiTomo serviu como inspiração para esse trabalho, porém existe uma diferença crucial entre o ele e o sistema sendo desenvolvido nesse trabalho. Ele utiliza táticas mais antigas de OCR, trabalhando principalmente com alinhamento e contagem de pixels, comparando as informações da imagem com os valores em sua base de dados, o programa deste trabalho, por outro lado, tenta utiliza redes neurais convolucionais para alcançar esse objetivo.

Esse aplicativo funciona bem, porém, ele tem algumas falhas. Ele foi desenvolvido para ler principalmente mangas, revista em quadrinhos japonesas, e por causa disso, ele funciona melhor com fontes mais simples e padronizadas, que estejam localizadas em fundos com alto contraste, como uma fonte preta em um fundo branco ou vice-versa. Redes neurais tem uma vantagem nesse aspecto, pois elas conseguem se adaptar melhor às informações mais distintas e complexas.

2.2 FUNDAMENTAÇÃO TEÓRICA

Essa seção apresenta uma fundamentação teórica de todos os conceitos que serão usados neste trabalho. Ela apresenta kanjis, reconhecimento óptico de caracteres, redes neurais simples e redes neurais convolucionais profundas. Ele também mostra

como esses conceitos podem trabalhar juntos para classificar caracteres complexos em tempo real.

2.2.1 Kanjis

Kanji é um sistema de escrita logográfico de caracteres de origem chinesa. Ele foi adotado pelo Japão em seus primórdios sendo usado até hoje junto de outros dois silabários japoneses, hiragana e katakana, na escrita do japonês. Kanji, assim como outros logogramas, não transcreve sons, mas sim pedaços mínimos de significados, também chamados de morfemas (SOANES; STEVENSON, 2004).

Muitos kanjis são construídos a partir de pedaços de significado chamados de radicais. Outra forma de expressar esse conceito é dizendo que kanjis são um conjunto de pequenas ideias formando uma ideia maior. Esses radicais podem ser usado para tentar deduzir o significado de um kanji, mas muitas vezes não funciona, especialmente com kanjis mais simples como os estudados no começo de um curso de japonês.

A língua japonesa não é composta apenas por logogramas como chinês, mas sim três sistemas de escrita diferentes. Além dos kanjis existem os sistemas de hiragana e katakana, esses são silabários, ou seja, tem um caractere para cada sílaba. Eles são sistemas bem simples e podem ser aprendidos em uma semana pela maioria das pessoas. Os kanjis, por outro lado, podem levar anos de aprendizado. Como eles representam significados e não sons, existe uma quantidade bem maior de caracteres, o sistema kanji, por exemplo, contém mais de 50.000 caracteres. Desses caracteres apenas 3.000 são necessários para leituras do dia a dia (MOROHASHI, 1960). Se um indivíduo conseguir memorizar 20 kanjis por dia, ele levaria mais de seis meses para aprender apenas os necessários para conversas, isso considerando que ele esteja aprendendo todos perfeitamente.

A quantidade de caracteres é uma das maiores dificuldades de se desenvolver um sistema de reconhecimento de caracteres para essa língua. Outra dificuldade é que kanjis são caracteres muito mais complexos que letras, contendo muitas vezes mais de sete linhas individuais para serem desenhadas em cada um, mas existem exemplos de sucessos quando se trata de reconhecimento óptico de caracteres derivados do

logograma chinês, tal como os trabalhos de Zhang, Cire E Wu (ZHANG et al., 2017) (CIREŞAN; MEIER, 2015) (WU et al., 2014).

2.2.2 Reconhecimento Óptico de Caracteres

O Reconhecimento Óptico de Caracteres (OCR, do inglês Optical Character Recognition), é uma tecnologia que permite o reconhecimento automático de caracteres passados para sistemas através de mecanismos visuais. No caso do corpo humano esses mecanismos seriam os olhos, para um computador esse mecanismo pode ser um scanner, uma câmera ou um arquivo de imagem. Após a informação visual ter sido coletada, o texto é convertido em texto manipulável para ele poder ser utilizado em outros processos (MITHE; INDALKAR; DIVEKAR, 2013).

O reconhecimento óptico de caracteres podem ser divididos em duas categorias: sistemas online e sistemas offline. Esses nomes podem dar a ideia de conectividade com a internet, mas nesse contexto online significa que o reconhecimento do caractere está acontecendo em tempo real, ou seja, o caractere está sendo escrito durante sua análise permitindo que o sistema considere vários outros aspectos, como a velocidade que está sendo escrito, o número de pinceladas, etc. Isso difere dos sistemas offline que analisam caracteres preexistentes em imagens ou fotografias importadas para o sistema (ISLAM; ISLAM; NOOR, 2017).

Nos últimos anos redes neurais têm sido cada vez mais usadas para o reconhecimento de caracteres, especialmente redes neurais convolucionais, isso é graças a capacidade dessas redes de imitar as funções de reconhecimento de padrões presentes em sistemas biológicos. (Data Science Academy, 2022). Elas também mostraram ser eficientes na categorização de caracteres mais complexos como os caracteres chineses, dos quais kanjis são derivados (CIREŞAN; MEIER, 2015).

Embora o OCR seja amplamente utilizado, especialmente por redes neurais, sua acurácia hoje ainda está longe de conseguir acompanhar a de uma criança. As redes dos artigos estudados para esse trabalho, por exemplo mostram precisões variando entre 95% e 99% (CLANUWAT et al., 2018) (WU et al., 2014) (SIMARD et al., 2003) (LY et al., 2017) (AFROGE; AHMED; MAHMUD, 2016). Na primeira vista esses podem

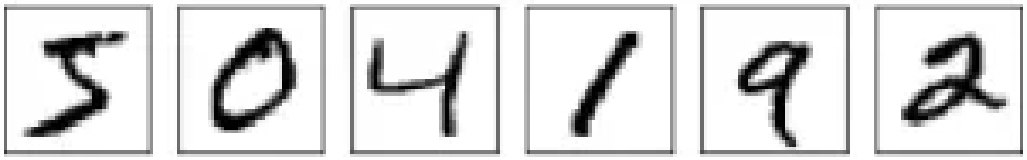
parecer níveis aceitáveis, mas eles são muito baixos comparados com o nível da acurácia de humanos.

2.2.3 Redes Neurais Artificiais

O cérebro humano é uma das coisas mais incríveis do planeta terra, uma máquina altamente poderosa que consegue processar uma quantidade enorme de informações em milésimos de segundos. Isso tudo é graças aos seus milhões de neurônios, células altamente interconectadas que juntas são capazes de fazer cálculos incrivelmente complicados, analisar imagens em tempo real e o mais importante, aprender. Para tentar emular essas capacidades do cérebro, foram criadas as redes neurais artificiais (RNAs), elas começaram a surgir nos anos 1943, quando Warren McCulloch e Walter Pitts criaram um modelo computacional para redes neurais baseadas em matemática e algoritmos denominados lógica de liminar (MCCULLOCH; PITTS, 1943). Esse modelo foi se desenvolvendo durante os anos através do trabalho de pesquisadores como Frank Rosenblatt, Kunihiko Fukushima e muitos outros (ROSENBLATT, 1957) (FUKUSHIMA, 1988). A evolução das redes neurais continua até hoje, com elas sendo usadas nos mais diversos campos da ciência.

As RNAs são compostas por uma coleção de unidades interconectadas chamadas de neurônios artificiais (ou neurônios matemáticos). Essas estruturas são baseadas nos neurônios biológicos que encontramos nos seres vivos. Uma forma fácil de entender esses neurônios é imaginá-los como funções de primeiro grau capazes de guardar um valor. Assim como uma função, cada neurônio é composto por variáveis, parâmetros e o resultado da equação. Neste trabalho uma rede de exemplo será usada para tornar a explicação de redes neurais mais intuitiva. Essa será uma rede neural que receberá uma imagem de escala cinza de 28x28 pixels como entrada e tentará reconhecer qual número está escrito na imagem recebida.

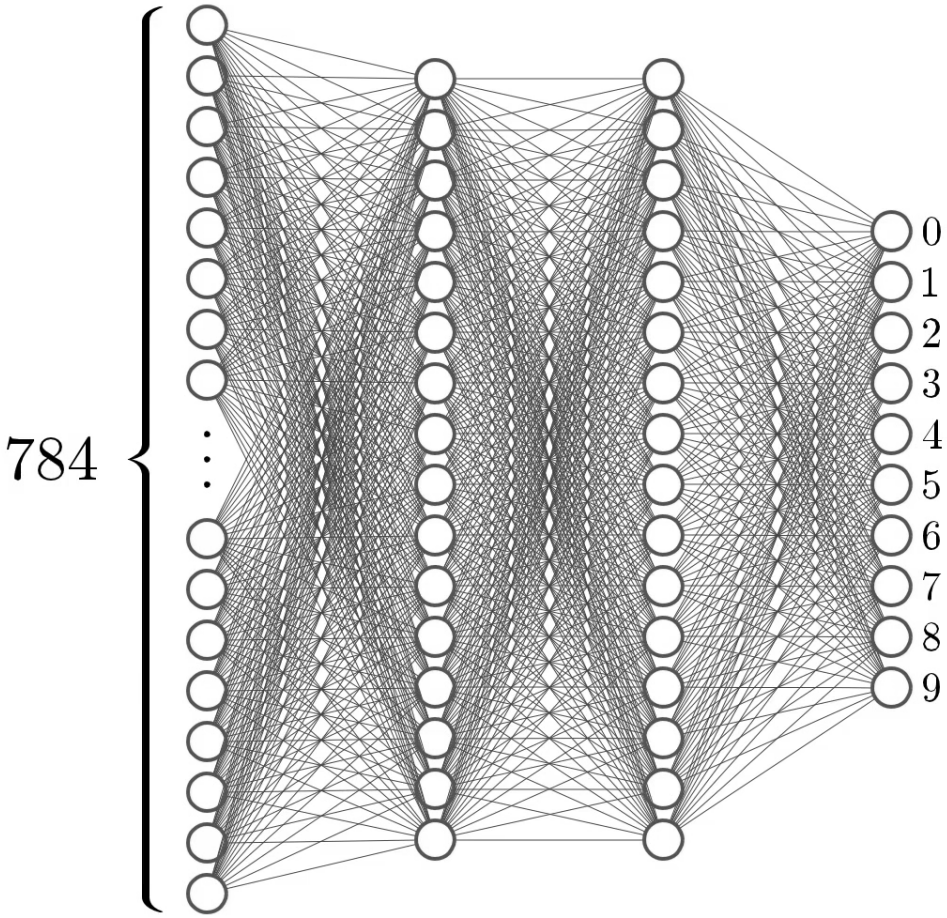
Figura 2 – Cinco exemplos de entradas para a rede



Fonte: The Deep Learning Book

Para que a rede possa interpretar essas imagens, suas informações devem ser inseridas na rede através dos neurônios da camada de entrada da rede (*input layer*). Camadas são conjuntos de neurônios com uma mesma função trabalhando em paralelo. Esse conceito é exemplificado na Figura 11. A camada de entrada dessa rede vai ter um neurônio para cada valor que ela for receber. Nesse caso, as imagens têm 784 (28x28) pixels que serão transformados em valores reais entre 0 e 1 representando a sua tonalidade de cinza, com branco sendo transformado em 0 e preto em 1. Quando um neurônio tem uma resposta próxima de 1 ele é considerado um neurônio ativo, e se o valor estiver perto de 0 um neurônio inativo.

