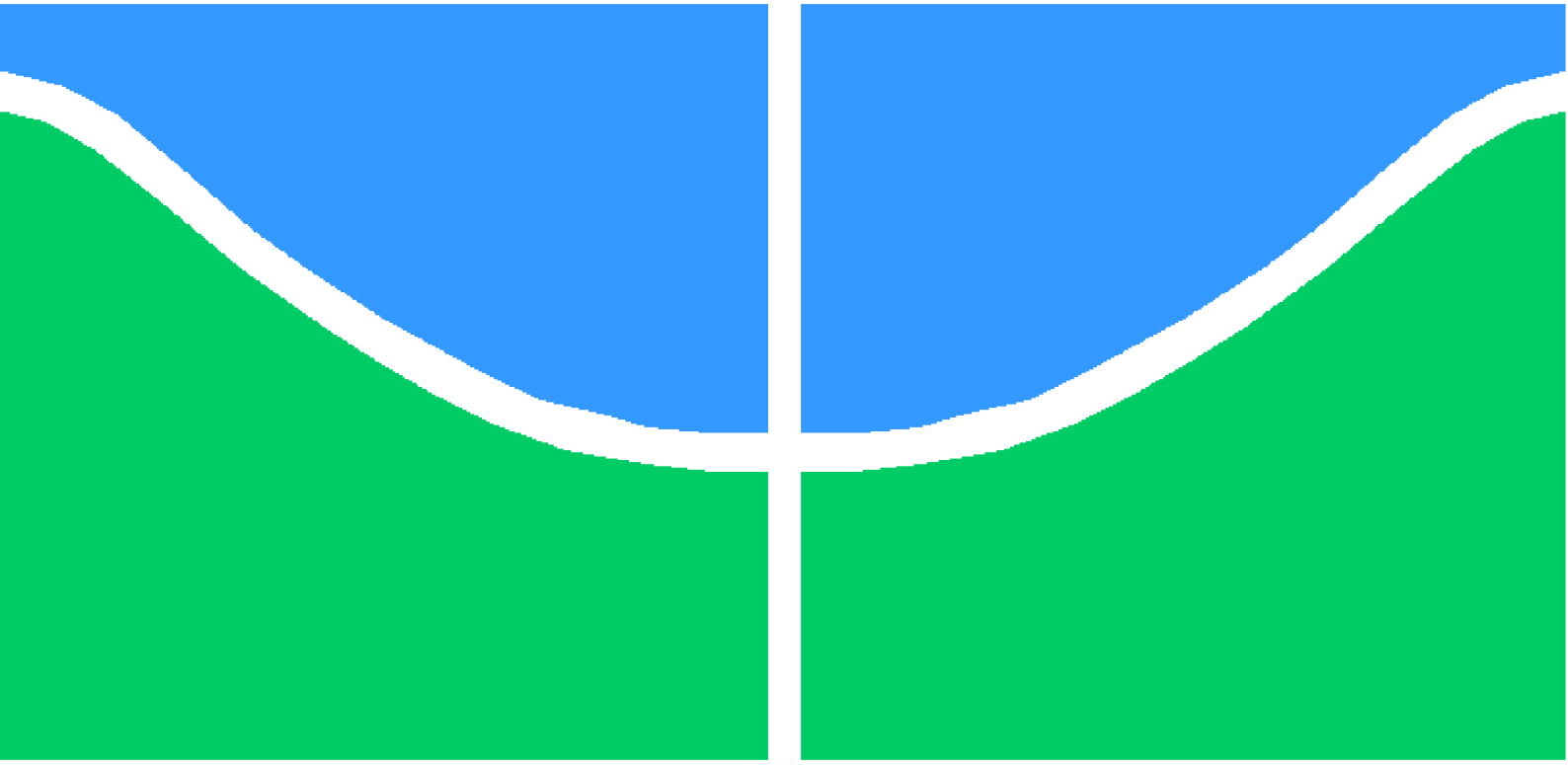


Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Reconhecimento de Gestos usando Redes Neurais Convolucionadas

Autor: André Bernardes Soares Guedes
Orientador: Doutor Teófilo de Campos

Brasília, DF
2017

André Bernardes Soares Guedes

Reconhecimento de Gestos usando Redes Neurais Convolucionadas

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Doutor Teófilo de Campos

Brasília, DF

2017

André Bernardes Soares Guedes
Reconhecimento de Gestos usando Redes Neurais Convolucionadas/ André Bernardes Soares Guedes. – Brasília, DF, 2017-
47 p. : il. (algumas color.) ; 30 cm.

Orientador: Doutor Teófilo de Campos

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2017.

1. Redes Neurais. 2. Gestos. I. Doutor Teófilo de Campos. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Reconhecimento de Gestos usando Redes Neurais Convolucionadas

CDU 02:141:005.6

André Bernardes Soares Guedes

Reconhecimento de Gestos usando Redes Neurais Convolucionadas

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, novembro de 2016:

Doutor Teófilo de Campos
Orientador

Doutor Nilton Correia
Convidado 1

Doutor Fábio Mendes
Convidado 2

Brasília, DF
2017

Resumo

Com o avanço tecnológio das plataformas de computação paralela surgiram novas oportunidades na exploração de Redes Neurais Convolucionadas (CNN). A popularização da computação em GPUs possibilita a solução de problemas reais em um intervalo de tempo muito inferior ao previamente experimentado. O objetivo desse trabalho é fazer uso de CNNs no contexto de detecção de gestos, no qual são investigadas as possibilidades de realizar reconhecimento em tempo real, refinamento de CNNs previamente treinadas, treinamento de uma nova rede e rastreamento de mãos. Uma introdução aos conceitos relevantes de redes neurais convolucionais assim como uma revisão do *estado-da-arte* no reconhecimento de gestos foram realizadas para compor o referencial teórico no qual o trabalho é fundamentado. Baseado nessa revisão, foi decidido usar o trabalho de Koller et al. (KOLLER; NEY; BOWDEN, 2016) como base do desenvolvimento deste trabalho. Experimentos de reprodução de resultados também são abordados visando verificar os resultados apresentados por Koller et al. O trabalho também consta um planejamento para futuros esforços orientados pelos resultados obtidos.

Palavras-chaves: redes neurais; reconhecimento de gestos; visão computacional.

Abstract

With the technological advancement of parallel computing platforms, the opportunity to explore Convolutional Neural Networks (CNN) has emerged. The popularization of GPU computing enables the solution of real-life problems in a smaller time-frame than ever before. The objective of this work is to make use of CNNs in the scope of hand shape classification. Some possibilities are investigated, such as real time recognition, refining of pre-trained CNNs, training networks from scratch and hand tracking. An introduction to the relevant concepts of convolutional neural networks as well as a revision of the state-of-the-art were performed to constitute a base theoretical framework in which this work is grounded. From this literature review, we decided to build upon the work of Koller et al. (KOLLER; NEY; BOWDEN, 2016). Also, experiments to reproduce Koller et al.'s results are presented. This reports also presents a plan of future efforts oriented by the obtained results.

Key-words: neural networks. gesture recognition. computer vision.

Lista de ilustrações

Lista de abreviaturas e siglas

CNN	Convolutional Neural Network
HMM	Hidden Markov Model
EM	Expectation Maximization
GPU	Graphics Processing Unit
ReLU	Rectified Linear Unit
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge
SGD	Stochastic Gradient Descent

Sumário

Introdução

A utilização de gestos na comunicação entre indivíduos é uma forma de comunicação muito utilizada em qualquer cultura globalmente, mesmo quando esse tipo de comunicação não é expressamente direta como no caso das linguagens de sinais.

Por ser uma forma de transmitir informação tão naturalmente humana, com o aumento da interatividade dos dispositivos e a busca por interfaces para a interação humano-computador, os gestos tem sido cada vez mais explorados para cobrir essa necessidade.

