Deep Learning: Redes Neurais Convolucionais [1]

Prof. Dr. Giancarlo D. Salton

outubro de 2023

[1] Universidade Federal da Fronteira Sul
Campus Chapecó
gian@uffs.edu.br

As redes neurais artificiais (RNAs) conseguem realizar diversas tarefas de *machine learning* que envolvem classificação ou regressão com resultados bastante satisfatórios. Mesmo redes *feedforward* com camadas totalmente conectadas (*i.e.*do tipo "densas") conseguem aprender quaisquer funções contínuas[2] dentro de um limite acurácia, desde que possuam o número correto de neurônios.

[2] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2: 359–366, 1989

No entanto, existem algumas desvantagens na utilização de arquiteturas de RNA *feedforward* com camadas totalmente conectadas quando processamos imagens. Considere o exemplo da Figura 1. Para inserir a imagem na rede, precisamos selecionar cada pixel da imagem e colocar seu valor no neurônio correspondente ao índice deste pixel. Perceba que precisamos transformar a imagem num vetor unidimensional ($1D$), deixando de lado o formato de matriz bidimensional ($2D$) da imagem.

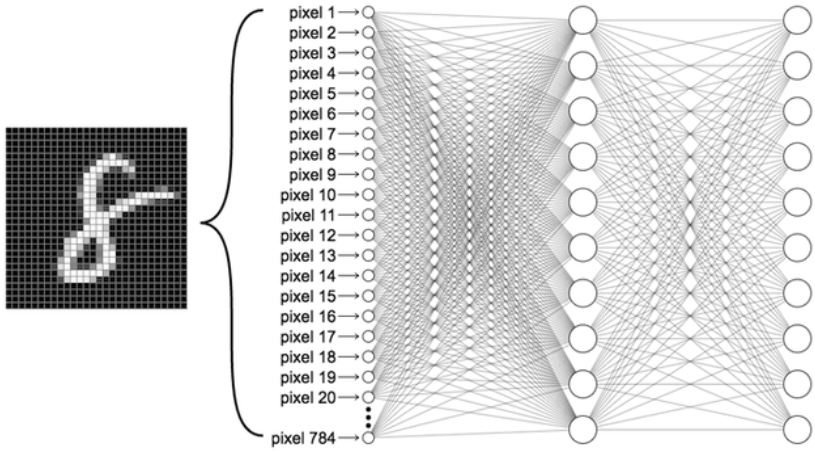

Figura 1: Exemplo de como uma imagem do dataset MNIST, representada em tons de cinza, é inserida em uma RNA de camadas totalmente conectadas.

Ao analisarmos mais de perto a disposição de cada pixel da imagem, como demonstrado na Figura 2, fica claro que muitos dos valores > 0 ficam distantes uns dos outros quando tornamos a imagem $1D$ para inserção no modelo. Isto faz com que a RNA tenha uma dificuldade maior para aprender a relação entre cada pixel da imagem[3] e, portanto, tenha uma capacidade sub-ótima.

[3] Desconsidere o fato de que mesmo uma RNA *feedforward* com camadas totalmente conectadas conseguem acurácias acima de 96% neste dataset e tenta imaginar imagens mais complexas do que digitos em escalas de ciza.

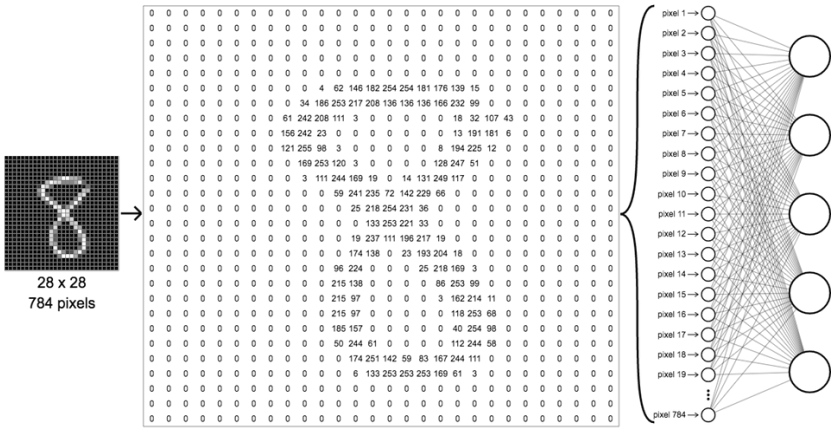

Figura 2: Exemplo da estrutura de pixels de uma imagem do dataset MNST. Como a imagem está em representada em tons de cinza, os valores dos pixels variam no intervalo $[0, 1]$, onde 0 representa um pixel preto e 1 presenta pixel branco. Valores dentro do intervalo variam em tons de cinza.

Redes Neurais com Camadas de Convolução

Redes Neurais Convolucionais (CNNs), ou Convolutional Neural Networks (CNNs ou ConvNets) em inglês, são uma classe especializada de redes neurais profundas projetadas para processar e analisar dados que possuem uma grade de estrutura espacial, como imagens. Elas foram introduzidas para resolver desafios específicos relacionados ao processamento de informações bidimensionais, explorando a natureza local e hierárquica dos dados visuais. A arquitetura das CNNs é composta por camadas específicas que desempenham papéis cruciais no processo de aprendizado. Abaixo, analisaremos as principais camadas que compõem a arquitetura das CNNs.

Convolução

A operação de convolução é um dos elementos fundamentais nas camadas cas CNNs. O processo de convolução é inspirado no processo de visão dos mamíferos[4] e nas técnicas de processamento de imagens tradicionais, que se utilizam de filtros (também chamados de *kernel*) para modificar imagens digitais[5].