Figura 3 – Visualização da rede neural de exemplo



Fonte: 3blue1brown series on neural networks

A última camada das redes neurais é conhecida como a camada de saída (*output layer*). Os neurônios dessa camada representam a resposta da rede para uma entrada, nesta rede a camada de saída tem 10 neurônios, cada um representando um dos possíveis números que podem estar escritos nas imagens. O objetivo da rede é que ela responda, ativando o neurônio do número da imagem que tenha recebido na camada de entrada e desativando os outros neurônios. Por exemplo, se a rede receber uma imagem com o número 3, essa rede deveria ativar apenas o neurônio de número 3 e desativar todos os outros 9.

As outras camadas entre a camada de entrada e a camada de saída são chamadas de camadas ocultas (*hidden layers*). A rede desse exemplo contém duas camadas ocultas, cada uma com 16 neurônios. Essa configuração não é um padrão, é possível

construir uma rede neural apenas com as camadas de entrada e saída, como também é possível construir uma rede com dezenas camadas ocultas, cada uma com milhares de neurônios. Se uma rede neural for construída com múltiplas camadas ocultas ela recebe o nome de rede neural artificial profunda. É dessas redes profundas que vem o termo *deep learning* (aprendizado profundo).

Para facilitar essa explicação é útil olhar para apenas um único neurônio. Como foi dito acima, os neurônios de uma rede neural são análogos a uma função. Eles recebem muitos valores e calculam um resultado usando uma série de parâmetros. Para um neurônio artificial, esses parâmetros são chamados de pesos e vieses (*weight and biases*). Cada neurônio tem múltiplos pesos **w**, um para cada neurônio da camada anterior, e um viés **b**. Por exemplo, o primeiro neurônio da primeira camada oculta tem 784 pesos e 1 viés. O neurônio também tem os valores **a**, esses são os valores de ativação de todos os neurônios da camada anterior. Esses valores são jogados em uma função **f(x)**, chamada função de ativação. O resultado dessa função determina o valor de ativação **R** do neurônio usando a fórmula abaixo.

Figura 4 – Função de ativação

$$\mathbf{R} = \mathbf{f}((\mathbf{a_1} * \mathbf{w_1}) + (\mathbf{a_2} * \mathbf{w_2}) + (\mathbf{a_3} * \mathbf{w_3}) + (\mathbf{a_4} * \mathbf{w_4}) + (\mathbf{a_5} * \mathbf{w_5}) + \ldots + (\mathbf{a_{784}} * \mathbf{w_{784}}) - \mathbf{b})$$

Para entender essa fórmula é melhor dar um passo de cada vez. A primeira parte a ser discutida é a relação entre os valores recebidos da camada anterior e os pesos do neurônio em questão. Uma forma de pensar nessa relação é que os pesos indicam o quanto cada neurônio da camada anterior vai influenciar a resposta dessa função, se o valor de um peso for alto, isso significa que uma mudança no neurônio relacionado a ele vai ter uma maior influência no valor de ativação do cálculo em questão e vice-versa. Por exemplo, se o peso **w2** for maior que o peso **w4** a mesma mudança em **a2** teria um maior efeito na resposta do que em **a4**. Especialmente se o peso w4 for negativo.

A próxima parte dessa fórmula é a função **f(x)**, também chamada de função de ativação. O trabalho dela é convergir a resposta do cálculo acima para um valor entre 0 e 1, com valores maiores que 1 convergindo para 1 e valores menores que 0 convergindo para

0. Existem diversas funções de ativação que são usadas em muitos tipos de redes neurais diferentes. Algumas das mais usadas são as funções Sigmóide, Tanh e ReLU (Data Science Academy, 2022).

Essas funções podem acabar causando um problema. Alguns neurônios são mais eficiente se forem ativados quando seu valor estiver acima de um valor arbitrário como 5, 10 ou -5 ao invés de 1. Para resolver isso, um viés **b** é adicionado no cálculo. Seu propósito é mover o ponto de ativação da função para um local desejado.

Esse cálculo descrito acima é a função de ativação de apenas um neurônio da rede. Todos os outros neurônios da rede realizam esse mesmo cálculo, mas com valores diferentes para seus pesos e seus vieses. Nessa rede relativamente pequena existem 12.960 pesos e 42 vieses para escolher. Em redes maiores podem existir centenas de milhares de pesos e vieses para serem decididos. Infelizmente decidir cada um desses valores manualmente é inviável. O próximo passo do desenvolvimento de redes neurais é fazer com que as redes consigam escolher o melhor valor para cada um desses milhares de parâmetros de uma forma autônoma. É assim que redes neurais aprendem, elas recebem dados de treinamento e usando esses dados elas refinam os seus parâmetros, e com isso melhoram o seu mecanismo de escolha.

Esse processo de aprendizado é realizado através de um método chamado de descida de gradiente. Ele funciona através de um mecanismo chamado de retropropagação (*backpropagation*). Quando uma rede neural é inicializada, todos os seus parâmetros começam de uma forma aleatória, existem maneiras de controlar essa aleatoriedade para melhorar a performance (Data Science Academy, 2022), mas isso está fora do escopo dessa explicação. O importante é entender que redes neurais são inicializadas sem saber nada sobre como trabalhar, se uma imagem foi inserida em uma rede que acabou de ser inicializada a resposta vai estar muito errada na maioria dos casos.

A ideia da descida de gradiente é usar uma função custo para descobrir como cada um dos milhares de parâmetros devem ser alterados para melhorar a rede. A primeira parte desse processo é calcular um valor de custo para os dados, esse cálculo tem dois componentes: a resposta da rede, ou seja, o valor dos neurônios da camada de

saída, e o rótulo da imagem, os valores que os neurônios da última camada deveriam ter. Esses dados são usados na função abaixo.

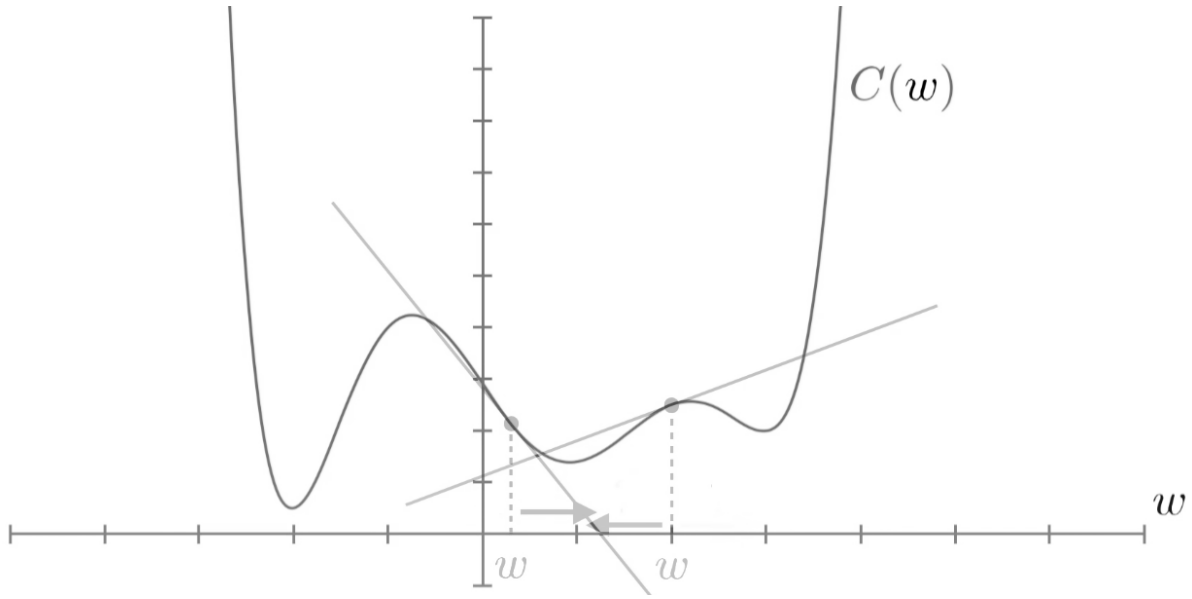
Figura 5 – Equação do valor de custo

$$\mathbf{C} = ((\mathbf{a_1} - \mathbf{y_1})^2 + (\mathbf{a_2} - \mathbf{y_2})^2 + (\mathbf{a_3} - \mathbf{y_3})^2 + (\mathbf{a_4} - \mathbf{y_4})^2 + ... + (\mathbf{a_{10}} - \mathbf{y_{10}})^2)$$

Nessa função o valor **C** representa o custo da resposta, os valores **a** representam as respostas dos 10 neurônios da última camada e os valores **y** representam os valores que a rede deveria ter respondido para a entrada que ela recebeu. Essa função retorna um custo baixo quando a resposta da rede está muito perto do resultado esperado e um valor alto quando a resposta está longe do valor esperado. Isso é útil porque o custo da função é usado para calcular o tamanho da alteração da rede, se a rede estiver muito errada ela deve ser alterada drasticamente, mas se ela estiver perto da resposta correta as mudanças não precisam ser drásticas.

O aprendizado de uma rede neural pode ser visto como uma função de localização do mínimo, ou seja, o objetivo é encontrar os parâmetros que resultam no menor custo possível. Com funções mais simples, esse valor pode ser calculado diretamente, mas as redes neurais não são simples, e calcular diretamente o valor mínimo em uma função com milhares de parâmetros é muito ineficiente. O que o processo de descida de gradientes faz é calcular a derivada de cada um dos parâmetros em relação ao custo, dessa forma, é possível saber para onde cada valor deve andar para reduzir o custo. Uma forma fácil de entender esse conceito é olhando para o gráfico da função de custo.

Figura 6 – Visualização das retas tangentes usadas na descida de gradiente



Fonte: 3blue1brown series on neural networks

O gráfico mostra as derivadas dos pesos em relação ao custo. Como pode ser observado, é possível usar a inclinação da reta tangente das derivadas para determinar em qual direção o valor de um peso **w** deve ser alterado para que o valor custo diminua. Esse cálculo de derivadas será repetido para todos os outros pesos e vieses de todos os outros neurônios dessa camada. Vale notar que o mínimo que as retas do gráfico estão tendo não é o mínimo possível. Esse método, mesmo sendo o melhor para resolver esse problema, não garante que a rede chegue no melhor resultado, mas ele garante que o resultado será bom.

Para esse exemplo foi usado apenas uma imagem, mas se o custo fosse calculado para apenas uma imagem, a rede estaria sendo treinada para identificar apenas aquela imagem. Quando redes neurais são treinadas, o resultado do custo é, na verdade, um média calculada entre os custos de um grupo de entradas. Dessa forma, a rede está sendo treinada para identificar qualquer uma dessas entradas corretamente, e não uma específica. Esses grupos são comumente chamados de lotes ou *batches* (Data Science Academy, 2022).

Esse método de calcular derivadas foi usado para descobrir como alterar os pesos e vieses da última camada. Os pesos e vieses das outras camadas são caculados

através do processo de retropropagação (*backpropagation*). A retropropagação nada mais é do que usar os resultados das mudanças de uma camada para calcular as mudanças da camada anterior, ou seja, a função acima foi usada para determinar como cada parâmetro da última camada deve ser alterado, tornando possível usar esses parâmetros para calcular como os parâmetros da camada anterior devem ser alterados e assim por diante, propagando as mudanças retroativamente pela rede.

Mesmo com todos esses sistemas de descida de gradiente e retropropagação, uma rede só será boa se seus dados de treinamento forem bons. Esse é o maior desafio do desenvolvimento de redes neurais, elas precisam ser treinada com dados de qualidade. Para a rede usada neste exemplo, um *dataset* usado foi o do MNIST (LECUN et al., 1998). Esse *dataset* contém 50.000 imagens de números rotuladas por humanos. Outras redes vão necessitar de outros *datasets* de dados. Uma rede sendo treinada para pilotar carros autônomos precisará de muitas imagens de ruas, enquanto uma criada para identificar falas de pessoas precisará de muitos arquivos de áudio com pessoas falando, e todos esses dados devem ser rotulados.