Em sua forma direta de comunicação os gestos são a principal forma padronizada de comunicação visual de deficientes auditivos em todo o mundo, onde existem diversas linguagens diferentes atualmente reconhecidas.

Outra utilização menos convencional de gestos é presente na fisioterapia e tratamentos de recuperação de lesões, cujas repetições de gestos especificados são essenciais para a avaliação e restauração dos movimentos dos pacientes sob tratamento. Uma pesquisa em andamento nessa área está sendo realizada numa parceria com a Universidade de São Paulo e a University of Surrey[1] Baseando-se nesses e outros fatos é fácil perceber a necessidade da solução do problema de reconhecimento de gestos ou de um de seus subproblemas como o reconhecimento de *handshapes*.

Enquanto o reconhecimento de gestos representa a determinação da configuração da mão humana durante um período contínuo de tempo, assim como suas posições e orientações, o reconhecimento de *handshapes* se restringe a determinação dessa mesma configuração em apenas um momento discreto de tempo, não levando em consideração qualquer influência temporal.

Nesse trabalho o problema de reconhecimento de gestos é abordado com o foco no reconhecimento de *handshapes*.

Por conta da quantidade de parâmetros a serem considerados (ângulos das juntas, posição da mão, orientação tridimensional, além de deformações dos musculos e da pele), mesmo o problema isolado de reconhecimento de *handshapes* se torna muito complicado para uma solução utilizando algoritmos determinísticos convencionais. Métodos de aprendizado de máquina, baseados em observação de um grande conjunto de dados de treinamento, oferecem soluções viáveis para esse problema.

Aprendizado de máquina é uma área do conhecimento cujo objetivo geral é dar a

[1] Para detalhes sobre essa parceria e o fomento da FAPESP, vide <http://personal.ee.surrey.ac.uk/Personal/T.Decampos/OccupationalTherapy/> e <http://www.bv.fapesp.br/en/auxilios/91508/hand-tracking-for-occupational-therapy/>.

computadores a habilidade de aprender (a resolver um problema, modelar dados ou tomar decisões) sem serem explicitamente programados para tal (WIKIPEDIA, 2017). Essa área se relaciona intimamente com Inteligência Artificial e Estatística. A extração automática de informações oriundas de imagens ou sequencias de vídeo é o objeto do trabalho na área de Visão Computacional. Até os anos 90, a maioria dos problemas de Visão Computacional eram tratados usando modelos dos objetos de interesse, construídos manualmente. Havia uma ênfase maior no uso de geometria. Com os avanços em Aprendizado de Máquina, houve uma transformação nessa área de pesquisa. Métodos geométricos passaram a ser aplicados somente quando há uma necessidade de se inferir medições tridimensionais do espaço ou dos objetos de interesse, enquanto que tarefas relacionadas com reconhecimento e identificação de objetos, ações e outros padrões visuais passaram a ser abordadas usando métodos de aprendizado de máquina.

Recentemente, significantes avanços em aprendizado de máquina foram alcançados utilizando técnicas baseadas em redes neurais convolucionais. Foram quebradas barreiras do desempenho em várias tarefas, começando pelo reconhecimento de objetos em imagens. Por isso, este trabalho foca no uso de redes neurais convolucionais para reconhecimento de *handshapes*.

O capítulo 1 faz uma breve introdução a redes neurais, com ênfase nas técnicas propostas na última década para viabilizar seu funcionamento em problemas de grande escala. Posteriormente, o capítulo 2 revisa um método do estado-da-arte em reconhecimento de *handshapes*. No capítulo 3 são apresentados detalhes sobre a reprodução dos resultados desse método e novos experimentos. Este trabalho conclui no capítulo 4, que traça planos de trabalho para os próximos passos.

1 Redes Neurais Convolucionais

1.1 Introdução

Redes neurais são grupos de neurônios conectados entre si que formam um sistema nervoso, tal como o cérebro humano. As redes neurais artificiais são abstrações do modelo biológico que permitem representar os neurônios como unidades simplistas de computação. Nessa analogia os neurônios artificiais simulam o comportamento dos biológicos ao acumular os impulsos provenientes de entradas ou de axônios de outros neurônios até que um certo limite é atingido, esse limite é definido pela função de ativação. Quando esse limite é atingido o neurônio dispara um impulso por seu axônio que por sua vez pode estimular outros neurônios ou ser uma saída (veja Figura 1).

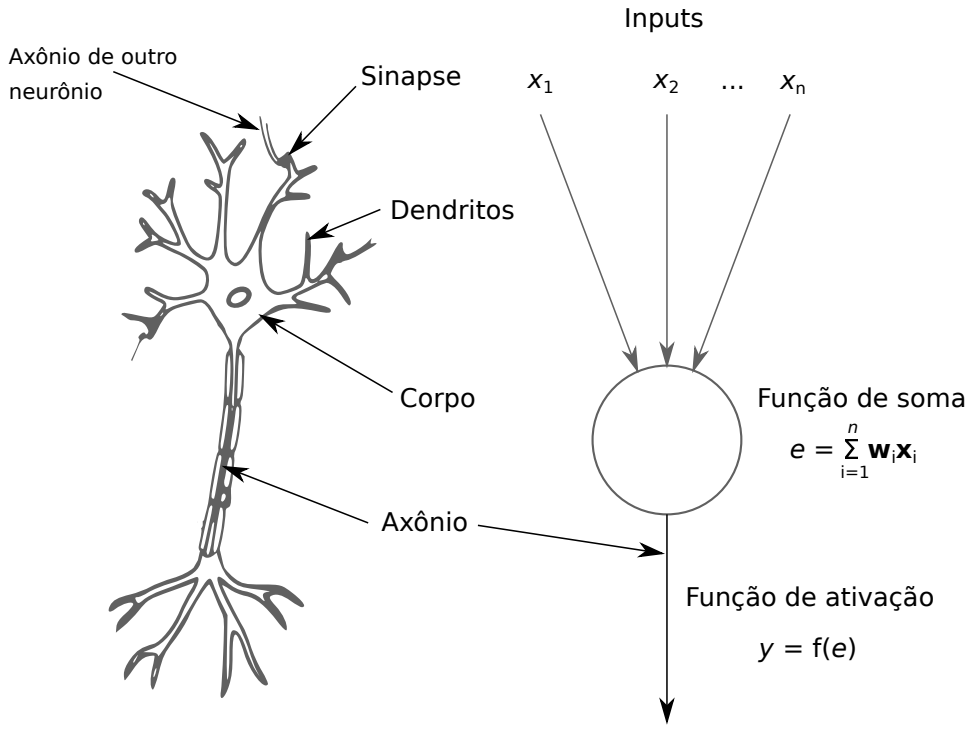


Figura 1 – Neurônio biológico e neurônio artificial generalizado.

O conceito de artificialmente imitar o comportamento das redes neurais biológicas já é explorado desde a década de 1940, quando Warren McCulloch e Walter Pitts modelaram uma simples rede neural usando circuitos elétricos (MCCULLOCH; PITTS, 1943). Isso gerou uma grande especulação quanto às infindas possibilidades que poderiam ser alcançadas, porém a falta de tecnologia para suportar esses avanços e um sentimento de medo crescente quanto aos efeitos dessa tecnologia nas vidas humanas reduziram bastante o seu impulso (ROBERTS, 2000), sendo posteriormente quase totalmente ofuscado pelo

surgimento da arquitetura de John von Neumann.

Atualmente com a evolução do hardware e o avanço da computação paralela a utilização de redes neurais está cada vez mais proeminente, porém, redes neurais convencionais ainda necessitam de um nível de processamento muito elevado, mesmo com uma baixa quantidade de camadas. Isso se dá pois essas redes, chamadas de *"totalmente conectadas"*, contem um número quadrático de conexões entre as camadas o que torna o treinamento muito lento (ROBERTS, 2000).