Neste caso, um filtro ou *kernel* é uma pequena matriz bidimensional de neurônios artificiais que são aplicados sobre as imagens ou a saída de uma camada convolucional anterior. O filtro é deslizado sobre a entrada, sempre da esquerda para direita, de cima para baixo. O processo inicia no ponto mais a esquerda e encerra no ponto mais a direita. Ao chegar ao final da linha, no ponto mais a direita, retorna para o ponto mais a esquerda (início da linha) e reinicia 1 ou mais pixels para baixo. Em cada posição, os elementos correspondentes na região atual da entrada são inseridos nos neurônios do filtro e suas

[4] David H. Hubel and Torsten N. Wiesel. Effects of monocular deprivation in kittens. *Naunyn-Schmiedebergs Archiv for Experimentelle Pathologie und Pharmakologie*, 248:492–497, 1964

[5] Alejandro Dominguez. A history of the convolution operation. *IEE Pulse*, 2015. URL `https://www.embs.org/pulse/articles/history-convolution-operation/`

saídas formam os *feature maps*[6], conforme veremos na Seção . Este processo de posicionar um filtro sobre a imagem de entrada está representado na Figura 3.

[6] Também chamados de "mapas de ativação" ou "mapas de *features*"

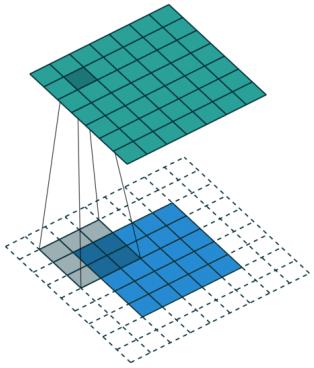

Figura 3: Exemplo de aplicação de filtro de convolução em uma imagem. A imagem está representada pela matriz mais escuro (azul) na parte inferior e o filtro de neurônios está representado pela matriz sombreada (cinza) menor, na parte inferior da imagem. O *feature map* de saída está representado pela matriz na parte superior da imagem (verde), em destaque com relação às demais. Os quadrados sem cor (brancos) representam o *padding da imagem.*

A fórmula matemática para a convolução de um filtro W com uma entrada X em uma posição (i, j) pode ser representada como:

$$(W \star x)_{i,j} = \sum_m \sum_n W_{m,n} \cdot x_{i+m,j+n} \tag{1}$$

onde: (i, j) é a posição do *feature map* saída; (m, n) representa as coordenadas no filtro; e $\star$ indica o operador de convolução. Esta mesma equação pode ser reescrita em formato de matrizes:

$$\mathbf{b}_{C_{out_j}} + \sum_{k=0}^{C_{in}-1} \mathbf{W}_{C_{out_j},k} \star \mathbf{x}_{N_i,k} \tag{2}$$

onde: $\star$ é o operador de convolução.

O processo de convolução de um filtro em uma imagem, comporta-se, visualmente, como demonstrado na Figura 4. A operação de convolução é repetida para vários filtros e posições dos dados de entrada, gerando múltiplos *feature maps* como saída. Esses mapas representam a ativação de diferentes padrões na entrada para detectar características específicas, como bordas, texturas ou formas, e são usados como entrada para as camadas subsequentes da rede. Desta forma, todos os filtros aplicados em um mesmo momento para sobre a entrada compõem uma camada da ConvNet.

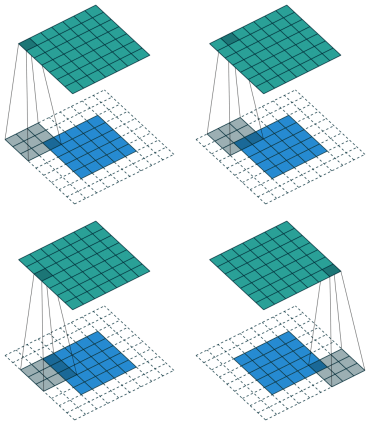

Figura 4: Processo de convolução em uma imagem com *padding* e *stride* $(1, 1)$. Na primeira linha temos os dois passos iniciais da convolução. Na segunda linha temos um passo intermediário e o passo final. Perceba que sempre há pelo menos um neurônio sobreposto à imagem.

Durante o processo de convolução, é necessário definir como o filtro se move sobre os dados de entrada, isto é, durante a convolução, o filtro se move em passos determinados pelo *stride*. O *stride* define a quantidade de pixels que o filtro avança em cada etapa. Por exemplo, se definirmos um *stride* de 2, cada vez que o filtro se mover para

esquerda ou para baixo, irá se deslocar 2 pixels para o lado em que se mover. No entanto, é mais comum definirmos o *stride* como uma tupla, *e.g.* $(2, 1)$, onde o primeiro índice define o deslocamento para esquerda e o segundo índice define o deslocamento para baixo. Neste exemplo, o deslocamento ocorre 2 pixels para esquerda e 1 para baixo. Um *stride* maior resulta em *feature maps* menores devido ao fato de que cada filtro gera apenas uma saída e, portanto, quando o filtro "pula" um número mais de pixels, acaba gerando um número menor de saídas. Por outro lado, um *stride* menor aumenta a resolução do mapa, mas pode aumentar a carga computacional pois deve ser aplicado mais vezes sobre as matrizes de entrada.

Outro aspecto interessante a ser observado nas camadas convolucionais é que os filtros da camada de entrada são aplicados sobre as matrizes representando os canais de cores da imagem, ou seja, 1 no caso de imagem em tons de cinza ou 3 no padrão RGB. Portanto, é necessário definir qual o formato de representação das imagens[7]. É importante ressaltar os filtros de uma camada convolucional são aplicados sobre todos os canais de cores da imagem de entrada ou *feature maps* resultantes da camada anterior, gerando um número cada vez maior de mapas nas camadas subsequentes, como demonstra na arquitetura de uma ConvNet completa na Figura 5. Desta forma, se aplicarmos 8 filtros na camada de entrada sobre 3 canais de cores, teremos $3 \times 8 = 24$ *feature maps* como resultado. Se aplicarmos outros 8 filtros sobre os 24 mapas resultantes da primeira camada, teremos $8 \times 24 = 192$ *feature maps* como saída da segunda camada, e assim por diante.

[7] Note que os filtros aplicados em cada canal de cor no padrão RGB são diferentes, *i.e.*, os filtros aplicados sobre o canal R*ed* são diferentes dos aplicados no canal G*reen* que, por sua vez, são diferentes dos filtros aplicados sobre o canal B*lue*.