Rotular um dado apenas significa que um humano ou algum outro sistema fora da rede, manualmente indicou o que cada um dos dados é, para que a função de custo possa ser calculada. Esse tipo de aprendizado de rede é o mais comum ele é chamado de aprendizado supervisionado. Temos também redes que usam aprendizado não supervisionado. Esse é um tipo de treinamento de rede que não inclui rótulos. Nesse aprendizado as redes não são treinadas para desempenharem um trabalho específico, elas são apenas direcionadas a um resultado esperado. O aprendizado não supervisionado é mais lento que o supervisionado, mas ele é bem útil se for impossível adquirir dados rotulados para um aprendizado supervisionado (Data Science Academy, 2022).

É importante que a rede seja treinada com o maior *dataset* possível. Ter uma abundância de dados não só ajuda a rede a lidar com as mais diversas situações, mas também ajuda a reduzir o *overfitting*. *Overfitting* é um problema que ocorre quando uma rede é treinada com os mesmos dados por muito tempo. Isso faz com que a rede aprenda padrões muito específicos dos dados de treinamento, fazendo com que a rede não entenda dados que ela nunca viu. Por exemplo, se uma rede estiver sendo treinada

com um conjunto muito pequeno, a rede pode acabar ficando tão acostumada com os dados que ela deixa de detectar outros números, já que eles não são exatamente iguais ao que ela já está acostumada. (O'SHEA; NASH, 2015).

As redes neurais artificiais são uma inovação incrível da área de sistemas de informação. Elas ajudam a resolver diversos problemas nos mais variados mercados de trabalho. Existem muitos tipos de redes que são mais especializadas para certas tarefas com redes neurais recorrentes, redes de *Hopfield*, redes *LSTM (Long Short-Term Memory)*, *Generative Adversarial Networks* entre outras. Algumas das redes mais usadas, especialmente para o reconhecimento de imagens, são as redes neurais convolucionais. Uma rede neural convolucional será treinada nesse trabalho para o reconhecimento de caracteres kanji, com a intenção de auxiliar a leitura de japonês para aqueles que estão aprendendo a língua (Data Science Academy, 2022).

2.2.4 Redes Neurais Convolucionais Profundas

Redes convolucionais (ConvNets) são um algoritmo de aprendizado profundo que recebe uma imagem de entrada e atribui importâncias aos aspectos ou padrões da imagem, conseguindo identificar e diferenciar esses padrões. O pré-processamento exigido em uma ConvNet é muito menor em comparação com outros algoritmos de classificação, como o exemplificado acima. Enquanto nos métodos primitivos os filtros são feitos à mão, com treinamento suficiente, as ConvNets conseguem aprender as características para fazer essa configuração automaticamente. Por causa dessas características, as redes convolucionais são redes neurais especializadas no reconhecimento de padrões em imagens, especialmente imagens grandes e cheias de complexidade. (Data Science Academy, 2022)

A arquitetura de uma ConvNet é ainda mais análoga à conectividade dos neurônios no cérebro humano do que outras redes neurais mais tradicionais. Ela foi inspirada na organização do córtex visual, uma parte do cérebro humano responsável por interpretar as imagens captadas pelos olhos. Nele os neurônios individuais respondem a estímulos apenas em uma região restrita do campo visual, conhecida como campo receptivo. Uma coleção desses campos se sobrepõem para cobrir toda a área visual, permitindo a compreensão de padrões mais complexos.

Para explicar como as ConvNets conseguem trabalhar com imagens maiores, é importante primeiro explicar o problema que redes mais comuns, como a rede deste exemplo, muitas vezes apresentam. A rede usada de exemplo recebe uma entrada muito simples, uma foto em escala cinza com apenas o tamanho de 28x28 pixels. Essa entrada pode ser descrita com apenas 784 valores distintos, por outro lado, ConvNets costumam trabalhar com imagem que usam a escala de cores RGB (*red, green and blue*, ou vermelho, verde e azul em português). Isso significa que para cada pixel são necessários 3 neurônios diferentes. As imagens também tendem a ser bem maiores, se uma rede neural precisa trabalhar com imagens em 8K (7680 × 4320), por exemplo, ela vai precisar de 99.532.800 neurônios de entrada. Além disso, cada neurônio da segunda camada precisaria de 99.532.800 pesos diferentes, o que aumentaria muito o processamento necessário, não só para treinar a rede, mas para fazê-la funcionar.

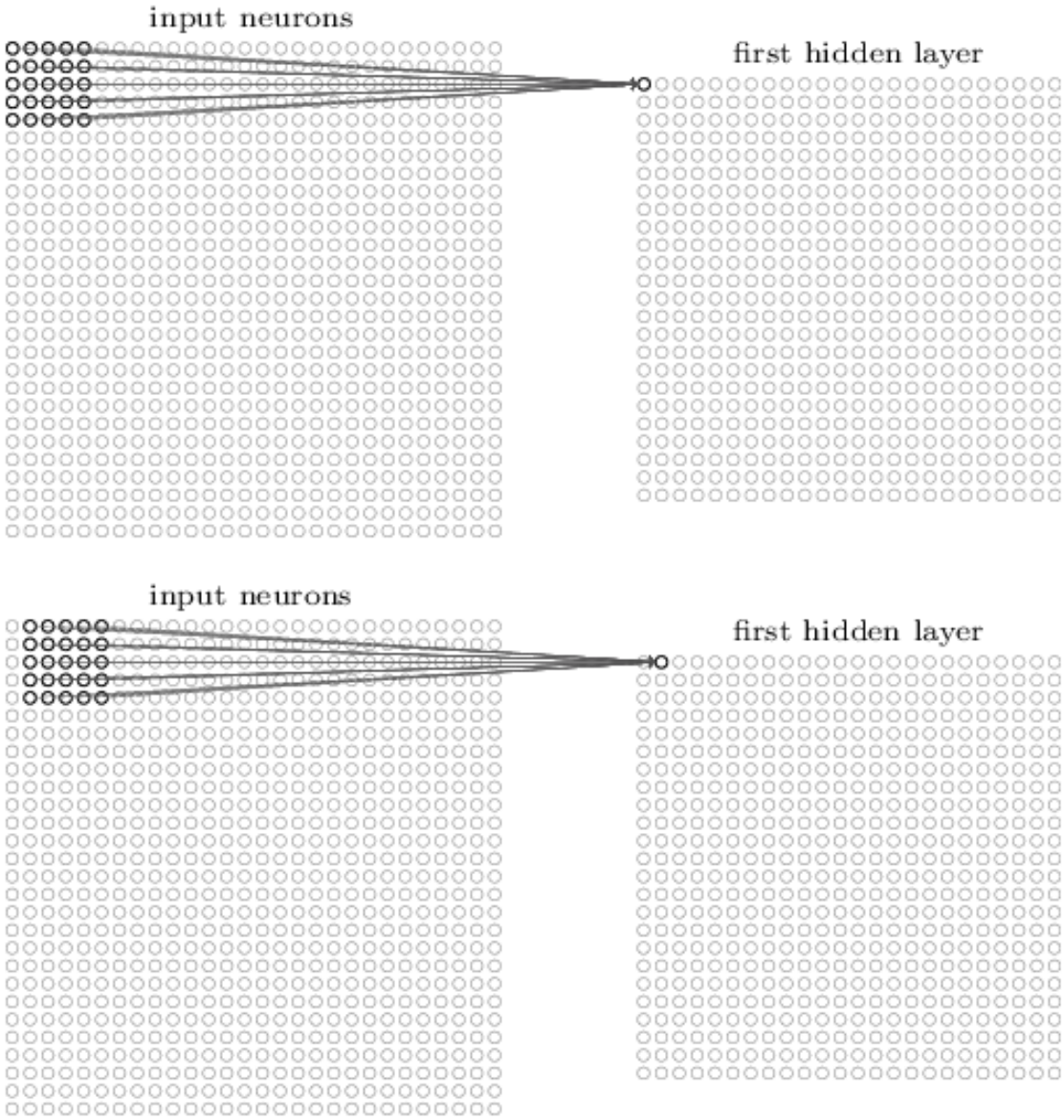
As imagens usadas neste trabalho são consideravelmente mais simples. Sendo compostas de apenas 36 bits de autura e largura e apenas uma escala de cor, mas mesmo sendo pequenas o suficiente para serem alimentadas a uma rede tradicional, melhores resultados podem ser alcançados utilizando redes neurais convolucionais.

A função de uma ConvNet é reduzir as imagens para um estado mais fácil de processar, sem perder recursos críticos necessários para uma boa previsão. Isso é importante para que a arquitetura seja boa lidando com recursos de aprendizado, e conjuntos de dados massivos. Para isso, as redes neurais convolucionais são baseadas em algumas ideias básicas: campos receptivos locais, pesos compartilhados e pooling.

Para tornar a explicação mais simples, cada uma dessas ideias vai ser explicada separadamente. O primeiro ponto a ser discutido são os campos receptivos locais, para isso é necessário entender o que é visão computacional. A visão computacional é o processo de modelagem e replicação da visão humana usando software e hardware, ou seja, ela é uma disciplina que tenta compreender uma cena 3D a partir de imagens 2D. Essa é uma ideia que tem sido cada vez mais usada nos últimos anos com rede de direção automática, reconhecimento de rostos, análise de documentos e previsões climáticas, sendo alguns dos sistemas que tem utilizado visão computacional através de redes neurais convolucionais.

Para tentar reproduzir os processos da visão computacional, as ConvNet utilizam campos receptivos locais. Como na rede de exemplo, as redes convolucionais precisam de um neurônio de entrada para cada pixel da imagem, ou mais dependendo do seu modelo de coloração, porém ao invés de conectar todos esses neurônios com todos os neurônios da primeira camada oculta da rede, conectam apenas uma pequena região da imagem com cada neurônio da camada oculta. Isso é muito mais fácil de entender com um exemplo visual.