1.2 Viabilizando Redes Neurais para uso em problemas reais

Com o avanço nas pesquisas sobre o córtex visual animal foi constatado que a organização dos neurônios e suas correlações são mais esparsas que as do cérebro humano (HUBEL; WIESEL, 1968), onde pequenas regiões do espaço visual eram responsáveis pelos estímulos que acionavam neurônios individualmente. Essas regiões são conhecidas como campos receptivos.

As redes neurais convolucionais são baseadas nesses princípios para minimizar o custo computacional necessário para o treinamento das mesmas. Quando comparadas com as redes neurais multi-camadas tradicionais o ganho de desempenho é muito significativo, e é ainda mais evidenciado pelo surgimento de algorítimos eficientes para computação de tais redes usando GPUs.

Algumas características importantes diferenciam esse tipo de rede neural dos demais tipos:

- Por conta da correlação mais esparsa dos neurônios, características espacialmente afastadas terão ligações bem fracas, se não nulas, entre si. Isso torna a rede mais capaz de generalizar características afastadas individualmente, porém também invalida o uso desse tipo de rede quando exitem relações importantes entre características que não estão próximas espacialmente. Imagens em geral muitas vezes obedecem essa restrição, pois características visuais só agregam valor semântico quando espacialmente aproximadas. Dessa forma, uma generalização válida é que o uso desse tipo de rede é mais indicado para imagens ou qualquer outro tipo de dados que pode ser transformado em uma imagem sem perda de informações relevantes.

- Diferentemente das redes neurais muti-camadas tradicionais, onde a estrutura de entrada é unidimensional, as redes neurais convolucionais possuem como entrada uma estrutura tridimensional com dimensões $h \times w \times d$ onde h é sua altura, w é sua largura e d é sua profundidade. Quando referentes a uma imagem, essas dimensões são relativas à altura, largura e ao número de características distintas a serem observadas.

- Assim como a estrutura de entrada, a disposição dos neurônios nesse tipo de rede também é um volume tridimensional, dessa forma cada grupo de neurônios dispostos ao longo da profundidade do volume é estimulado por seu respectivo campo receptivo e os parâmetros são compartilhados entre os neurônios de uma mesma profundidade. Isso efetivamente faz com que todos os neurônios de uma profundidade na camada detectem a mesma característica através de todo o campo visual.

1.2.1 Camadas

Esse tipo de rede recebe o nome de rede neural convolucional por conta de sua camada mais predominante, a camada de convolução. Além desse tipo de camada existem outros que também são utilizados para melhorar o desempenho desse tipo de rede. Algumas das mais importantes estão detalhadas a seguir.

1.2.1.1 Convolução

Sendo a camada mais fundamental de uma rede neural convolucional, é nesse tipo de camada que cada característica específica em alguma posição espacial da entrada é detectada. Os parâmetros dessa camada são distribuídos em um conjunto de *kernels* com campos receptivos pequenos porém atuando em toda a profundidade da entrada. A convolução se é dada durante o *forward pass*, onde cada um desses *kernels* são convolvidos pela largura e o tamanho da entrada. Esse tipo de convolução é realizada entre sinais discretos e por se tratar de um sistema linear invariante com o tempo, uma simplificação de convoluções entre sinais é empregada. Essa simplificação torna a operação de convolução em um produto entre os elementos da entrada sobrepostos pelo *kernel* deslocado e o próprio *kernel* que os sobrepôs (AHN, 2012). Dessa forma essa camada é responsável por estabelecer a relação de conectividade local desse tipo de rede.

Isso melhora sua capacidade de reconhecer padrões visuais, pois assim são mantidas as relações localizadas e ignoradas as influencias dispersas que poderiam levar a um sobreajuste (*overfitting*) da rede.

Alguns hiperparâmetros são importantes na configuração de tal camada:

- **Tamanho do campo receptivo** (*kernel size*): É o tamanho bidimensional do campo receptivo que será convolvido com a entrada, é o mesmo para todos os neurônios da camada. Usualmente assume valores como 5×5 ou 16×16, dependendo do tamanho da entrada.

- **Profundidade** (*depth*): É o número de neurônios que serão estimulados pela mesma parte do campo visual (campo receptivo) ao longo da dimensão da profundidade. Cada um desses níveis de profundidade é responsável por detectar uma caracterís-

tica por toda entrada, constituindo então um conjunto de mapas de característica (*feature maps*). A Figura 2 ilustra uma camada de convolução com profundidade cinco.

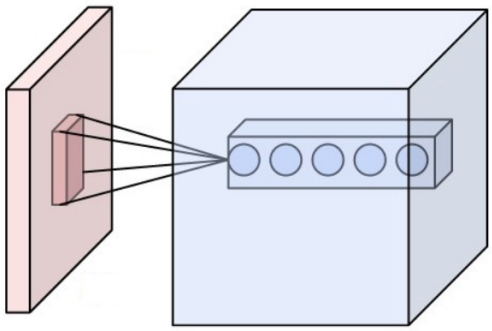


Figura 2 – Disposição dos neurônios na camada de convolução e seu campo receptivo. (APHEX34, 2015)

- **Stride**: É o número utilizado para definir o espaçamento da sobreposição dos campos receptivos dos neurônios de colunas diferentes com relação à profundidade. A Figura 3 mostra como diferentes valores para esse hiperparâmetro podem alterar os campos receptivos de tais neurônios e o número total dos neurônios de cada fatia da profundidade (*depth slice*). Esse parâmetro é decisivo na capacidade de identificar os padrões visuais da entrada, sendo responsável por definir quão refinada será a identificação do padrão mesmo que trasladado, onde o aumento desse refinamento é inversamente proporcional ao desempenho da camada, por conta do maior número de neurônios da mesma.

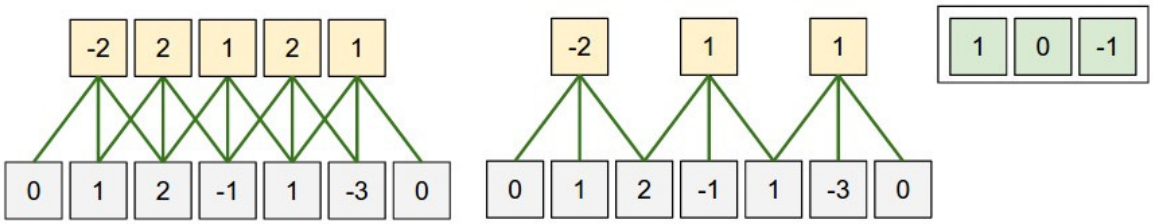


Figura 3 – Neurônios espaçados por *stride* 1 e 2 da esquerda para direita respectivamente. Separados à direita estão os pesos compartilhados entre os todos os neurônios (KARPATHY, 2015).

- **Zero-padding**: Determina qual tamanho do preenchimento com zeros que será adicionado às bordas da entrada. Isso muitas vezes é utilizado para adequar o tamanho da entrada ao *stride* escolhido, como pode ser visto na Figura 3 onde o tamanho da entrada (representada unidimensionalmente) é cinco e o *zero-padding* é um.

Outra característica importante que torna essa camada mais eficiente computacionalmente é o compartilhamento de pesos entre os neurônios de uma mesma fatia de profundidade. Como visto na Figura 3 os valores posicionados mais à direita são os pesos compartilhados entre os neurônios, onde cada valor é referente a uma das arestas do neurônio. Essa organização reduz grandemente a quantidade de parâmetros mutáveis e possibilita uma computação muito eficiente do volume de saída da camada, também diminuindo o consumo de memória da rede e principalmente melhorando o desempenho da operação de atualização de tais pesos.