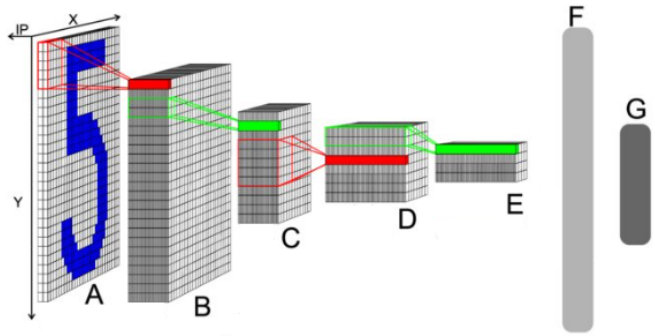

Figura 5: Arquitetura completa de uma CNN, com camadas convolucionais (de A para B e de C para D) e de *pooling* (de B para C e de D para E). As camadas totalmente conectadas F e G completam a arquitetura desta ConvNet. Perceba que o número de *feature maps* não aumenta nem diminui após uma camada de *pooling*.

Finalmente, é importante falar sobre o processo de *padding*, que adiciona bordas a imagem ou aos *feature maps* antes do processo de convolução ou *pooling*. O objetivo do padding é preservar a informação nas bordas da imagem e evitar uma redução excessiva na dimensionalidade espacial. O valor do *padding* determina o número de zeros adicionados ao redor da entrada.

Pooling

A OPERAÇÃO DE *POOLING*[8], é comumente utilizada nas ConvNets após uma ou mais camadas convolucionais. Essa operação tem como objetivo reduzir a dimensionalidade espacial dos *feature maps*, preservando as características mais relevantes, em uma operação semelhante a um "foco". A operação de pooling é frequentemente aplicada separadamente em cada canal do *feature map*.

[8] Também chamada de camada de agrupamento, foco ou camada de *subsampling*.

O processo de pooling ocorre de forma semelhante ao da convolução, onde um *kernel* retangular (*e.g.*, $(2,2)$, $(3,3)$) com uma função matemática específica é aplicado sobre os *feature maps* deslocando-o com um determinado *stride*, da esquerda para direita, de cima para baixo.

$$\text{Pooling}(R) = \text{Max}(R) \tag{3}$$
$$= \text{Average}(R) \tag{4}$$

onde: R é a região onde o *pooling* está sendo aplicado num determinado momento; Max é a operação que extrai o maior valor da região R; e $Average$ é a operação que extrai a média dos valores da região R.

Feature Maps

Os *feature maps* são representações espaciais das ativações em diferentes camadas da rede, especialmente após as camadas convolucionais. Cada *feature maps* destaca a presença de padrões específicos aprendidos pela ConvNet, como bordas, texturas ou formas mais complexas, dependendo da profundidade da camada.

Captura de Padrões Locais: Cada *feature map* é ativado por padrões locais específicos na entrada. Nas primeiras camadas convolucionais, os *feature maps* podem detectar bordas, gradientes e texturas simples. Conforme você avança nas camadas, os *feature maps* tornam-se mais complexos e capazes de capturar padrões hierárquicos e abstrações mais avançadas.

Hierarquia de Representações: À medida que os dados passam por camadas convolucionais sucessivas, os *feature maps* capturam representações cada vez mais abstratas. Padrões simples são combinados para formar padrões mais complexos e contextualmente relevantes. Isso permite que a ConvNet aprenda a representação hierárquica das características presentes nos dados. Esta representação hierárquica em ConvNets com várias camadas escondidas que auxiliou na popularização do nome *Deep Learning*.

Localização Espacial: Cada posição em um *feature map* corresponde a uma região específica na entrada. Isso mantém a informação espacial, permitindo que a rede saiba onde diferentes padrões estão localizados na imagem.

Visualização de Ativações: Visualizar os feature maps ativados em diferentes camadas da rede pode fornecer insights sobre o que a rede está aprendendo. Ferramentas como mapas de calor de ativação ajudam a entender quais partes da entrada são mais relevantes para determinadas características.

Redução Dimensional: Os *feature maps* são uma representação dimensional reduzida da entrada original, o que ajuda a reduzir a carga computacional e os requisitos de memória.

Eles servem como uma representação intermediária que captura gradualmente informações mais abstratas e contextualizadas à medida que a informação flui pela rede. Exemplos de *feature maps* podem ser observados na Figura 6, na Figura 7, na Figura 8 e na Figura 9.

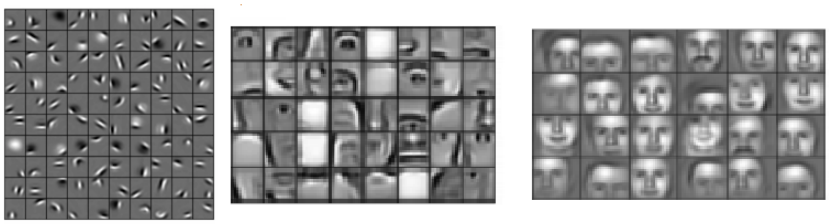

Figura 6: Exemplo de *feature maps* extraídos de uma RNA convolucional e que identificam *features* quando uma imagem de um rosto é fornecido como entrada para o modelo.

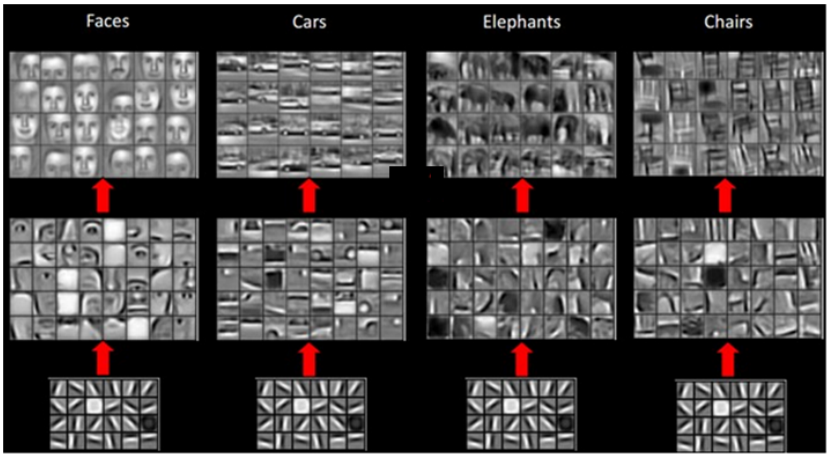

Figura 7: Exemplo de *feature maps* extraídos de uma RNA convolucional e que identificam *features* de conforme vários de imagens fornecidos como entrada para o modelo. Perceba que os mapas menores, que ficam mais próximos da camadas de saída, *i.e.*, mais à frente na RNA são os mesmos e conseguem compor as demais imagens.