Figura 7 – Visualização dos campos visuais em uma rede convolucional

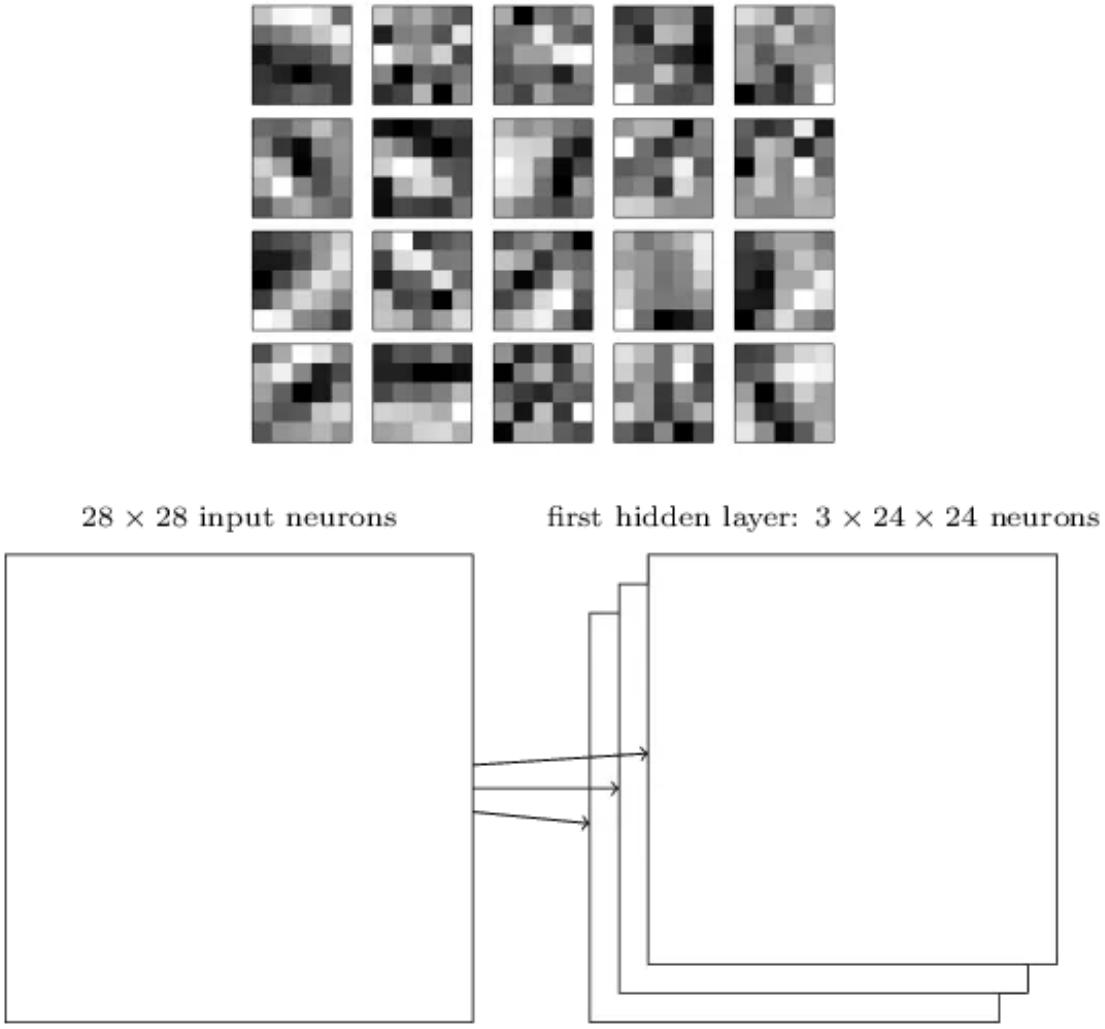


Fonte: The Deep Learning Book

Essas regiões na imagem de entrada são chamadas de campos receptivos locais para o neurônio oculto, com cada conexão tendo um peso e com o neurônio oculto tendo um viés. Esse campo de análise viaja pela imagem, ligando todos os possíveis espaços da camada de entrada com neurônios da camada convolucional. É importante saber que o tamanho desse campo não é constante, podendo mudar de rede para rede. Neste exemplo um campo de 5x5 foi usado apenas para exemplificação.

Todo esse processo de campos receptivos locais é apenas uma pequena divergência do modelo padrão, a parte realmente revolucionária está na forma que a rede convolucional trata os pesos da rede. Diferente de uma rede *feedforward*, todos os neurônios de cada uma das camadas ocultas convolucionais das redes têm os mesmos pesos. Por isso a rede alcança complexidade, tendo várias camadas convolucionais paralelas com pesos diferentes. Uma forma fácil de imaginar isso é que cada uma dessas camadas está procurando por apenas um padrão na imagem e cada um de seus neurônios olha em um lugar diferente da imagem para tentar encontrar o que quer.

Figura 8 – Visualização dos pesos em uma rede convolucional



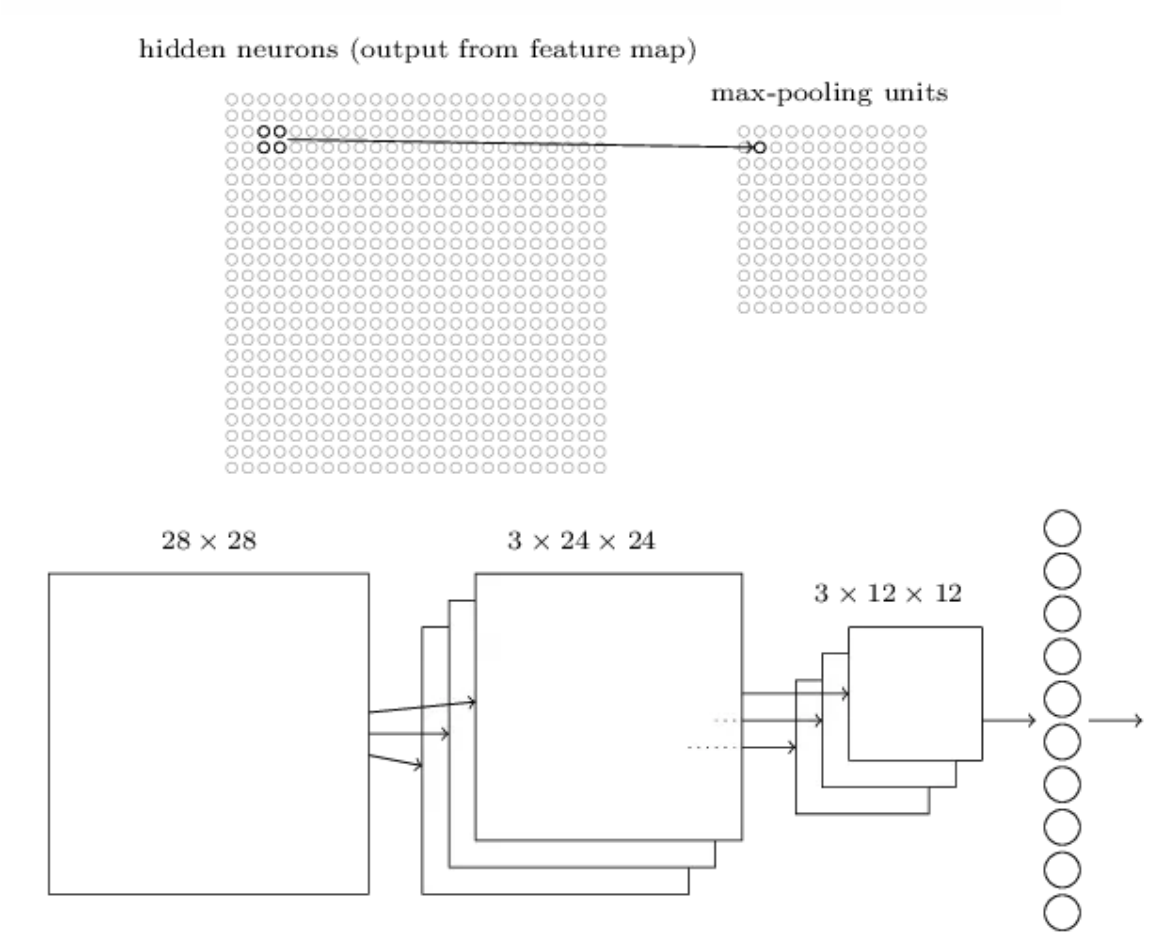
Fonte: The Deep Learning Book

Cada uma dessas imagens de 5x5 pixel representa o peso de uma das camadas exemplificadas abaixo deles. Os pixels mais escuros representam pesos maiores e os mais claros menores. A rede desse exemplo teria vinte camadas paralelas de neurônios, cada bloco de pesos tentando localizar um desses padrões na imagem. Isso mostra o porquê de redes convolucionais serem muito mais eficientes do que redes neurais. Uma rede com entrada de 28x28 pixel em escala cinza tem 25 (5×5) pesos compartilhados entre todos os neurônios de cada mapa, além de um único viés compartilhado. Portanto, cada mapa de recursos requer apenas 26 parâmetros. Essa rede tem 20 mapas de recursos para um total de 520 (20×26) parâmetros necessários para definir a camada

oculta dessa rede. Essa é uma redução enorme quando comparada com os 13 mil parâmetros da primeira camada oculta da rede de exemplo.

E por fim temos as camadas de pooling. Camadas de pooling são geralmente usadas imediatamente após os mapas de características. Elas têm a função de simplificar as informações na saída da camada convolucional. Por exemplo, cada unidade na camada de pooling pode resumir uma região de 2×2 neurônios na camada anterior. As camadas de pooling reduzem a quantidade de neurônios de cada mapa, diminuindo a quantidade de parâmetros necessários, e também são usadas para eliminar a informação posicional exata dos mapas, isso é feito para que a rede aprenda a detectar o padrão em questão em qualquer lugar da imagem. A intuição é que, uma vez que um recurso tenha sido encontrado, sua localização exata não é tão importante quanto sua localização aproximada em relação a outros recursos. O processo de *polling* pode ser visto na figura 7.

Figura 9 – Visualização das camada de pooling



Fonte: The Deep Learning Book

Esses são os mecanismos que fazem as redes convolucionais funcionarem, a partir desse ponto a rede se torna bem mais tradicional. A camada final da rede é uma camada totalmente conectada, ou seja, essa camada conecta todos os neurônios da camada de *max-pooling* a cada um dos neurônios de saída. É a mesma arquitetura da rede de exemplo. Vale notar que podem existir mais camadas ocultas entre as camadas de pooling e a camada de saída, mesmo que na imagem a camada de *poolling* leve diretamente à camada de saída.

As camadas convolucionais e de agrupamento são diretamente inspiradas nas noções clássicas de células simples e células complexas em neurociência visual. Essas redes neurais têm sido usadas desde dos anos 90 para o reconhecimento de fala e texto (SIMARD et al., 2003). A Microsoft, por exemplo, lançou sistemas de reconhecimento de

textos escritos que se baseiam em redes neurais em 1989 (WAIBEL et al., 1989). Essas redes também já foram usadas na leitura de caracteres de línguas mais complexas, como manuscritos chineses que são a base do kanji (CIREŞAN; MEIER, 2015). Isso inclui caracteres kanji atuais (LY et al., 2017) e caracteres kanji antigos (CLANUWAT et al., 2018). O que demonstra a sua eficácia na interpretação de caracteres logográficos.

Essa é uma tecnologia altamente modificável, que apresenta soluções para os mais diversos problemas, especialmente na área de classificação de imagens. Os modelos usados neste trabalho foram treinados para trabalhar com imagens mais simples do que as usadas nos exemplos acima, porém eles são excelentes demonstrações da tecnologia. Com esse conhecimento, é possível começar a entender como este trabalho consegue utilizar um modelo de rede neural convolucional para categorizar imagens de kanjis.

2.3 CONSIDERAÇÕES SOBRE O CAPÍTULO

Este capítulo aborda a parte teórica deste trabalho. Explicando os conceitos de kanjis, redes neurais, redes neurais convolucionais e como esses conceitos interagem. O próximo capítulo vai abordar o dataset, o modelo de rede neural e o aplicativo de interface gráfica desenvolvidos neste trabalho.

3 DESENVOLVIMENTO

Esse capítulo entra nos detalhes do processo de desenvolvimento deste projeto. Ele fala como o banco de dados *CharactersTrimPad28* foi modificado para melhor se enquadrar às especificações deste trabalho, além de como seus dados foram aprimorados através da API Keras.

Ele também explica como foi a construção dos três modelos usados neste trabalho, os tipos de camadas que eles usam, os motivos por trás das suas configurações e os códigos usados para treiná-los. Por fim, esse capítulo passa a falar sobre o desenvolvimento da interface visual usada para demonstrar o uso dessa tecnologia em um tempo real.

3.1 FERRAMENTAS USADAS

Este trabalho foi desenvolvido em Python 3.9.6, utilizando um ambiente virtual miniconda3. Para a implementação da rede neural convolucional foi usada a biblioteca TensorFlow e a sua API Keras. O *dataset* utilizado foi o *dataset* de caracteres chineses *CharactersTrimPad28* (BURKIMSHER, 2018).

TensorFlow é uma biblioteca FOSS (Free and open-source software), gratuita e de código aberto, utilizada para aprendizado de máquina e inteligência artificial. Ela pode ser usada em uma variedade de tarefas, mas tem um foco particular no treinamento e inferência de redes neurais profundas.