1.2.1.2 Pooling

A camada de pooling é também uma das mais utilizadas na construção de redes neurais convolucionais pois são as responsáveis por generalizar as posições dos padrões geralmente encontrados pelas camadas de convolução. Efetivamente essa camada é responsável por fazer um downsampling não linear parametrizado, onde, assim como na camada de convolução, existem os parâmetros tamanho do filtro e *stride* que juntamente com a função de ativação configuram essa camada.

Algumas funções de ativação para esse tipo de camada são mais usadas, como a função de média e a função de máximo, porém ultimamente a função de máximo tem sido preferida por obter um maior desempenho empírico. Essa preferência é tão expressiva que muitas vezes essa camada é referida pelo nome *Maxpooling*.

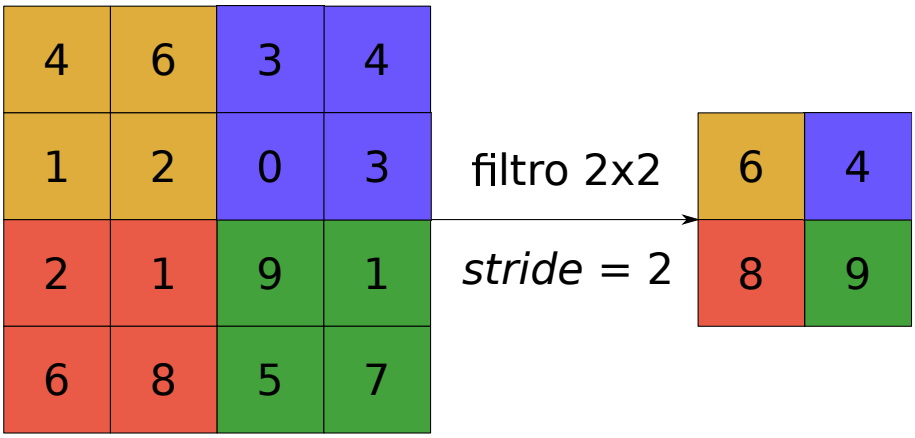


Figura 4 – Operação da camada *pooling* utilizando a função de máximo com tamanho do filtro 2×2 e *stride* igual a 2

1.2.2 Funções de ativação

Funções de ativação em redes neurais convolucionais são filtros aplicados à saída de uma camada da rede para definir como os impulsos da entrada terão influencia na saída. Juntamente com as funções de ativação dos neurônios das camadas de uma rede

também existem outras funções de ativação adicionais que são empregadas para melhorar o desempenho da rede. Esses filtros são chamados *inplace* pois a dimensionalidade de entrada é mantida e cada elemento é processado individualmente.

Existem várias funções de ativação *inplace* que são utilizadas para aumentar a não linearidade da rede, algumas delas sendo a função sigmoid, a tangente hiperbólica, o threshold e a unidade linear retificada (*Rectified Linear Unit - ReLU*), onde entre essas a *ReLU* é a mais utilizada atualmente por ter uma eficácia equivalente a de funções complexas mesmo sendo uma função simples, e portanto, com menor custo computacional o que a torna mais viável. A *ReLU* é denotada matematicamente por $ReLU(input) = \max(0, input)$ (KARPATHY, 2016).

1.3 Treinamento

Como as redes neurais convolucionais são geralmente usadas para aprendizado supervisionado elas precisam de treinamento com grandes montantes de dados para serem capazes de generalizar melhor as demais entradas às quais nunca foi exposta. A forma com que a rede é treinada é um fator determinante no desempenho da rede, tanto quanto sua própria arquitetura.

Esse treinamento é realizado através da *passagem* dos dados de treinamento pela rede múltiplas vezes, onde o número dessas iterações é um dos parâmetros a serem determinados. Cada passagem de dados pela rede é dividida em duas etapas, *forward pass* e *backpropagation* (KARPATHY, 2016).

O *Forward pass* consiste na passagem da entrada por todos os neurônios necessários para alcançar a saída, ou seja, é a passagem do volume de entrada pelas camadas da rede, nas quais suas operações típicas serão realizadas levando em consideração os pesos já aprendidos (ou inicializados) . Os resultados obtidos pelo *forward pass* de cada camada é então utilizado na etapa de *backpropagation* para a atualização dos parâmetros da rede.

Formalmente nessa etapa a entrada é propagada pelas funções específicas de cada camada da rede, na forma $f_n(\ldots f_2(f_1(f_0(x)))\ldots)$ onde f_i é a função de ativação da camada i e n se refere ao número de camadas da rede.

1.3.1 Backpropagation

A etapa de *backpropagation* é onde a aprendizagem dos pesos é realizada. Esse aprendizado se dá através da otimização (minimização) da função *loss*, a qual determina a *qualidade* da classificação do dado de entrada. Atualmente esse tipo de otimização é realizado utilizando um método chamado *Stochastic Gradient Descent* (SGD) que busca a minimização da função *loss* ao alterar os pesos na direção de maior declive do gradiente

dessa função. Essa alteração dos parâmetros é feita através da modificação dos pesos W e dos valores de *bias* b que são parâmetros livres da rede. Esse passo busca otimizar $s_j = f(x_i, W, b)_j$ de forma que quando $j = y_i$ o valor de s_j seja o maior possível. Onde x_i e y_i são referentes à entrada e o rótulo da amostra i respectivamente e f a função composta das camadas da rede.

Essa otimização é realizada em *batches* muitas vezes chamados *mini-batches* que são subconjuntos da entrada, divididos assim por conta de limitações na capacidade de memória das GPUs, as quais são os dispositivos mais utilizados nos treinamento de redes neurais convolucionais atualmente. Essa divisão também é um hiperparâmetro da rede, sendo geralmente aceito que o treinamento com *batches* maiores resulta numa melhor performance da mesma.

O método SGD requer o cálculo do gradiente da função *loss* o qual pode ser encontrado tanto numericamente quanto analiticamente. Por ser exato, o método analítico é preferido, porém muitas vezes o método numérico é utilizado para validar o resultado do método analítico. Atualmente as implementações de redes neurais convolucionais já possuem funções *loss* implementadas, assim como seus gradientes, para a qualificação das diferenças entre as classes preditas e as classes esperadas. Em geral as funções de *loss* são calculadas como a média da perda individual de cada exemplo $L = \frac{1}{N}\sum_{i=1}^{N} L_i$ (nas equações subsequentes a função de ativação das saídas da rede serão denotadas por f e os rótulos de cada exemplo por y_i). Dentre as funções mais usadas para classificação uma que se destaca é a *cross-entropy classification loss* (1.1) que indica o grau de perda entre duas distribuições de probabilidades, uma sendo referente a predição e a outra referente ao esperado. Outros exemplos de função *loss* utilizadas para classificação são a Support Vector Machine (SVM) *loss* (1.2) e a função *loss* para classificação binária (1.3). Dependendo da tarefa a ser realizada, como regressão ou classificação de atributos, funções *loss* diferentes devem ser usadas.

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \tag{1.1}$$

$$L_i = \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + 1) \tag{1.2}$$

$$L_i = \sum_j \max(0, 1 - y_{ij} f_j) \tag{1.3}$$

Esse gradiente mesmo que calculado analiticamente precisa ser propagado para as camadas para saber qual é o gradiente local de cada uma delas, possibilitando então a atualização apenas dos pesos necessários para a minimização da função *loss*. Esse processo de *backpropagation* é realizado utilizando a regra da cadeia do cálculo para gradientes, essa regra indica que um gradiente pode ser composto pela multiplicação de gradientes locais mais simples. Por exemplo, considerando a função $f(x, y, z) = (x + y)z$, ela pode

ser dividida em duas funções menores $q = x + y$ e $f = qz$ de onde obtemos as derivadas parciais $\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$ e $\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$. Com essas derivadas parciais podemos usar a regra da cadeia ao multiplicar as mesmas na forma $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$ para obter o gradiente completo da função (KARPATHY, 2016).