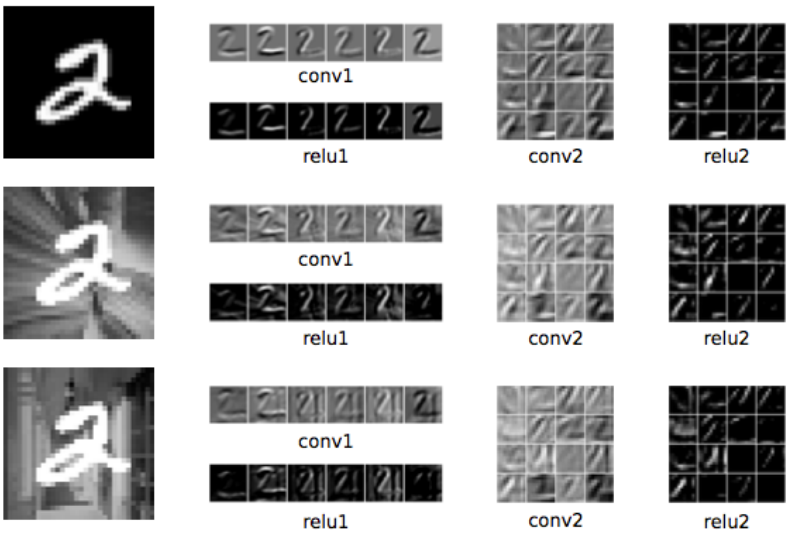

Figura 8: Exemplo de *feature maps* ativos em uma RNA convolucional quando uma imagem do dígito 2 é passada como entrada para o modelo.

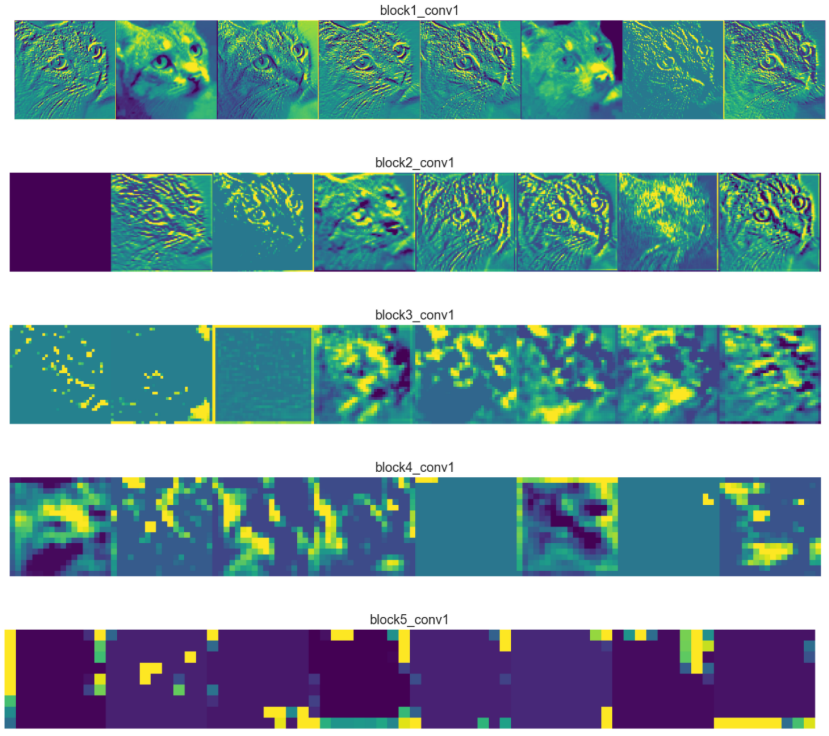

Figura 9: Exemplo de *feature maps* ativos em uma RNA convolucional quando uma imagem de um gato no canal de cor *G* (*green*) é fornecido como entrada para o modelo.

Transfer Learning

Uma ConvNet sempre terá camadas de convolução seguida de camadas de *pooling*. Estas camadas aprendem a identificar diferentes características da imagem, mantendo a sua estrutura espacial. No entanto, para completar a arquitetura da RNA, precisamos obrigatoriamente incluir uma ou mais camadas totalmente conectadas. Estas camadas totalmente conectadas servem para processar os *feature maps* antes de realizar a predição. Observe que, para utilizamos a estrutura dos *feature maps*, precisamos primeiro transformá-los num vetor unidimensional[9], numa operação chamada de *flatten*. Esta operação concatena os *feature maps* e os "achata" para que fiquem no formato de um vetor coluna. Desta forma, podemos utilizar uma ou mais camadas totalmente conectadas para transformar os vetores "achatados" de *feature maps* antes de serem fornecidos como entrada para uma camada com ativação $softmax$.

A Figura 10 apresenta a arquitetura de camadas da LeNet-5[10], como exemplo da arquitetura completa contendo as camadas totalmente conectadas, um dos primeiros modelos a obter sucesso na indústria, utilizada pelo United States Postal Service (USPS):

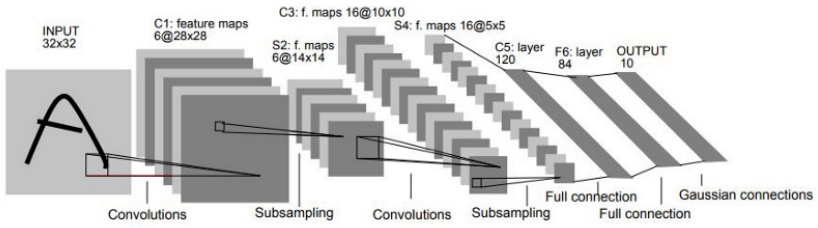

Figura 10: Representação da arquitetura da LeNet-5.