Keras é uma API de aprendizado profundo escrita em Python, executada na plataforma de aprendizado de máquina TensorFlow. Ela foi desenvolvida para permitir a experimentação rápida, permitindo que mais iterações de testes sejam feitas, levando a melhores resultados.

O *dataset CharactersTrimPad28* foi desenvolvido por Peter Burkimsher em 2018. O *dataset* original consiste em 15 milhões de imagens PNG de caracteres chineses em

formato grayscale (ou escala cinza) com o tamanho de 28x28 pixels. Essas imagens são categorizadas em 52.835 classes, cada uma dessas representando um caractere chinês. O *dataset* completo possui 9.98 GB quando compactado e 13.48 GB quando descompactado.

Este trabalho utiliza dois modelos como base de comparação para o modelo desenvolvido. O primeiro é o modelo de referência usado no trabalho acadêmico “Deep Learning for Classical Japanese Literature” (CLANUWAT et al., 2018), que foi escrito por Tarin Clanuwat e a sua equipe em 2018, com o intuito de avaliar um *dataset* de kanjis chamado Kuzushiji-Kanji que eles construíram. Esse *dataset* foi uma das escolhas iniciais para este trabalho, mas foi dado como insuficiente, já que ele contém apenas imagens manuscritas de kanjis, enquanto este, tem foco nos caracteres em fontes de imprensa.

O segundo é um modelo de exemplo oferecido pelo site do keras para a construção de redes convolucionais em *datasets* no formato MNIST (KERAS, 2019).

3.2 AQUISIÇÃO E DESENVOLVIMENTO DO DATASET

A parte mais importante do treinamento de uma rede neural convolucional é o *dataset* usado para treinar a rede. Mesmo que uma rede tenha um modelo eficiente e bem construído, sem um *dataset* de qualidade ela nunca alcançará um patamar de performance e acurácia necessário para validar o seu uso em um ambiente externo.

O *dataset CharactersTrimPad28* foi escolhido por apresentar diversas vantagens específicas para este trabalho. Primeiramente, ele é bem extenso, permitindo que a rede seja treinada com uma enorme quantidade de dados, também contém caracteres de imprensa ao invés de cursivos como no *dataset* Kuzushiji-Kanji. Caracteres cursivos normalmente são mais desejados, já que muitas redes neurais são treinadas para reconhecer fotografias ou documentos, mas este trabalho visa a classificação de caracteres em imagens, vídeos e páginas de novelas gráficas digitais, mídias que normalmente usam fontes de imprensa, tornando esse *dataset* especialmente adequado para o trabalho.

O *dataset* foi construído usando cerca de 1260 fontes de caracteres chineses extraídos do site chinesefontdesign.com. Caracteres kanjis são derivados de caracteres chineses. Os dois sistemas de escrita são considerados distintos, principalmente devido às suas pronúncias diferentes e às mudanças dos significados de alguns caracteres, mas a forma de escrever os caracteres é idêntica entre os dois sistemas, então desde que o intérprete das classes da rede seja construído em japonês, os dados podem ser usados para o treinamento.

O próximo passo para a construção do *dataset* foi um processo de limpeza. Nesse processo são retirados das fontes os caracteres que não são kanjis, como os pontos, os acentos e os caracteres radicais. Em seguida, Burkimsher utiliza o sistema hb-view do programa harfbuzz, desenvolvido por Alexis King e Behdad Esfahbod, para gerar uma imagem para cada um dos caracteres na sua base de dados.

Com todas as imagens prontas, a última etapa do projeto é padronizá-las em um formato 28x28 grayscale. Esse formato foi escolhido porque ele é o padrão usado pelo *dataset* MNIST (LECUN et al., 1998), um dos *dataset* mais usados para o desenvolvimento e treinamento de redes.

Esse *dataset* contém 52.835 classes de caracteres, ou seja, 52.835 caracteres chineses diferentes, mas como esse trabalho é uma demonstração da tecnologia, foram usadas apenas 222 classe de caracteres, com um total de 246.741 imagens. Essas classes foram selecionadas utilizando os níveis de dificuldade dos kanji no teste de proficiência da língua japonesa (JLPT Japanese-Language Proficiency Test). Esse é um teste usado pelo governo japonês para categorizar o nível de proficiência de um indivíduo na língua. Os kanjis escolhidos para o *dataset* fazem parte das duas primeiras dificuldades.

Um aspecto do *dataset* que precisou ser mudado foi o tamanho das imagens. No *dataset* usado para treinar a rede deste trabalho as imagens são 36x36 ao invés do padrão MNIST de 28x28. Isso foi necessário para que os deslocamentos da imagem possam funcionar corretamente durante o processo de *data augmentation*. Isso foi feito através de um algoritmo em Python que adiciona 4 pixels brancos em cada um dos lados da imagem, atingindo assim o tamanho desejado.

O próximo passo do processo foi separar as imagens de cada um dos caracteres em três subconjuntos distintas. Um de treinamento, para treinar a rede neural, um de validação, que será usado para medir o avanço do treinamento e guiar o processo de retropropagação e uma subconjunto de teste que vai ser usado para testar a capacidade da rede de lidar com imagens que ela nunca viu. A divisão dos dados é a seguinte: 10% de todas as imagens de uma classe sendo colocadas na pasta de treinamento, com 10% do restante (9%) indo para a pasta de validação e o resto (81%) permanecendo na pasta de treino.

Mesmo com essas mudanças, os dados ainda não estão adequados para o treino, mas agora é possível aprimorá-los usando funções de *data augmentation*, buscando alcançar um melhor resultado para a rede.

3.3 DATA AUGMENTATION

Data augmentation, ou aprimoramento de dados, é uma série de técnicas usadas para aumentar a quantidade de dados em um grupo, adicionando cópias ligeiramente modificadas de dados já existentes, também chamados de dados sintéticos. Eles atuam como regularizadores e ajudam a reduzir o *overfitting* durante o treinamento da rede neural convolucional.

Este trabalho utiliza de alguns códigos pré-escritos da API Keras para o aprimoramento de dados, esses sendo a classe **ImageDataGenerator** e a sua função **flow_from_directory()**. A seguir encontra-se o código Python usado no processo de *Data augmentation*.

Algoritmo 3.4.3: Codigo de Data Augmentation.

```
1  def create_generators_kanji(batch_size, train_data_path):
2
3      train_preprocessor = ImageDataGenerator(
4          width_shift_range=0.2,
5          height_shift_range=0.2,
6          brightness_range=[0.8,1.2],
7          zoom_range=[0.7,1.3],
8          rescale= 1/255.,
9      )
10
11     train_generator = train_preprocessor.flow_from_directory(
12         train_data_path,
13         batch_size=batch_size,
14         class_mode="categorical",
15         target_size=(36,36),
16         color_mode='grayscale',
17         shuffle=True
18     )
19
20     return train_generator
```

Fonte: Autor

É importante notar que o código do algorítimo 3.4.3 é apenas para o gerador dos dados de treino, outros dois geradores foram escritos para os conjuntos de validação e de testes, mas como neste trabalho os três geradores foram configurados da mesma forma, para padronizar as imagens, não há a necessidade de mostrar o código completo. Esse código é dividido em duas partes. Primeiro, a instanciação do objeto **ImageDataGenerator**, linhas 3 a 9. Segundo, a execução da função **flow_from_directory()**.

Durante a instanciação de **ImageDataGenerator** , é necessário passar argumentos que indicam quais mudanças devem ser feitas nas imagens e como essas mudanças devem ser feitas. A lista abaixo indica a função dos argumentos da sua linha.

L 4. **“width_shift_range = 0.2”** É usado para definir o deslocamento da imagem no eixo horizontal, com o valor 0.2 indicando a intensidade deste deslocamento, neste caso 20%;

L 5. **“height_shift_range = 0.2”** Serve para uma função similar a linha anterior, desta vez deslocando a imagem no sentido horizontal. Essas duas alterações foram feitas para a rede identificar melhor as extremidades dos kanjis, assim como kanjis cortados na borda;

L 6. **“brightness_range = [0.8, 1.2]”** É usado para modificar o brilho das imagens, deixando elas mais claras ou mais escuras. A intensidade dessa modificação é decidida através dos dois valores passados para a variável, neste caso o Keras está deixando a imagem 20% mais escura (80% do seu brilho normal) ou 20% mais clara (120% do seu brilho normal). Essa configuração ajuda a rede a categorizar imagens com um plano de fundo de cor similar ao kanji presente nele;

L 7. **“zoom_range = [0.7, 1.3]”** Aplica uma distorção na imagem, esticando ela tanto horizontalmente quanto verticalmente. Uma qualidade boa desse parâmetro é que ele aplica essas mudanças separadamente, aumentando o número de permutações do sistema. Ele foi inserido para garantir que a rede consiga classificar kanjis mesmo quando eles estiverem distorcidos. Os valores utilizam a mesma lógica de porcentagem que o item anterior;

L 8. **“rescale= 1/255.”** Esse é um argumento especial. Ele é usado para adequar o tamanho dos dados das imagens para a taxa de aprendizado da rede. Como a taxa de aprendizado da rede é bem pequena essa mudança faz com que o aprendizado seja mais suave.

Após o objeto **ImageDataGenerator** ser instanciado e configurado, a função **flow_from_directory()** é usada para gerar o conjunto final de imagens que será usado na rede. Os parâmetros que essa função precisa receber são os seguintes:

L 12. **“train_data_path”** É o caminho para a pasta das imagens que devem ser aprimoradas. Esse é o primeiro parâmetro da função **create_generators_kanji()**;

L 13. **“batch_size”** É o tamanho do lote que vai ser usado no treinamento da rede. Esse é o segundo parâmetro da função **create_generators_kanji()**;

L 14. **“class_mode=’categorical’”** Esse parâmetro é usado para informar qual função de perda a interface Kerar deve preparar as imagens. Neste trabalho, a função de perda usada é uma chamada *categorical crossentropy*, e por causa disso esse parâmetro deve receber o valor “categorical”;

L 15. **“target_size = (36 ,36)”** Esse parâmetro é usado para garantir que todas as imagens permaneçam com tamanho 36x36, mesmo depois do processo de *Data augmentation*;

L 16. **“color_mode=’grayscale’”** Assim como o anterior, esse parâmetro é usado para garantir que as imagens permaneçam no modo de escala cinza após o processo de *Data augmentation*;

L 17. **“shuffle=True”** O parâmetro *shuffle* serve para embaralhar a ordem em que as imagens são apresentadas para a rede durante o treinamento. O propósito disso é reduzir *overfitting* e garantir que os modelos permaneçam gerais.

Agora, com o conjunto de dados completo, já é possível treinar redes usando eles. A única necessidade é que os modelos de arquitetura de rede tenham uma entrada compatível com o *dataset*, que neste caso seriam imagens 36x36 *grayscale*..

3.4 MODELOS DE REDE

Um modelo funciona como uma receita. Ele descreve quais ingredientes devem ser usados para construir uma rede neural e as suas camadas. Nele são especificados todos os aspectos da rede, como os padrões dos dados de entrada, a quantidade de camadas ocultas, qual o tipo de cada camada, suas configurações, como elas devem ser organizadas na rede e como devem ser os dados de saída da rede.

Um modelo original foi desenvolvido para este trabalho, visando melhorar a acurácia da rede de classificação de kanjis sendo desenvolvida. Outros dois modelos também foram treinados usando o mesmo *dataset* de kanjis deste trabalho, servindo como meios de comparação para o desempenho do modelo. Os códigos dos 3 modelos estão presentes neste trabalho e serão explicados de uma forma simples e descritiva, mas primeiro é importante explicar os tipos de camadas que serão usadas neste modelo e quais são suas finalidades.