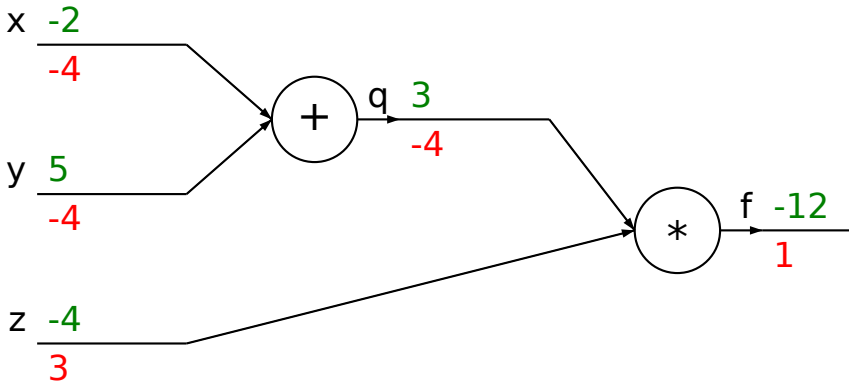


Figura 5 – Exemplo da decomposição da função f em nós de operação, onde os valores em verde representam as entradas e subsequentes computações das mesmas (*forward pass*) e os valores em vermelho denotam o valor do gradiente em cada passo da operação.

Dessa forma se considerarmos a função f da Figura 5 como sendo a função *loss* e a entrada z como um peso podemos observar que o *backpropagation* nos informa qual a direção que esse peso deve variar para aumentar a saída da função, nesse caso mostrando que o peso precisa ser aumentado para que o valor da saída fique menos negativo. Portanto se atualizarmos o peso com os valores invertidos (-3 ao invés de 3) do resultado do *backpropagation* podemos fazer com que a saída da função diminua, o qual é o propósito dessa etapa, minimizar a função *loss*.

Com essa decomposição se torna possível otimizar expressões relativamente arbitrárias, tornando essa ferramenta muito poderosa e necessária ao se utilizar multiplas e diferentes camadas em uma rede neural convolucional.

1.3.2 Atualização de parâmetros

Quando portado do gradiente calculado para cada parâmetro ainda existe a necessidade de atualizar os parâmetros de acordo com esse resultado obtido. Existem atualmente diversos métodos para alcançar esse fim, esses métodos utilizam de alguns hiperparâmetros para calibrar a forma com que os pesos devem ser atualizados. Dentre os mais utilizados se destacam:

- **Stochastic Gradient Descent (SGD) com Nesterov Momentum** é um método onde, em uma alusão à mecânica, a velocidade de descida do resultado da função *loss* na superfície multidimensional determinada pelo gradiente é calculada e

usada para escalar o tamanho do passo que será dado em direção ao ponto mínimo. Utilizando essa velocidade e o valor do gradiente na posição atual é possível prever a próxima posição dessa *partícula virtual* onde então é calculado o gradiente da nova posição. Os parâmetros são então atualizados baseados nesse segundo gradiente e em um hiperparâmetro denominado taxa de aprendizado (*learning rate*) (BENGIO; BOULANGER-LEWANDOWSKI; PASCANU, 2013).

- **Adagrad** é um método adaptativo de taxa de aprendizado proposto originalmente por (DUCHI; HAZAN; SINGER, 2011) onde a taxa de aprendizado é normalizada pela raiz da soma dos gradientes de cada parâmetro ao quadrado. Isso tem o efeito de reduzir a taxa de aprendizado dos pesos que recebem gradientes altos e ao mesmo tempo aumentar a taxa de aprendizado de pesos que recebem atualizações infrequentes. Um hiperparâmetro de suavização também é usado para inibir a divisão por zero.

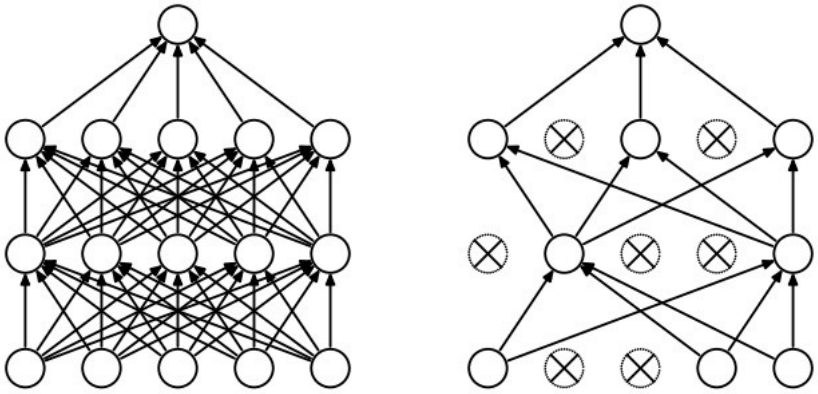
- **Adam** é um método derivado de um outro método chamado RMSprop que basicamente ajusta o método Adagrad para que a taxa de aprendizado não diminua agressivamente. A contribuição do método Adam é adicionar um momento na atualização (semelhante ao Nesterov Momentum) e suavizar os ruídos do gradiente antes de fazer essa operação. Ele herda do RMSprop a adição de uma taxa de decaimento na soma dos gradientes de cada parâmetro que reduz a agressividade de quanto a taxa de aprendizado é reduzida a cada passo (KINGMA; BA, 2014).

A taxa de aprendizado mencionada é aplicada à atualização de cada parâmetro individualmente na forma $w = -\lambda * \nabla w$ onde λ representa a taxa de aprendizado e ∇w o gradiente local da camada desse peso.

Além da atualização dos parâmetros com o gradiente da função *loss* também existem outras atualizações para aumentar a performance da rede e torná-la mais robusta em sua capacidade de generalização.

Uma função que é muitas vezes embutida na própria função *loss* é a função de regularização. Ela é introduzida para regularizar os valores dos pesos buscando distribuir melhor o aprendizado por todos os neurônios. A função provavelmente mais usada para esse fim é a da regularização ℓ^2, onde para cada parâmetro da rede o termo $\frac{1}{2}\lambda w^2$ é adicionado ao resultado da função *loss*, w representando o peso em questão, λ sendo a força da regularização e o fator $\frac{1}{2}$ é multiplicado para simplificar o gradiente da função. Essa regularização penaliza neurônios com pesos muito altos levando a uma distribuição das atualizações para os neurônios com pesos menores. Ao introduzir essa função é necessário lembrar que os pesos também devem ser atualizados pelo gradiente da função de regularização, o que é um passo geralmente realizado após a atualização quanto ao gradiente da função *loss*.

A introdução da função de regularização ajuda a combater o *overfitting* e juntamente com ela uma outra técnica que é muito utilizada para esse fim é o *Dropout* (SRIVASTAVA et al., 2014). O *Dropout* é uma técnica que condiciona à existência de um neurônio na fase de treinamento em função de uma probabilidade predefinida por um hiperparâmetro. Esse método essencialmente desativa neurônios com uma probabilidade p para que a rede seja capaz de otimizar a função *loss* mesmo sem a presença de todos os seus neurônios. Para manter a proporção durante o teste esses neurônios tem que ter suas ativações escaladas pelo valor de p para levar em conta os neurônios que foram desativados. Uma variação que tem se provado útil é ao invés de fazer essa escala durante a fase de teste, onde a performance é crucial, a escala é realizada durante a fase de treinamento, possibilitando o teste sem qualquer alterações ou perda de performance. Essa variação é chamada de *Dropout invertido* (KARPATHY, 2016).