De modo geral, *Transfer Learning* é uma técnica em aprendizado de máquina onde um modelo treinado em uma tarefa específica é adaptado (transferido) para uma tarefa relacionada, geralmente usando um conjunto de dados menor para adaptar o modelo à nova tarefa. Essa abordagem é particularmente útil quando se lida com conjuntos de dados limitados, pois permite que modelos pré-treinados em grandes conjuntos de dados se beneficiem do conhecimento aprendido anteriormente. *Transfer Learning* geralmente envolve o uso de modelos pré-treinados em conjuntos de dados massivos, como o dataset ImageNet[11] para tarefas de visão computacional ou modelos treinados em grandes conjuntos de texto para tarefas de processamento de linguagem natural (PLN). Esses modelos têm a capacidade de aprender representações ricas e hierárquicas das características nos dados.

Uma das principais propriedades que tornam as ConvNets muito

[9] Embora na motivação inicial para as ConvNets tenhamos analisado brevemente as desvantagens de fornecer uma estrutura originalmente bidimensional num formato unidimensional, quando chegamos no ponto onde os *feature maps* capturam abstrações mais avançadas, o impacto de mudarmos a estrutura da entrada para a próxima camada é mínimo.

[10] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. DOI: 10.1109/CVPR.2009.5206848

interessantes é a sua utilidade para realizar "transferência de aprendizado". Os *feature maps* gerados por uma ConvNet treinada em um grande conjunto de dados podem ser usados como características pré-extraídas para tarefas relacionadas com imagens semelhantes. Esse é exatamente parte do conceito de *transfer learning*, onde uma rede pré-treinada em uma tarefa pode ser ajustada para outra tarefa com um conjunto de dados menor.

Após o pré-treinamento em uma tarefa específica, o modelo é adaptado para a nova tarefa usando um conjunto de dados menor e mais específico. Isso envolve modificar as camadas finais do modelo para se ajustar à nova tarefa, mantendo as camadas iniciais (pré-treinadas) praticamente inalteradas. Essa adaptação pode envolver o treinamento de algumas ou todas as camadas do modelo. De modo geral, em uma ConvNet, mantém-se todas (ou quase todas) as camadas convolucionais e de *pooling*, alterando-se apenas as camadas totalmente conectadas da parte final do modelo[12].

Ao remover as camadas totalmente conectadas originais do modelo, precisamos plugar novas camadas totalmente conectadas ao modelo para completar o *transfer learning*. Após esta etapa, precisamos decidir se treinamos apenas as novas camadas, mantendo os parâmetros das camadas convolucionais inalterados (*model freeze*) ou se fazemos um *fine-tuning* do modelo como um todo. O *fine-tuning* é uma técnica comum de t*transfer learning* onde o modelo é retreinado em um novo conjunto de dados. Isso permite que o modelo ajuste seus parâmetros para se tornar mais específico para a nova tarefa.

[12] Muitos autores consideram parte da RNA que contém convoluções e *pooling* um "extrator de *features*", enquanto as camadas totalmente conectadas seriam um modelo à parte, um "classificador" treinado em conjunto com o extrator de *features*.

Lars Hertel, Erhardt Barth, Thomas Käster, and Thomas Martinetz. Deep convolutional neural networks as generic feature extractors. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–4, 2015. DOI: 10.1109/IJCNN.2015.7280683; and Ben Athiwaratkun and Keegan Kang. Feature representation in convolutional neural networks. *arXiv*, abs/1507.02313, 2015

AlexNet

A AlexNet[13] é uma ConvNet que ganhou destaque ao vencer a competição ImageNet Large Scale Visual Recognition Challenge (ILSVRC) em 2012, marcando um avanço significativo no campo da visão computacional. Sua arquitetura estabeleceu um padrão para arquiteturas de CNNs subsequentes. A Figura 11 demonstra sua arquitetura.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, page 1097–1105, 2012

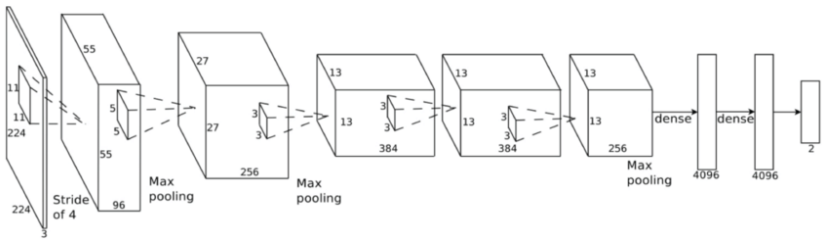

Figura 11: Representação da arquitetura da AlexNet.

A AlexNet é composta por oito camadas, sendo cinco camadas convolucionais seguidas por três camadas totalmente conectadas. As camadas convolucionais são intercaladas com camadas de *max pooling* e utiliza ativação ReLU após cada camada convolucional. As camadas convolucionais utilizam filtros convolucionais de tamanho pequeno (3×3) com um stride de 1, o que permite a captura eficiente de padrões locais. A AlexNet processa imagens de entrada de tamanho 227×227 pixels e possui cerca de 60 milhões de parâmetros.

Após as primeiras duas camadas convolucionais, a AlexNet utiliza camadas de max pooling para reduzir a dimensionalidade espacial das representações, preservando as características mais importantes. Isso também ajuda a aumentar a robustez da rede a variações de escala e posição. Esta arquitetura também incorpora a técnica de *dropout*[14] nas camadas totalmente conectadas para mitigar o *overfitting* e sua camada de saída utiliza a função de ativação softmax.

[14] Veja mais sobre esta técnica e sobre regularização no Handout 6 - Treinamento e Regularização.