Os modelos apresentados neste trabalho contém os seis tipos de camadas descritas a seguir:

a) **Camada convolucional de duas dimensões (Conv2D):** Essa camada é a parte central das redes convolucionais. Ela é responsável pelo processo de convolução, em que uma imagem é usada para gerar múltiplas imagens através de um peso global, também chamado kernel. As especificidades dessa camada são explicadas na seção 2.2.4 da página 24 deste trabalho;

b) **Camada de pooling máximo de duas dimensões (MaxPooling2D):** Essa é a camada complementar à camada convolucional. Elas são normalmente usadas após uma camada convolucional e tem o objetivo de condensar os vários blocos de dados sem perder informações importantes no processo;

c) **Camada nova de normalização de lotes (BatchNormalization):** A normalização em lote é uma técnica projetada para normalizar alguns neurônios da rede em cada lote de dados processado durante o treinamento. Isso tem o efeito de estabilizar o

processo de treinamento, o que pode acabar reduzindo drasticamente a quantidade de épocas necessárias para treinar uma rede profunda;

d) **Camada de achatamento (Flatten):** A função dessa camada é bem simples, ela reduz as dimensões atuais da rede para apenas uma, ou seja, a rede deixa de ser composta por conjuntos de imagens convolucionais e se torna uma lista de dados unidimensional. Isso é usado para transformar a rede convolucional em uma rede neural simples, um processo necessário para que a rede possa dar respostas compreensíveis;

e) **Camada de perda (Dropout):** Essa camada tem a função de mudar de forma aleatória os valores de alguns neurônios para 0, com a probabilidade deste processo dependendo dos argumentos passados para ela. Essa camada é muito usada para evitar *overfitting*;

f) **Camada densa (Dense):** A camada densa é a camada padrão de uma rede neural simples. Ela normalmente só é usada depois que uma camada Flatten tenha convertido a rede para uma rede simples, sendo usada como a camada final das redes, com o seu resultado servindo como a resposta da rede.

É importante notar que existem muitos outros tipos de camadas disponibilizadas na API Keras. As descritas acima são apenas as que foram utilizadas nos modelos apresentados neste trabalho.

3.4.1 Modelo Original

O modelo descrito foi desenvolvido para este trabalho, ele tem algumas características escolhidas para complementar o *dataset* em questão. A seguir encontra-se o código do algoritmo 3.4.3 que pode ser utilizado para criar, instanciar, treinar e avaliar o modelo. No programa de treinamento muitas dessas estampas são feitas em arquivos diferentes, mas para melhorar as explicações elas foram colocadas no mesmo arquivo. Abaixo do código encontra-se uma lista enumerada que explica as partes mais importantes de cada uma das linhas.

Algoritmo 3.4.3: Modelo Original.

```
def original_model():

    batch_size = 64
    epochs = 15

    train_generator, val_generator, test_generator =
       create_generators_kanji(
        batch_size,
        path_to_train = "./Dataset/Train",
        path_to_val = "./Dataset/Val",
        path_to_test = "./Dataset/Test"
        )
    numero_classes = train_generator.num_classes

    model = Sequential()

    model.add(Conv2D(128, kernel_size=(3,3), activation='relu',
       input_shape=(36, 36, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(BatchNormalization())

    model.add(Conv2D(128, (3,3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(BatchNormalization())

    model.add(Flatten())

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

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

```

```
31      model.fit(
32          train_generator,
33          epochs=epochs,
34          batch_size=batch_size,
35          validation_data=val_generator,
36      )
37
38      model.evaluate(test_generator)
```

Fonte: Autor

L 3. A primeira parte do código é uma das mais importantes. Nela são decididas as duas variáveis que controlam grande parte do processo de treinamento da rede. O tamanho do lote, ou *batch size* em inglês, é a quantidade de imagens que serão processadas pela rede para cada etapa de retropropagação, quanto menor o valor mais específico o treinamento se torna, porém, ele também demora mais tempo, já que o processo de retropropagação está sendo executado mais vezes. Esse modelo tem um tamanho de lote de 64 entradas, um pouco menor do que o padrão usado pelos outros modelos, mas um bom equilíbrio de velocidade e acurácia;

L 4. A segunda configuração que precisa ser feita é a quantidade de épocas (epochs) de treinamento da rede. Esse valor indica quantas vezes a rede vai processar todas as imagens de treinamento. Neste caso a rede vai processar todas as imagens 15 vezes antes de completar o treinamento.

É importante notar que, ao final da primeira época, a rede neural já vai estar em um estado usável, capaz de categorizar imagens de kanjis inseridas nela. Porém, não é recomendado usar a rede depois de apenas uma época, pois neste ponto, ela provavelmente está mal otimizada, fazendo com que erre a maioria das suas respostas. Múltiplas épocas permitem que a rede aprenda mais com o mesmo

conjunto de dados, mas se o treinamento tiver muitas épocas, a rede pode acabar sofrendo *overfitting*;

L 6. O próximo passo do processo é a criação dos conjuntos de dados que vão ser usados para treinar, validar e testar os modelos. Para isso será usada a função de **create_generators_kanji()** de *data augmentation*, explicada na seção anterior.

Essa função recebe quatro parâmetros e retorna os três conjuntos de dados. O primeiro é o tamanho do lote escolhido na linha 3, enquanto os outros parâmetros são os caminhos para as pastas que contêm os grupos de treinamento, validação e teste, respectivamente;

L 12. Com os grupos de dados criados, o código salva o número de classes de kanji que o modelo vai ter. É importante salvar esse valor porque ele é usado em algumas outras partes do código;

L 14. Essa é parte mais importante do código. Nela é criado um objeto chamado “model”. Esse “model” é a rede neural e cada uma das linhas abaixo que contém o termo **model.add()** representam uma nova camada sendo adicionada à rede. Muitas dessas camadas têm configurações próprias que serão explicadas, mas agora é importante perceber que o objeto “model” está recebendo de uma função **Sequencial** Isso indica que o tipo de rede sendo construída é uma rede sequencial, ou seja, as camadas dessa rede só podem passar informações para a camada da frente sem nunca retroceder para a camada de anterior;

L 15. A primeira camada da rede, é uma camada convolucional de duas dimensões. Ela aplica o processo de convolução, gerando 128 novas imagens de tamanho 34x34 para a próxima camada. Essa camada possui os seguintes argumentos:

L 15.1. **”32”** Esse argumento indica a quantidade de neurônios desta camada. É por causa desse número que 128 novas imagens são geradas;

L 15.2. **“kernel_size=(3, 3)”** Indica o tamanho dos pesos do processo de convolução.

É esse valor que faz com que as imagens resultantes sejam do tamanho 34x34, pois cada conjunto possível de 9 pixels adjacentes se torna um novo pixel na imagem resultante;

L 15.3. **“activation=’relu’”** Representa a função de ativação usada pelos neurônios desta camada, a função ‘relu’. Muito usada em redes neurais já que ela é quase binária e sempre resulta em muito proximo de 1 quando ativada e de 0 quando desativada;

L 15.4. **“input_shape=(36, 36, 1)”** Esse é um argumento único, usado apenas na primeira camada da rede. Ele indica o padrão dos dados de entrada da rede. Essa rede utiliza imagens 36x36 em escala cinza, com cada imagem tendo 36 pixels de altura, 36 pixels de largura e 1 pixel de profundidade usado para representar o valor da escala cinza.

L 16. A próxima camada é uma camada de MaxPooling2D. Ela contém apenas um atributo, **“pool_size=(2, 2)”**, que indica para a rede que cada conjunto 2x2 pixel deve ser reduzido para 1 pixel no processo de pooling, efetivamente fazendo com que a nova imagem tenha um quarto do seu tamanho original. Isso reduz a quantidade de parâmetros de treinamento na rede, tornando a sua execução mais rápida;

L 17. A seguir uma camada nova de normalização de lotes é usada para padronizar os dados após o processo convolucional;

L 18. Com o primeiro processo convolucional completo, é configurado no modelo um segundo processo convolucional, repetindo as últimas três camadas na mesma ordem, com as mesmas configurações. O objetivo é permitir que a rede aprenda a encontrar mais características visuais nos kanjis através do segundo processo convolucional;

L 21. Após o segundo processo convolucional, a rede passa pelo processo de achatamento, com a camada *Flatten*, sendo convertida para uma rede neural simples;

L 22. Com a rede simplificada, duas camadas densas são adicionadas para finalizar o processo de treinamento. A primeira delas é construída com 128 neurônios e, assim como as Conv2D, utiliza a função de ativação ‘relu’;

L 23. A segunda função densa é especial. Ela é a última classe desse modelo, a camada de saída, e por causa disso, seus dois parâmetros diferem dos parâmetros das outras camadas. O primeiro deles informa que a quantidade de neurônios deve ser igual à quantidade de classes do grupo de treinamento, para que a rede consiga responder de forma sucinta. O segundo argumento indica a função matemática que deve ser usada para mostrar esse resultado, enquanto a ‘relu’ é mais usada para camadas ocultas. A função ‘softmax’ costuma ser escolhida para camadas de saída, devido aos seus valores serem melhores para a visualização dos resultados.

Com isso, o modelo da rede foi construído. O próximo passo agora é compilar o programa para fazer o treinamento e por fim avaliar os seus resultados;

L 24. Para compilar o modelo a função **.compile()** é chamada com os parâmetros a seguir:

L 24.a) **“optimizer = ‘adam”** representa o método de otimização escolhido para a compilação da rede. A otimização Adam é uma extensão para a descida de gradiente estocástica Que vem tendo uma adoção mais ampla para aplicativos de aprendizado profundo em visão computacional. Essa configuração foi a única mudança feita ao modelo Kuzushiji, por demonstrar um aumento significativo de acurácia quando treinado pela máquina utilizada neste trabalho;

L 24.b) **“loss = ’categorical_crossentropy’”** é a função de perda. Essa precisa ser a mesma que foi usada na preparação dos dados durante o processo de *Data augmentation*;

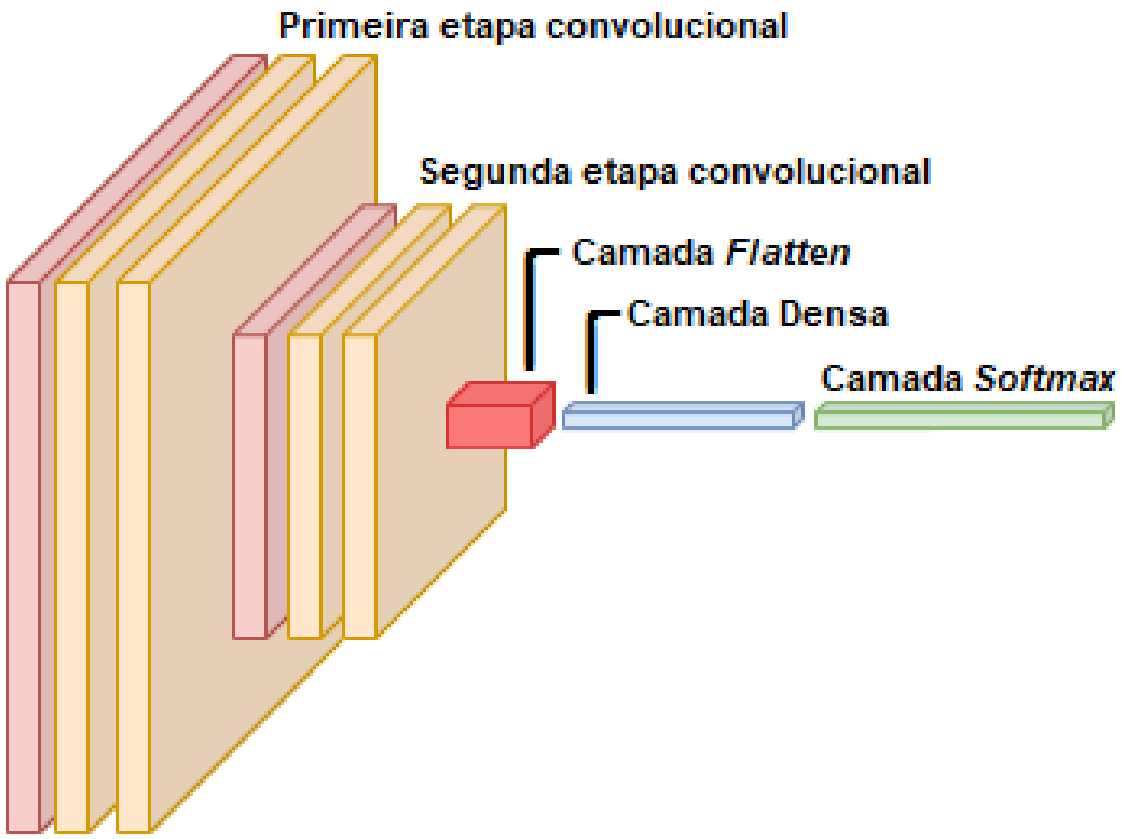
L 24.c) **“metrics=[’accuracy’]”** indica que a rede deve ser treinada, visando melhorar a sua acurácia (accuracy).