(a) Standard Neural Net (b) After applying dropout.

Figura 6 – Imagem ilustrando o efeito do *Dropout* em uma rede neural onde os neurônios marcados foram desativados, forçando a rede a não depender deles, evitando o *overfitting* (SRIVASTAVA et al., 2014).

2 Reconhecimento de Handshapes

Reconhecimento de *Handshapes* é a determinação da configuração da mão humana pela posição das juntas e os seus respectivos ângulos. Esse reconhecimento pode ser feito de várias formas incluindo a determinação dos ângulos através de imagens de profundidade ou imagens segmentadas e também através classificação de poses específicas da mão utilizando apenas imagens convencionais.

Vários métodos atualmente demonstram resultados positivos nesse tipo de reconhecimento, utilizando desde *template matching* (POTAMIAS; ATHITSOS, 2008) até regressão com redes neurais (OBERWEGER; WOHLHART; LEPETIT, 2015). Dentre estes métodos o de Koller *et al.* utiliza uma técnica de aprendizado fracamente supervisionado para treinar uma rede neural convolucional a partir de imagens ambiguamente rotuladas (KOLLER; NEY; BOWDEN, 2016). Será revisto nessa seção o artigo de Koller *et al.* com o foco no treinamento da rede neural convolucional.

O artigo introduz um método de treinamento para redes neurais convolucionais utilizando anotações de sequências ambíguas de *frames*. Para o uso desse tipo de anotações com redes neurais convolucionais essas anotações precisam ser transformadas em anotações de *frame* pois esse tipo de rede classifica *frames* unitários e não sequências de *frames*.

Esse tipo de transformação não pode assumir que todos os *frames* anotados com um certo rótulo possuem aquele rótulo, pois existem aqueles dos quais nenhuma classe pode ser inferida. Para resolver esse tipo de problema é preciso determinar a janela na qual os frames anotados podem ser rotulados conforme a anotação da sequência.

A solução desse tipo de problema é vantajosa pois possibilita o treinamento dessas redes com um número de amostras muito maior do que o previamente observado, além de tornar o treinamento mais robusto onde *frames* com conteúdo não convencional podem ser rotulados com a classe adequada, por conta dos dados de treinamento serem “contínuos”.

O trabalho foca no problema do reconhecimento de *handshapes* independentemente da pose do indivíduo cuja imagem da mão foi capturada. Koller *et al.* propõem o embutimento de uma rede neural convolucional num *framework* de *Expectation Maximization* (EM) para treinar um classificador baseado em *frames*. Esse classificador deve ser capaz de propriamente identificar a classe esperada dentro dum total de 60 classes de *handshapes* ou corretamente descartar a existência de qualquer um dos gestos disponíveis classificando a entrada com uma classe de lixo.

2.1 Estratégia utilizada para o treinamento da rede

O alvo da pesquisa explicitada pelo artigo de Koller *et al.* é a proposta do preprocessamento realizado em *datasets* ambíguos e fracamente anotados para o treinamento de redes neurais convolucionais. Nas próximas seções os *datasets* utilizados são identificados, a proposta é sumarizada e a arquitetura da rede neural convolucional é detalhada.

2.2 Datasets

Datasets no contexto de aprendizado supervisionado consistem de conjuntos de dados (neste caso imagens) onde idealmente para cada dado desse conjunto existe um rótulo confiável correspondente a classe do dado em questão. No escopo do trabalho de Koller *et al.* essas premissas não se aplicam inteiramente, pois os dados nem sempre possuem rótulos e quando possuem não são inteiramente confiáveis por conta da ambiguidade das classes.

Os dados utilizados para o treinamento da rede são provenientes de três *datasets* distintos. Dois deles são constituídos de vídeos disponíveis publicamente com sequências de sinais isolados do léxico da linguagem de sinais Dinamarquesa e da linguagem de sinais da Nova Zelândia. Já o outro representa o conjunto de treinamento publicamente disponível do RWTH-PHOENIX-Weather 2014.

Os dados provenientes dos *datasets* neozelandês e dinamarquês possuem rótulos linguísticos que por sua vez têm sua própria taxonomia. No trabalho de Koller *et al.* a taxonomia utilizada foi a do conjunto dinamarquês e as taxonomias dos demais conjuntos foram mapeados para ela. O conjunto de dados do PHOENIX-Weather não possui anotações de *handshape* portanto para a obtenção da taxonomia dos *hanshapes* foi utilizado um léxico de linguagem de sinais chamado *SignWriting*. Dessa forma a taxonomia derivada do léxico *SignWriting* foi também mapeada para a taxonomia dinamarquesa.

Os *datasets* neozelandês, dinamarquês e PHOENIX-Weather possuem 97, 192 e 532 minutos respectivamente, exibindo um total de 71531 sinais, executados por 23 indivíduos distintos.

2.3 Algorítimo apresentado

Sendo responsável pelo refinamento das anotações dos vídeos, o algorítimo proposto treina um classificador de *frames* baseado em uma rede neural convolucional a partir de uma versão preprocessada dos conjuntos de treinamento.

Esse preprocessamento consiste do rastreamento e recorte das mãos nos *frames* de entrada e também uma disposição inicial desses *frames* de acordo com suas classes

anotadas e a classe de lixo. Para efetuar o rastreamento foi utilizado um rastreador *offline* que utiliza programação dinâmica para rastrear as mãos sem necessidade de um modelo prévio. A disposição inicial dos dados é feita de acordo com as anotações originais dos vídeos as quais serão refinadas pelo algorítimo em questão.

A etapa de refinamento é precedida pela construção do léxico, onde são adicionados para cada sequência particionada os *handshapes* anotados. No caso de haver mais de um *handshape* para uma mesma sequência, ambos são adicionados separadamente ao léxico além da própria sequência em si. Esse passo se faz necessário para a equivalência das classes dos diferentes *datasets*.

Com o léxico definido, as imagens pré-processadas e a inicialização concluída resta a execução do algorítimo de treinamento da rede. Para o treinamento do modelo é utilizado um algorítimo de EM (DEMPSTER; LAIRD; RUBIN, 1977) no cenário de *Hidden Markov Models* (HMMs) onde a rede neural convolucional é utilizada para modelar a probabilidade posterior de uma classe de *handshape* dada uma imagem.

O algorítimo de EM consiste em duas etapas, onde a primeira (etapa E, de *Expectation*) identifica a melhor atribuição da disposição das classes para as imagens e a segunda (etapa M, de *Maximization*) re-estima o modelo para adaptar a essa mudança. Tradicionalmente esse algorítimo é aplicado utilizando um HMM onde a probabilidade posterior utilizada para refinar a disposição das classes é uma Mistura Gaussiana (DEMPSTER; LAIRD; RUBIN, 1977). No escopo do artigo essa Mistura Gaussiana é substituída pela rede neural convolucional a qual é treinada a cada iteração do algorítimo.

A estrutura do HMM é heterogenia, onde existe um modelo de transição de estados cuja estrutura é na forma *bakis* enquanto os estados de lixo são estados ergódicos para ser possível a inserção entre sequencias de símbolos (KOLLER; NEY; BOWDEN, 2016).

Sumarizando, o algorítimo recebe as imagens preprocessadas juntamente com o léxico preparado e primeiramente treina a rede com a disposição inicial das classes. Após isso o passo E do algorítimo de EM utiliza as probabilidades de classes oferecidas pela rede neural convolucional para alimentar o HMM e obter a sequencia de símbolos alinhados às imagens que melhor se encaixa no modelo. Já o segundo passo (passo M) estima as mudanças nos parâmetros necessários para atualizar o modelo segundo o melhor alinhamento encontrado no passo E. Após a iteração de EM a rede é novamente treinada utilizando a nova disposição de classes para sequencias de imagens, sendo repetido após isso o algorítimo de EM fechando assim o ciclo da solução proposta, como é possível observar na Figura 7.