A AlexNet foi uma das primeiras arquiteturas a demonstrar a eficácia das CNNs em tarefas de visão computacional em grande escala. Sua vitória no ILSVRC 2012 desencadeou o interesse renovado em deep learning e CNNs e incentivou o desenvolvimento de arquiteturas mais complexas, como o surgimento de redes mais profundas, como VGG, GoogLeNet e ResNet.

VGG-16

A ARQUITETURA VGG[15] (*Visual Geometry Group*) é uma série de arquiteturas de redes neurais convolucionais (CNNs) que foi desenvolvida pelo *Visual Geometry Group* na Universidade de Oxford. A VGGNet, especificamente, ganhou destaque por sua simplicidade e eficácia, sendo finalista na competição ImageNet Large Scale Visual Recognition Challenge (ILSVRC) em 2014. A Figura 12 demonstra a arquitetura desta ConvNet.

[15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015

A característica notável da VGG é sua profundidade. A VGGNet original tinha duas variantes principais: a VGG16, com 16 camadas (13 camadas convolucionais e 3 totalmente conectadas), e a VGG19, com 19 camadas (16 camadas convolucionais e 3 totalmente conectadas). Essa profundidade relativamente grande foi uma das maiores à época. O modelo é composto principalmente por camadas convolucionais. Cada bloco convolucional é formado por camadas convolucionais 3×3, seguidas por camadas de ativação ReLU e, frequentemente, camadas de max pooling 2×2. Essa estrutura uniforme de blocos convolucionais é uma das razões para a simplicidade e eficácia da arquitetura.

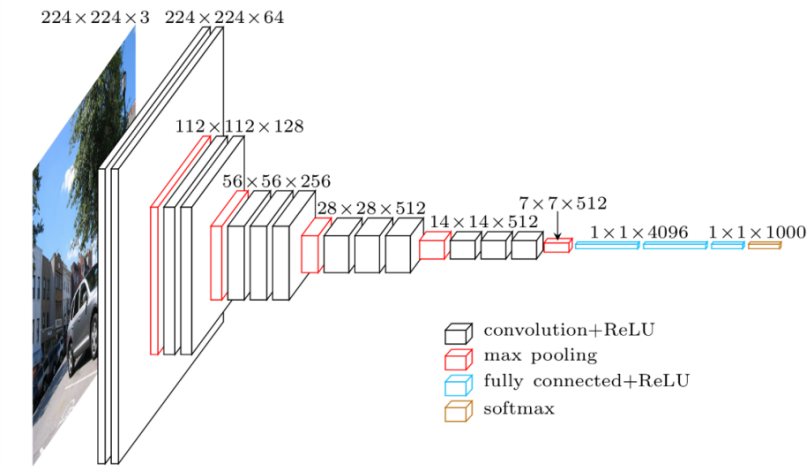

Figura 12: Representação da arquitetura da VGG-16.

A escolha de usar filtros convolucionais 3×3 em vez de filtros maiores, como 5×5 ou 7×7, foi feita para manter a profundidade da rede e reduzir o número de parâmetros. Vários filtros 3×3 em cascata têm o mesmo campo receptivo de um filtro maior, mas com menos parâmetros. Após os blocos convolucionais, a VGGNet inclui camadas totalmente conectadas, que são comuns em redes neurais mais tradicionais. Essas camadas são seguidas por funções de ativação ReLU, e a camada de saída utiliza a função de ativação softmax para classificação.

A VGGNet foi treinada usando gradiente descendente (SGD) com *momentum*. A inclusão de camadas de dropout ajudou a mitigar o *overfitting*, contribuindo para a eficácia do treinamento Apesar de ser superada em termos de precisão por arquiteturas mais recentes, a VGGNet teve um impacto significativo no campo de visão computacional. Sua simplicidade e eficácia destacaram a importância da profundidade na representação de características complexas em imagens.

GoogLeNet (Inception)

A ARQUITETURA GOOGLENET[16], TAMBÉM conhecida como *Inception*, é uma arquitetura de ConvNet desenvolvida pelo Google. Foi projetada para participar do desafio ImageNet Large Scale Visual Recognition Challenge (ILSVRC) em 2014, e obteve uma vitória notável devido à sua eficiência computacional e desempenho impressionante. A Figura 13 demonstra a primeira versão desta arquitetura.

[16] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014

O destaque da arquitetura Inception é o uso do que é conhecido

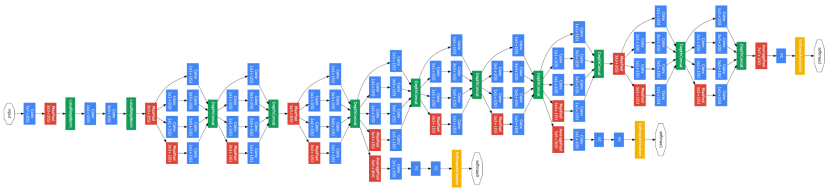

Figura 13: Representação da arquitetura da primeira versão da GoogLeNet.

como o módulo Inception. Esse módulo é uma estrutura composta por múltiplos caminhos paralelos, cada um com uma operação de convolução diferente. A ideia é capturar padrões em diferentes escalas espaciais usando filtros de diferentes tamanhos, conforme a Figura 14.

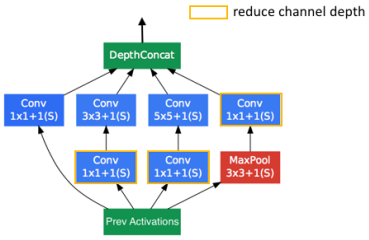

Figura 14: Representação da arquitetura dos módulos *Inception* da GoogLeNet.

No módulo Inception, diferentes caminhos realizam operações convolucionais com diferentes tamanhos de filtros (por exemplo, 1×1[17], 3×3, 5×5). Esses caminhos são executados em paralelo, e suas saídas são concatenadas para formar a saída do módulo Inception. Para reduzir a dimensionalidade entre os módulos Inception, são incorporadas camadas de pooling 3×3 com strides maiores (por exemplo, 2×2). Essas camadas de pooling contribuem para a redução da resolução espacial, ajudando na eficiência computacional. Além disso são usadas camadas de bottleneck, que consistem em convoluções 1×1 seguidas por convoluções 3×3. Essa configuração ajuda a reduzir a carga computacional sem perder muita informação.