L 25. Com o modelo compilado, a função de treinamento **.fit()** pode ser executada. Ela recebe como argumentos o conjunto de treinamento, o número de épocas que ele deve treinar, o tamanho dos seus lotes e o grupo de imagens que deve ser usado para validação;

L 26. Após a função de treinamento, a rede é avaliada utilizando o *dataset* de teste, composto de imagens que nunca foram processadas pela rede. Isso é feito para garantir que a rede consiga funcionar em novos ambientes de trabalho, mesmo que ela não tenha sido treinada especialmente para eles.

Esse é o modelo usado na interface de usuário para classificar kanjis, mas é importante que ele não seja avaliado de uma forma isolada, por isso outros dois modelos foram implementados e treinados utilizando o mesmo *dataset*, na mesma máquina.

Figura 10 – Modelo Original



Fonte: Autor

A figura 8 apresenta uma apresentação gráfica do modelo original. Ela mostra às duas seções execuções do processo convolucional, a camada *flatten* e as camadas densas do trabalho.

3.4.2 Modelo Kuzushiji

Esse modelo foi desenvolvido para testar o *dataset* do artigo acadêmico “Deep Learning for Classical Japanese Literature” (CLANUWAT et al., 2018). O foco desse artigo é o *dataset*. Ele acaba não sendo tão poderoso como outros no mercado, mas independentemente disso ele é muito útil para comparações com o modelo deste artigo, pois ele também foi construído visando a classificar kanjis

Algoritmo 3.4.3: Modelo Kuzushiji.

```
1  def kuzushiji_model():
2
3      batch_size = 128
4      epochs = 12
5
6      train_generator, val_generator, test_generator =
          create_generators_kanji(
7          batch_size,
8          path_to_train = "./Dataset/Train",
9          path_to_val = "./Dataset/Val",
10         path_to_test = "./Dataset/Test"
11         )
12     numero_classes = train_generator.num_classes
13
14     model = Sequential()
15     model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
          input_shape=(36, 36, 1)))
16     model.add(Conv2D(64, (3, 3), activation='relu'))
17     model.add(MaxPooling2D(pool_size=(2, 2)))
18     model.add(Dropout(0.25))
19     model.add(Flatten())
```

```
20      model.add(Dense(128, activation='relu'))
21      model.add(Dropout(0.5))
22      model.add(Dense(numero_classes, activation='softmax'))
23
24      model.compile(optimizer='adadelta', loss='
            categorical_crossentropy', metrics=['accuracy'])
25
26      model.fit(
27          train_generator,
28          epochs=epochs,
29          batch_size=batch_size,
30          validation_data=val_generator,
31      )
32
33      model.evaluate(test_generator)
```

Fonte: (LECUN et al., 1998)

Algoritmo 3.4.3 apresenta o código Python usado para criar, treinar e avaliar este modelo. Muitas linhas deste código são iguais às linhas do código anterior e por isso suas explicações serão mais diretas, focando nas diferenças entre os modelos.

A primeira diferença do modelo Kuzushiji são os valores escolhidos para suas variáveis de treinamento, com tamanho de lote sendo 128 imagens ao invés de 64, e o número de épocas de treinamento sendo 12 ao invés de 15. Essas mudanças aceleram o processo de treinamento, mas aumentam a generalização dos dados durante a retropropagação.

Após configurar os geradores de dados na linha 6 e o número de classes na linha 12, o modelo começa a ser construído na linha 14. Ele é do tipo sequencial e sua primeira camada recebe o argumento de configuração dos dados de entradas. A sua última camada, linha 21, é idêntica à do modelo anterior, com o número de neurônios sendo baseado no número de classes e sua função de ativação sendo a ‘softmax’.

Mas é aqui que as semelhanças acabam, com as camadas ocultas tendo uma organização bem diferente do modelo original. A primeira diferença é o tratamento das camadas convolucionais, com esse modelo tendo apenas um processo convolucional que passa por duas classes Conv2D antes da etapa de *pooling*. vale notar que a quantidade de neurônios é incremental, aumentando ao decorrer do modelo.

A segunda diferença é a falta de camadas de normalização de lotes. Elas foram adicionadas no modelo original para auxiliar no processo de treinamento da rede, mas como esse modelo foi construído para um *dataset* com menos imagens, ele não precisa dessas camadas.

A terceira diferença presente neste modelo é a inclusão de camadas de perda. Duas foram incluídas, uma depois do processo convolucional e outra antes da camada de saída. Como explicado anteriormente esse processo é muito usado para evitar *overfitting*.

A última diferença é o otimizador usado para compilar a rede. Diferente dos outros dois modelos mostrados neste trabalho, que utilizam a função adam, o modelo Kuzushiji utiliza a função de otimização 'adadelta'. A otimização 'adadelta' é um método de descida de gradiente estocástico que se baseia em uma taxa de aprendizagem adaptativa, tentando resolver a decadência contínua das taxas de aprendizado ao longo do treinamento e a necessidade de uma taxa de aprendizado global ser selecionada manualmente. Ela é mais robusta que a função adam, porém os seus pontos fracos são a sua escalabilidade e o seu uso de memória, dois aspectos importantes para este trabalho em que o método 'adam' é superior.

3.4.3 Modelo Keras

A seguir encontra-se o código do algoritmo 3.4.3. Ele foi utilizado para construção, treinamento e avaliação do modelo:

Algoritmo 3.4.3: Modelo Keras.

```
1 def benchmark_model():
```

```
2     batch_size = 128
3     epochs = 15
4
5     train_generator, val_generator, test_generator =
          create_generators_kanji(
6          batch_size,
7          path_to_train = "./Dataset/Train",
8          path_to_val = "./Dataset/Val",
9          path_to_test = "./Dataset/Test"
10         )
11    numero_classes = train_generator.num_classes
12
13    model = Sequential()
14    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
          input_shape=(36, 36, 1)))
15    model.add(MaxPooling2D(pool_size=(2, 2)))
16    model.add(Conv2D(64, kernel_size=(3, 3), activation="relu"))
17    model.add(MaxPooling2D(pool_size=(2, 2)))
18    model.add(Flatten())
19    model.add(Dropout(0.5))
20    model.add(Dense(numero_classes, activation="softmax"))
21
22    model.compile(optimizer='adam', loss='categorical_crossentropy'
          , metrics=['accuracy'])
23
24    model.fit(
25         train_generator,
26         epochs=epochs,
27         batch_size=batch_size,
28         validation_data=val_generator,
29    )
30
31    model.evaluate(test_generator)
```

Fonte: (KERAS, 2019)

Este é o último modelo e o mais simples dos três, recomendado pelo site do Keras como um modelo para iniciantes aprenderem a usar o *dataset* MNIST. Ele é bem-parecido com o modelo original contendo dois processos convolucionais antes de ser achatado, mas ele consegue se diferenciar do primeiro modelo em quatro aspectos.

a) Ele tem um tamanho de lote de 128, assim como o modelo Kuzushiji;

b) Suas camadas convolucionais seguem o padrão incremental de neurônios, começando com 32 e indo para 64;

c) A camada de normalização de lotes não está presente no modelo;

d) A penúltima camada não é uma camada densa, mas uma camada de perda com uma taxa de 50%.

3.5 CONSTRUÇÃO DA INTERFACE DE USUÁRIO

Para demonstrar que a classificação de kanjis pode ser usada para ajudar estudantes de japonês querendo aprender kanji, foi desenvolvida uma interface visual simples em Python usando a biblioteca Tkinter.

Essa interface tem três funcionalidades: adquirir uma imagem da tela do usuário e modificar essa imagem para ela poder ser processada pela rede neural, receber a sua resposta, e mostrar para o usuário o kanji que a rede identificou na imagem de forma simples e fácil de entender, com informações extras sobre o kanji detectado para que o usuário possa aprender mais sobre o caractere e as palavras que o utilizam. A janela da interface encontra-se na figura 9:

Figura 11 – Aplicativo de Interface Grafica

Fonte: Autor

Para capturar a imagem da tela o programa usa uma função do Windows chamada *screenclip*, que permite capturar uma imagem em qualquer parte da tela e salvar essa imagem no clipboard do Windows, onde ela pode ser facilmente acessada pelo Python. A seguir a imagem é convertida em imagem de tamanho 36x36 e escala cinza, para ser compatível com a rede. Kanjis, por serem caracteres de uma língua, são normalmente encontrados com apenas uma cor em planos de fundo de alto contraste, fazendo com que a imagem continue identificável, mesmo após ter sido simplificada.

A imagem é então passada para a rede, com a rede respondendo qual o número da classe que ela acredita estar presente na imagem. Isso acaba causando um problema, como foi explicado anteriormente. A rede não sabe o que é um kanji, ela só sabe que a imagem inserida se parece mais com as imagens de uma pasta do que as imagens das outras pastas. Por exemplo, se uma imagem com o kanji de “árvore” for inserida na rede, assumindo que a classificação esteja correta, a resposta não vai ser “árvore”, ela vai ser “114”. Isso porque “144” é o número da classe que contem os kanjis de “árvore”.

Por causa disso é necessário a criação de um intérprete que consiga transformar o valor da classe em um kanji que possa ser mostrado.

Para resolver esse problema, uma tabela foi criada para ser o intérprete da rede. Ela contém todos os 222 kanjis que foram usados para treinar a rede e as suas classes correspondentes. Desta forma quando a rede responder o número da classe da imagem, o programa pode olhar na tabela e identificar qual a resposta deve dar para o usuário.

Por último a interface de usuário tem algumas funções extras para ajudar estudantes aprendendo japonês: ela não responde apenas o kanji detectado, mas também o seu significado tanto em português quanto inglês. O programa também indica as pronúncias corretas do kanji detectado utilizando caracteres romanos e caracteres kana de origem japonesa, e por fim o programa dá ao usuário a opção de abrir a página no kanji detectado na jisho.org, um dos dicionários japoneses mais completos do mundo.

3.6 CONSIDERAÇÕES SOBRE O CAPÍTULO

Esse capítulo abordou o desenvolvimento deste trabalho. Ele explica como o *dataset* foi alterado, como os modelos de rede foram estruturados e como a interface gráfica foi construída. O próximo capítulo vai abordar os resultados deste trabalho e as comparações entre os modelos apresentados neste capítulo.