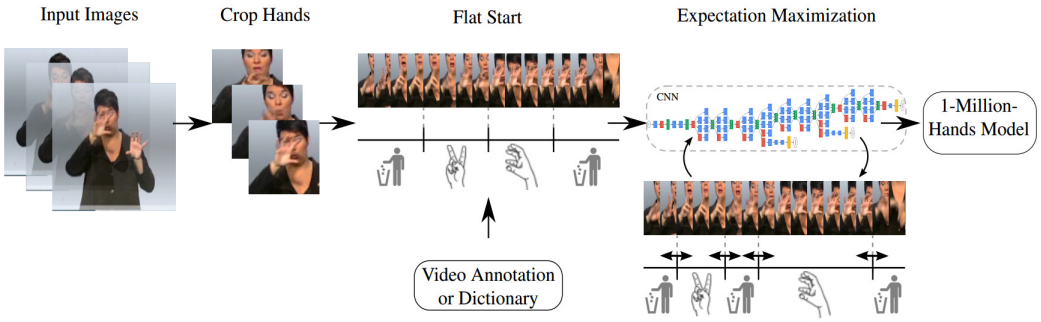


Figura 7 – Visão geral do algorítimo proposto. Extraído de (KOLLER; NEY; BOWDEN, 2016)

2.4 Arquitetura e Treinamento da Rede

A rede utilizada no artigo foi o modelo pré-treinado de 22 camadas vencedor da ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) de 2014. Denominada GoogLeNet ela é a rede que introduz o conceito de *Inception* utilizando uma série de módulos para diminuir o número de parâmetros livres da rede enquanto oferece um melhor desempenho do que o previamente experimentado (SZEGEDY et al., 2015).

2.4.1 Inception

O conceito de *Inception* se baseia no princípio que existe uma escala de correlação espacial entre os clusters resultantes das camadas de uma rede neural convolucional. Clusters correlacionados em camadas mais próximas à entrada ocupam uma região localizada espacialmente próxima. Já nas camadas mais afastadas da entrada os clusters que possuem correlações podem estar mais distantes espacialmente.

Esse pensamento pode ser alusionado ao entendimento que as características mais primitivas, como retas e curvas, que possuem relações entre si estão mais próximas umas das outras. Já em características mais complexas podemos ver que um objeto pode estar correlacionado com o outro mesmo estando espacialmente mais distantes.

Para atingir o objetivo de melhor modelar esse tipo de variação o conceito de *Inception* propõe uma arquitetura modular onde um mesmo módulo com pequenas variações é repetido multiplas vezes compondo as camadas da arquitetura. Para obedecer o princípio previamente mencionado esses módulos possuem quatro diferentes filtros. Os três primeiros sendo camadas convolucionais com tamanhos 1×1, 3×3 e 5×5. Essas camadas fazem o papel de relacionar tanto clusters proximamente localizados (camadas 1×1), quanto aqueles que estão mais esparsos (camadas 3×3 e 5×5). Além desses filtros também é adicionado uma camada de *max pooling* para manter uma capacidade de generalização que se mostra essencial para redes neurais convolucionais.

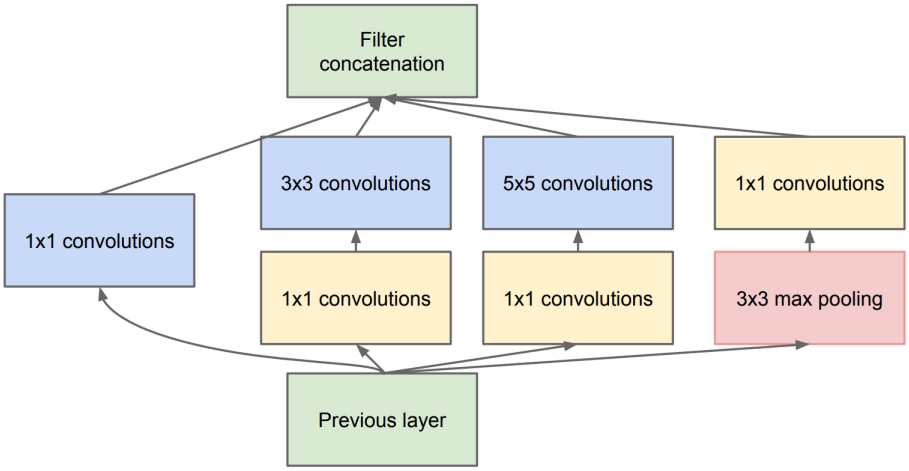


Figura 8 – Módulo *Inception* da GoogLeNet (SZEGEDY et al., 2015).

Além desses quatro filtros empregados em cada módulo existe também um outro tipo de camada que executa uma dupla função nesse conjunto. As camadas convolucionais de tamanho 1×1 em amarelo na Figura 8 são responsáveis por reduzir a dimensionalidade das entradas do módulo, pois se a saída de um módulo que realiza uma convolução 5×5 for passada como entrada para uma camada de convolução 5×5 pode gerar uma explosão computacional, aumentando drasticamente o número de parâmetros e limitando o número de módulos que podem ser empilhados. Além dessa importante função, as mesmas também são usadas como ReLUs introduzindo a não-linearidade à rede.

2.4.2 Treinamento

O treinamento da rede foi realizado a partir do modelo já treinado porém com a substituição das camadas totalmente conectadas da saída, cujas saídas representavam 1000 classes (de objetos do desafio ILSVRD 2014), por camadas totalmente conectadas inicializadas com zeros e possuindo um total de 61 saídas, relacionadas com as classes de *handshapes*.

O método de optimização utilizado para o treinamento foi o SGD e a função de *loss* é a *cross-entropy classification loss* baseada em *soft max*. A cada iteração do algorítimo proposto a rede foi treinada durante quatro *epochs* onde a taxa de aprendizado foi mantida como $lr = 0.0005$ nas três primeiras, sendo alterada para $lr = 0.00025$ para a última *epoch*. O termo *epoch* mencionado se refere a uma métrica de iterações de treinamento onde um *epoch* é referente ao número de iterações entre a uma passagem e outra de uma mesma imagem pela rede.

Uma consideração dos autores foi que opção por treinar toda a rede com a mesma taxa de aprendizado foi empiricamente mais proveitosa que treinar apenas as camadas substituídas ou amentar o peso da taxa de aprendizado dessas camadas.

2.5 Resultados

Os resultados apresentados nesse trabalho demonstram a eficácia do método em melhorar a precisão da rede neural convolucional à cada iteração do algorítimo proposto sendo alcançados no melhor conjunto de treinamento as precisões *top-1* de 62.8% e *top-5* de 85.6%. A precisão *top-1* se refere a porcentagem de acertos da classe rotulada com relação à saída de maior ativação da rede e a *top-5* se refere à porcentagem de vezes que a classe rotulada foi encontrada entre as 5 maiores ativações.

Para uma discriminação maior entre as classes também é disponibilizada a relação de confusão entre elas, disponíveis na Tabela 1, onde 13 classes dentre as classificadas são mostradas juntamente com suas precisões.