O modelo GoogLeNet completo consiste em múltiplos módulos Inception, alguns dos quais são seguidos por camadas totalmente conectadas para a tarefa final de classificação. A arquitetura Inception é global em sua abordagem, pois procura aprender representações complexas e hierárquicas ao explorar diferentes escalas de informações simultaneamente. Para combater o *overfitting*, a arquitetura GoogLeNet incorpora técnicas como a regularização $L2$ e camadas de *dropout*.

Em 2015[18] os autores publicaram uma nova arquitetura (GoogleNet v3) que demonstrava alternativas mais eficientes em relação ao módulo Inception, conforme arquitetura da Figura 15. Convoluções com filtros espaciais grandes (como 5×5 ou 7×7) são benéficas em termos de expressividade e capacidade de extrair características em uma escala maior, mas o custo computacional é desproporcionalmente alto.

Os pesquisadores apontaram que uma convolução 5×5 pode ser representada de maneira mais econômica por dois filtros empilhados de 3×3. Enquanto um filtro $5 \times 5 \times c$ requer $25c$ parâmetros, dois filtros $3 \times 3 \times c$ requerem apenas $18c$ parâmetros. Para representar com maior precisão um filtro 5×5, não devemos usar ativações entre as duas camadas de 3×3. No entanto, os autores argumentam que

[17] Essas convoluções 1×1 são usadas para reduzir a dimensionalidade, permitindo a combinação eficiente de informações entre canais.

[18] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015

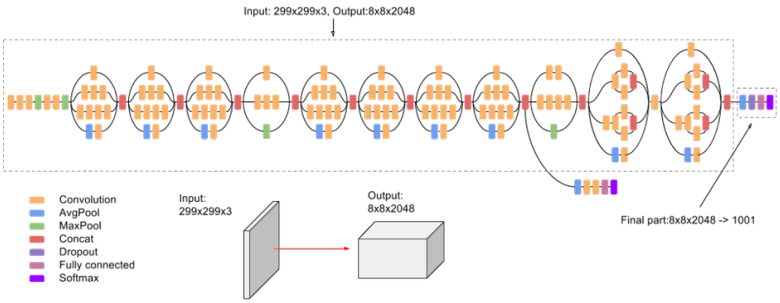

Figura 15: Representação da arquitetura da terceira versão GoogLeNet.

"ativação linear sempre foi inferior ao uso de unidades lineares retificadas (ReLU) em todas as etapas da fatoração". Os autores também demonstram que convoluções de 3×3 podem ser ainda mais decompostas em convoluções sucessivas de 3×1 e 1×3. Generalizando esse insight, podemos calcular de forma mais eficiente uma convolução $n \times n$ como uma convolução $n \times 1$ seguida por uma convolução $1 \times n$.

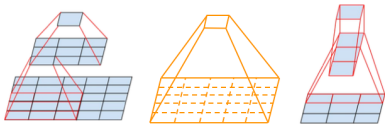

Figura 16: Nesta imagem temos a representação de duas convoluções sucessivas de 3×3 comparadas a uma única convolução 5×5 e com uma convolução 3×1 seguida de uma 1×3

A arquitetura Inception foi inovadora em seu tempo, mostrando que era possível criar arquiteturas profundas e eficientes em termos de computação para tarefas de visão computacional. Essa abordagem influenciou o desenvolvimento de arquiteturas subsequentes, incluindo a introdução de modelos mais avançados, como o InceptionV2, InceptionV3 e InceptionV4.

ResNet

A ResNet[19], ou *Residual Net*, é uma arquitetura ConvNet que introduziu uma inovação fundamental para treinar redes profundas de maneira mais eficaz. O principal problema que a ResNet aborda é o da degradação do desempenho à medida que modelo fica mais profundo. Contrariamente à intuição inicial de que redes mais profundas sempre levariam a um desempenho melhor, os pesquisadores observaram que o desempenho começou a piorar à medida que as redes ultrapassavam uma certa profundidade:

> Let us consider a shallower architecture and its deeper counterpart that adds more layers onto it. There exists a solution by construction to the deeper model: the added layers are identity mapping, and the other layers are copied from the learned shallower model. The existence of this constructed solution indicates that *a deeper model should produce no higher training error than its shallower counterpar*t. But experiments show that our current solvers on hand are unable to find solutions that are comparably good or better than the constructed solution (or unable to do so in feasible time).

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015

A ideia chave por trás da ResNet é a introdução de blocos residuais ou conexões residuais[20]. Em vez de tentar aprender diretamente a função desejada $H(x)$, a ResNet aprende a residual $F(x) = H(x) - x$. A saída da camada é então a soma do residual e da entrada original: $F(x) + x$. Essa abordagem, demonstrada na Figura 17 facilita o treinamento, pois a rede é incentivada a aprender mudanças incrementais em relação à entrada original.

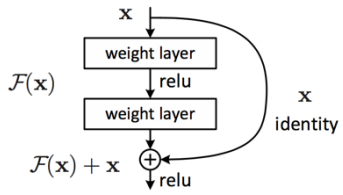

Figura 17: Conexões residuais da ResNet onde a mesma entrada x fornecida como entrada para uma camada l é fornecida mais à frente para outra camada $l + n$ juntamente com as entradas da cama $l + n - 1$.

[20] Também chamadas *skip-connection* ou *residual connections*.

Estes blocos residuais na ResNet são compostos por duas convoluções 3×3 e incluem uma conexão direta (*skip-connection*) que adiciona a entrada original à saída das convoluções. Essa conexão direta permite que o gradiente flua mais facilmente através da rede durante o treinamento, combatendo o problema do desaparecimento do gradiente. Além disso, a ResNet é conhecida por suas versões mais profundas, como a ResNet-50 e ResNet-101, que possuem 50 e 101 camadas, respectivamente (Figura 18). Essa profundidade notável foi alcançada justamente devido às conexões residuais, que facilitam o treinamento de redes profundas.