4 RESULTADOS

Neste trabalho foram apresentados três modelos de redes neurais convolucionais profundas para classificação de kanjis, um desenvolvido para o trabalho, e dois modelos de comparação. O primeiro modelo de comparação foi desenvolvido para testar um *dataset* de caracteres japoneses manuscritos, e o outro é exemplo padrão dado pelo site do Keras como exemplo de um modelo para *datasets* MNIST. Este capítulo vai mostrar os resultados desses modelos e as conclusões derivadas desses valores

Os três modelos foram treinados na versão modificada do *dataset CharactersTrimPad28* usada neste trabalho e os resultados se encontram abaixo:

Tabela 1 – Resultados comparativos dos três modelos construídos

Modelos	**Original**	**Kuzushiji**	**Keras**
Parâmeros	981.470	2.144.734	715.230
Val - Loss	0.3377	0.8247	0.5648
Val - Accuracy	0.9180	0.8284	0.8811
Test - Loss	0.3348	0.8107	0.5489
Test - Accuracy	0.9178	0.8314	0.8829
Tempo de Resposta (Segundos)	0.032522s	0.032825s	0.032525s

A primeira linha da tabela, desconsiderando os modelos, representa a quantidade de parâmetros que a rede usa para classificar as imagens. O modelo deste trabalho mostra um bom desempenho neste aspecto. Mesmo com as suas camadas convolucionais tendo quatro vezes mais neurônios que o modelo Keras, a sua quantidade de parâmetros se mantêm baixa, e isso é bom, porque quanto menos parâmetros uma rede tem, mais rápido fica o seu processo de classificação.

A abundância de parâmetros no modelo kuzushiji vem das suas duas camadas convolucionais consecutivas, enquanto essa estratégia pode resultar em uma acurácia alta, ela também aumenta o número de parâmetros que a rede precisa treinar.

Os próximos valores representam a perda (Loss) e a acurácia (Accuracy) que os modelos tiveram quando foram avaliados com o conjunto de validação (Val) e com o conjunto de teste (Test).

O valor de acurácia é o mais simples de entender. Ele representa a porcentagem de acerto que a rede teve em todas as imagens do conjunto. Por exemplo, se um conjunto de teste tiver 1000 imagens e a rede conseguir acertar o kanji de 909 imagens, essa rede teria um valor de validação de 0.9090.

O valor de perda, por outro lado, é mais complicado, pois ele não representa uma porcentagem como a acurácia. Ele é a somatória de todos os erros de uma rede. Isso pode parecer contraintuitivo, mas alguns erros são piores que outros, e quanto mais incorreta for a resposta, maior vai ser o valor somado ao número de perda. É importante notar que uma rede com acurácia alta normalmente vai ter um valor de perda baixo, pois ela teve menos erros sendo somados ao valor.

Os últimos valores representam o tempo médio que cada modelo leva para fazer uma classificação. Esses números foram calculados a partir de 100 imagens de classificação rodando em um computador com as seguintes especificações: processador Intel Core i3 8100, placa de video Nvidia GTX 1060 3GB e 8GB de memória ran DDR4. O computador também tem a versão 8101 da ferramenta cuDNN instalada. É importante mencionar esses requisitos porque os resultados mostrados na tabela possuem uma diferença muito pequena, mas isso é devido à força do computador, em máquinas mais fracas é esperado que as diferenças apresentadas tenham um impacto maior.

Como esperado, o modelo kuzushiji teve a pior performasse, a quantidade extra de parâmetros acaba aumentando o tempo de execução. O modelo original deve um desempenho melhor em comparação. O tempo de classificação do modelo original acabou sendo menor que até mesmo que o modelo Keras.

Outro aspecto importante que deve ser considerado nessa comparação é que o modelo kuzushiji está sendo usado em um ambiente diferente do ambiente que ele foi criado para testar, o que pode resultar em uma queda de performance. Quando seus resultados neste trabalho são comparados com os seus resultados no *dataset* Kuzushiji-

MNIST, um dos *datasets* que este modelo foi desenvolvido para testar, uma redução notável de performance pode ser vista, com a sua acurácia caindo de 94.63% para 83.14%.

Os resultados do modelo original, por outro lado, são promissores, mostrando um bom desempenho nos testes e uma boa velocidade durante sua execução na interface de usuário. O que é um forte argumento a favor do uso de redes neurais para a classificação de kanji durante o aprendizado.

Vale também falar sobre a interface visual, devido ao baixo número de parâmetros do modelo desenvolvido, a identificação de kanjis em um ambiente de trabalho é rápida e eficiente. O uso prático da classificação de kanjis acabou se mostrando mais precisa do que o esperado, com ela acertando 95.3% dos kanjis testados. Isso acontece porque a maioria dos kanjis que um usuário se depara durante seu estudo estão em fontes extremamente simples, melhorando a chance deles serem identificados pela rede.

Todos os arquivos desenvolvidos para este trabalho estão disponíveis no repositório público: github.com/ArthurFGrillo/KanjiDetectionNetwork. Incluindo os modelos já treinados e o aplicativo de interface gráfica.

4.1 CONSIDERAÇÕES SOBRE O CAPÍTULO

Esse capítulo abordou os resultados do modelo de rede neural desenvolvido para este trabalho, comparando eles com outros dois modelos implementados. Ele também apresenta interpretações dos dados e como esses resultados influenciam a aplicabilidade da interface gráfica.

5 CONCLUSÃO

O objetivo deste trabalho foi desenvolver uma rede neural convolucional profunda para a classificação de caracteres kanji, que demonstre a viabilidade dessa tecnologia, usada para auxiliar o aprendizado da língua japonesa.

Para isso, foi necessário a construção de uma *dataset* de treino com 222 classes e 246.741 imagens de caracteres kanji, o desenvolvimento de um modelo de rede convolucional que consiga alcançar um alto nível de acurácia, sem grande perda de performance e uma interface gráfica que possa ser usada para demonstrar a capacidade dessa rede de ser utilizada em um ambiente real.

O modelo desenvolvido mostrou excelentes resultados, mesmo quando comparado a outros modelos treinados no mesmo *dataset*, e o seu uso na interface gráfica se mostrou promissor, conseguindo identificar grande parte dos caracteres apresentados a ela durante os testes.

Isso não significa que este trabalho é perfeito, ele certamente não é, mas ele conseguiu alcançar os seus objetivos e, no futuro, pode servir como um excelente argumento a favor do uso de redes neurais no aprendizado linguístico, assim como um primeiro passo para outros que tentam levar os conceitos apresentados neste trabalho adiante.

Existem muitas formas de melhorar o sistema desenvolvido neste trabalho que podem ser usadas como base para trabalhos futuros, a principal delas sendo a criação de um *dataset* mais completo. Mesmo que 222 seja um número bem maior que o de outros *datasets* MNIST, ele não se aproxima dos 5000 kanjis necessários para atingir fluência na língua japonesa, fazendo com que essa seja uma excelente área para continuar o desenvolvimento.

Outra forma de melhorar a rede seria a criação de um modelo mais preciso. Um valor de acurácia de 91,78% não é ruim para essa demonstração. Ele não é um número aceitável se essa rede fosse ser disponibilizada para o público, e o mesmo se aplica

à interface de usuário, especialmente porque a linguagem Python não é apropriada para esse tipo de programa. Uma boa estratégia seria a construção de um sistema *backend* Python para a rede neural com o sistema *frontend* sendo construído em uma linguagem mais adequada para interfaces gráficas.

REFERÊNCIAS

AFROGE, S.; AHMED, B.; MAHMUD, F. Optical character recognition using back propagation neural network. In: IEEE. *2016 2nd International Conference on Electrical, Computer & Telecommunication Engineering (ICECTE)*. [S.l.], 2016. p. 1–4.

BURKIMSHER, P. *Making of a Chinese Characters dataset*. 2018. <https://medium.com/@peterburkimsher/making-of-a-chinese-characters-dataset-92d4065cc7cc>. [Online; accessed 6-february-2022].

CIREŞAN, D.; MEIER, U. Multi-column deep neural networks for offline handwritten chinese character classification. In: IEEE. *2015 international joint conference on neural networks (IJCNN)*. [S.l.], 2015. p. 1–6.

CLANUWAT, T. et al. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.

Data Science Academy. *Deep Learning Book*. 2022. <http://www.deeplearningbook.com.br/>. Accessed: 2022-2-014.

FUKUSHIMA, K. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, Elsevier, v. 1, n. 2, p. 119–130, 1988.

HOWARD, A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

ISLAM, N.; ISLAM, Z.; NOOR, N. A survey on optical character recognition system. *arXiv preprint arXiv:1710.05703*, 2017.

KanjiTomo Team. *KanjiTomo*. 2012. <https://www.kanjitomo.net/>. Accessed: 2022-2-014.

KERAS. *Simple MNIST convnet*. 2019. <https://keras.io/examples/vision/mnist_convnet/>. [Online; accessed 6-february-2022].

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, v. 25, p. 1097–1105, 2012.

LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Ieee, v. 86, n. 11, p. 2278–2324, 1998.

LY, N.-T. et al. Deep convolutional recurrent network for segmentation-free offline handwritten japanese text recognition. In: IEEE. *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. [S.l.], 2017. v. 7, p. 5–9.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943.

MITHE, R.; INDALKAR, S.; DIVEKAR, N. Optical character recognition. *International journal of recent technology and engineering (IJRTE)*, Citeseer, v. 2, n. 1, p. 72–75, 2013.

MOROHASHI, T. *Dai kan-wa jiten*. [S.l.]: Taishūkan shoten, 1960.

O'SHEA, K.; NASH, R. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

PTUCHA, R. et al. Intelligent character recognition using fully convolutional neural networks. *Pattern recognition*, Elsevier, v. 88, p. 604–613, 2019.

RAJAVELU, A.; MUSAVI, M. T.; SHIRVAIKAR, M. V. A neural network approach to character recognition. *Neural networks*, Elsevier, v. 2, n. 5, p. 387–393, 1989.

RAKUTEN. *Survey on attitudes towards English*. 2016. <https://global.rakuten.com/corp/news/press/2016/0826_01.html>. [Online; accessed 14-february-2022].

ROSENBLATT, F. *The perceptron, a perceiving and recognizing automaton Project Para*. [S.l.]: Cornell Aeronautical Laboratory, 1957.

SIMARD, P. Y. et al. Best practices for convolutional neural networks applied to visual document analysis. In: *Icdar*. [S.l.: s.n.], 2003. v. 3, n. 2003.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

SOANES, C.; STEVENSON, A. *Concise oxford English dictionary*. [S.l.]: Oxford University Press Oxford, 2004. v. 11.

WAIBEL, A. et al. Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, IEEE, v. 37, n. 3, p. 328–339, 1989.

WANG, T. et al. End-to-end text recognition with convolutional neural networks. In: IEEE. *Proceedings of the 21st international conference on pattern recognition (ICPR2012)*. [S.l.], 2012. p. 3304–3308.

WU, C. et al. Handwritten character recognition by alternately trained relaxation convolutional neural network. In: IEEE. *2014 14th International Conference on Frontiers in Handwriting Recognition*. [S.l.], 2014. p. 291–296.

XU, L.; NAGAYOSHI, H.; SAKO, H. Kanji character detection from complex real scene images based on character properties. In: IEEE. *2008 The Eighth IAPR International Workshop on Document Analysis Systems*. [S.l.], 2008. p. 278–285.

ZHANG, X.-Y. et al. Drawing and recognizing chinese characters with recurrent neural network. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 40, n. 4, p. 849–862, 2017.