Em sua arquitetura a ResNet inclui camadas de pooling e normalização[21] para melhorar a eficiência e acelerar o treinamento. Esses elementos são frequentemente usados em conjunto com os blocos residuais.

[21] Camada de normalização de batch ou *Batch Normalization*. Para mais detalhes veja o Handout 6 - Treinamento e Regularização.

Pelo fato dos *feature maps* possuírem um nível de abstração bastante alto, a ResNet não é apenas eficaz para tarefas de classificação de imagens, mas também demonstrou ser útil em várias outras tarefas, como detecção de objetos e segmentação semântica. Sua capacidade de servir de base para treinar redes profundas com eficácia a tornou uma escolha popular em várias aplicações. e teve um impacto significativo na pesquisa em visão computacional, influenciando o design de arquiteturas subsequentes e destacando a importância das conexões residuais para treinar redes profundas.

Considerações Finais

As RNAs do tipo convolucional (ConvNets) são um tipo especializado de RNA projetada para processar dados estruturados em grade, como imagens. Elas são amplamente utilizadas em tarefas de visão computacional. A operação de convolução consiste em deslizar um filtro (também chamado de kernel) sobre a entrada (que pode ser uma imagem ou a saída da camada anterior), multiplicando os valores dos pixels da entrada pelos neurônios do filtro e somando os resultados

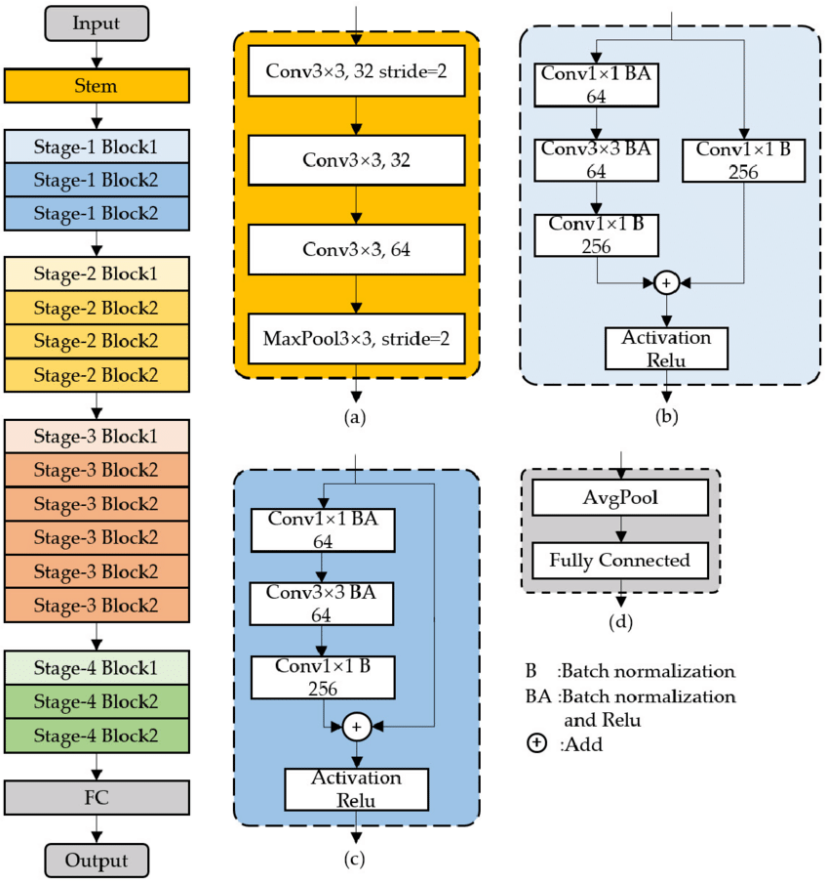

Figura 18: Exemplo de arquitetura ResNet com 34 camadas.

para produzir um único valor na saída. Este processo é repetido para cada posição possível na entrada, criando um mapa de características convolucional.

A operação de *pooling* é comumente utilizada nas ConvNets após as camadas convolucionais. Essa operação tem como objetivo reduzir a dimensionalidade espacial dos *feature maps*, preservando as características mais relevantes. A operação de pooling é frequentemente aplicada separadamente em cada canal do mapa de características.

À medida que os dados passam por camadas convolucionais sucessivas, os *feature maps* capturam representações cada vez mais abstratas. Padrões simples são combinados para formar padrões mais complexos e contextualmente relevantes. Isso permite que a rede aprenda a representação hierárquica das características presentes nos dados.

O aprendizado dos *featrure maos* facilita operações de *Transfer learning*. Esta é uma técnica em aprendizado de máquina onde um modelo treinado em uma tarefa específica é adaptado (transferido) para uma tarefa relacionada, geralmente usando um conjunto de dados menor para a nova tarefa. Essa abordagem é particularmente útil quando se lida com conjuntos de dados limitados, pois permite que modelos pré-treinados em grandes conjuntos de dados se beneficiem do conhecimento aprendido anteriormente.

Referências

Ben Athiwaratkun and Keegan Kang. Feature representation in convolutional neural networks. *arXiv*, abs/1507.02313, 2015.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. DOI: 10.1109/CVPR.2009.5206848.

Alejandro Dominguez. A history of the convolution operation. *IEE Pulse*, 2015. URL `https://www.embs.org/pulse/articles/history-convolution-operation/`.

Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Lars Hertel, Erhardt Barth, Thomas Käster, and Thomas Martinetz. Deep convolutional neural networks as generic feature extractors. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–4, 2015. DOI: 10.1109/IJCNN.2015.7280683.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

David H. Hubel and Torsten N. Wiesel. Effects of monocular deprivation in kittens. *Naunyn-Schmiedebergs Archiv for Experimentelle Pathologie und Pharmakologie*, 248:492–497, 1964.

John D. Kelleher and Brendan Tierney. *Deep Learning*. MIT Press, 1 edition, 2018.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, page 1097–1105, 2012.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL `http://neuralnetworksanddeeplearning.com/`.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.