96.5	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0
2.0	90.1	0.8	0.1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	2.7	94.3	2.3	0.0	0.0	0.0	0.0	0.0	1.9	0.0	0.0	0.0
0.0	0.0	0.0	49.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	18.8	41.7	6.0	0.0	0.0	0.0	0.2	0.0	0.0	0.0
0.0	0.0	0.0	4.1	9.4	81.9	0.0	0.0	0.0	0.5	0.0	0.0	0.0
0.0	0.0	0.0	1.7	0.0	0.0	47.6	2.1	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	1.6	0.0	0.0	0.0	95.5	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	3.9	1.3	0.0	0.0	0.0
0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	64.9	0.0	0.0	0.0
0.0	0.0	0.0	3.5	38.1	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0
0.0	0.0	0.0	0.8	0.0	0.0	0.0	0.0	0.0	0.2	0.0	100.0	0.0
0.4	0.0	0.0	0.6	0.0	0.0	0.0	0.0	0.0	0.8	0.0	0.0	64.3

Tabela 1 – Confusão de classes da rede treinada mostrando a precisão por classe na diagonal, as classes verdadeiras no eixo y e as classes preditas no eixo x. (KOLLER; NEY; BOWDEN, 2016)

Os autores também discutem a provável causa da diferença entre as precisões de diferentes classes como visto na Tabela 1. Eles apontam como um dos possíveis motivos a grande diferença de amostras entre algumas classes e outras, o que é corroborado pela evidência que as classes com maior número de amostras possuem maiores precisões em geral.

Outro resultado demonstrado pelo artigo foi quanto à performance do classificador no reconhecimento contínuo de linguagem de sinais. Os autores fazem uso de um sistema de reconhecimento previamente publicado para comparar a performance da rede

treinada com um algorítimo (HoG-3D) utilizado por vários sistemas *state-of-the-art* no reconhecimento de linguagens de sinais. Os resultados são favoráveis à rede proposta pelo artigo onde ela atinge uma menor taxa de erro de palavras (50.2/12.0) com relação à taxa obtida pelo HoG-3D (58.1/12.5). Sendo o primeiro valor referente ao conjunto de testes do *dataset* RWTH-PHOENIX-Weather 2014 Multisigner e o segundo referente ao *dataset* SIGNUM.

3 Reprodução dos Resultados

Ainda considerando o trabalho de Koller *et al.* (KOLLER; NEY; BOWDEN, 2016) com foco na rede neural convolucional treinada através do método definido no artigo, foi realizada a reprodução do experimento de validação utilizando o modelo publicamente disponibilizado pelos autores e, juntamente com ele, o *dataset* de validação de 3359 imagens manualmente rotuladas também disponibilizado pelos mesmos.

O modelo utilizado foi uma segunda interação do publicado no artigo que foi disponibilizado posteriormente no *website* do projeto. Essa segunda versão do modelo possue ganhos de desempenho notáveis como a melhora da precisão *top-1* de 62.8% para 85.5% e do *top-5* de 85.6% para 94.8%.

3.1 Caffe framework

Para tal reprodução foi necessária a instalação e configuração do *framework* utilizado para o treinamento do modelo disponibilizado. Esse *framework* chamado *Caffe* é uma implementação eficiente e otimizada para execução paralela em GPUs de redes neurais convolucionais (JIA et al., 2014). Ele agrega diferentes implementações de camadas, métodos de otimização e demais ferramentas necessárias para a treinamento, avaliação, *benchmark* e teste desse tipo de modelo.

Esse *framework* suporta vários tipos de *backends* para a obtenção dos dados dentre os quais o LevelDB, o LMDB e o HDF5 são os mais populares. Durante os testes realizados foram utilizados os *backends* LevelDB (GHEMAWAT; DEAN, 2011) e o LMDB (CHU, 2011). Ambos são implementações que armazenam os dados em uma forma chave-valor como *arrays* de bytes arbitrários, esse armazenamento é ordenado pelo valor da chave. Esses *backends* proporcionam a rapidez necessária para fazer possível a alimentação da rede com os dados necessários em um tempo hábil, possibilitando tempos de treinamento e avaliação reduzidos.

3.2 Conjunto de validação

De posse do modelo treinado foi necessária a modificação das camadas de entrada da arquitetura da rede para receber o conjunto de imagens de validação. Esse *dataset* por sua vez é constituído de um conjunto de pastas contendo as imagens pre-processadas através do rastreamento e o recorte definidos no artigo. Juntamente com essas pastas também é disponibilizado um arquivo índice contendo os caminhos relativos das imagens

assim como seus respectivos rótulos.

Tanto tradicionalmente como para facilitar a implementação das redes neurais convolucionais, a codificação dos rótulos de um conjunto discreto de classes é um valor inteiro que corresponde à uma das classes. Essa relação é definida por um mapeamento pré-definido.

Um obstáculo encontrado durante a validação da rede em questão foi relacionado à uma codificação anômala dos rótulos do *dataset* disponibilizado. Esses rótulos estavam traduzidos para os nomes dos sinais definidos pela taxonomia da linguagem de sinais dinamarquesa, ao invés dos respectivos inteiros utilizados para o treinamento da rede. A princípio esse foi um fator limitante e motivo de investigação, pois a validação do conjunto de dados conforme disponibilizado resultava em uma precisão *top-1* da rede menor que 35%, resultado que diverge vastamente do anunciado.

Juntamente com este imprevisto outras questões foram levantadas como possíveis suposições para a fonte do problema. Uma delas foi a possibilidade de uma incompatibilidade quanto aos *backends* de armazenamento dos dados para a validação, outra se referia à necessidade de se utilizar a mesma versão de *framework* onde foi o modelo treinado, assim como outras dúvidas menos pertinentes.

Para resolução dessas questões que limitavam o andamento da reprodução foi estabelecida a comunicação com um dos autores do artigo, o pesquisador Oscar Koller. Em suas respostas Koller levantou novas questões e disponibilizou novamente o conjunto de validação, desta vez já encapsulado num banco de dados *LevelDB*.

De posse desse banco de dados com as imagens pré-populadas foi possível verificar a validade dos resultados apresentados no artigo, atingindo os 85.5% de precisão no conjunto de validação.

Através do sucesso obtido com o *dataset* já encapsulado, foi identificado o motivo pelo qual a validação não correspondia ao esperado quando o banco de dados era manualmente populado com as imagens disponibilizadas publicamente. O problema se dava por conta da codificação dos rótulos empregada nesse *dataset*, como previamente descrito. Dessa forma foi necessária a criação de um *script* de conversão do arquivo índice, que mapeia os caminhos e os rótulos de cada imagem do conjunto.

A Tabela 3 em anexo foi utilizada para converter o arquivo índice para a notação dos rótulos usando inteiros[1]. As ferramentas utilizadas para realizar a validação e demais testes apresentados por este trabalho estão disponibilizados no repositório[2] do projeto.

[1] O *script* que realiza essa conversão pode ser encontrado em <https://gitlab.com/andrebsguedes/gesture-cnn/blob/master/src/convert_mapping.py>

[2] Disponível em <https://gitlab.com/andrebsguedes/gesture-cnn>

3.3 Experimentos realizados

Usando o modelo treinado e com os rótulos propriamente mapeados algumas considerações foram feitas quanto à essa nova versão do modelo disponibilizado. Uma das observações, através de testes singulares com imagens manualmente anotadas, mostrou uma provável dependência do classificador quanto à escala da mão na imagem, porém, pelo baixo numero de amostras do teste o resultado não é conclusivo.

Um experimento simples de variação de cores também foi realizado para entender qual era a resposta da rede à mudança de cores, visto que as cores nas imagens dos *datasets* de treinamento não constavam grandes diferenças entre imagens. Ainda usando algumas poucas amostras manualmente anotadas, não foi constatada nenhuma preferência quanto as cores do ambiente. Exemplos desses experimentos realizados podem ser observados na Figura 9.

Figura 9 – Ilustração dos experimentos realizados.

Outro experimento utilizando uma imagem manualmente recortada demonstra uma possível dificuldade da rede em detectar a classe correta quando à escala da mão na imagem não corresponde à observada nos *datasets* de treinamento.

Para comparação do desempenho da nova rede disponibilizada e a rede apresentada no artigo, a tabela de confusão de classes foi reproduzida, como visto na Tabela 2. A reprodução apresenta as classes da Tabela 1 em igual disposição.

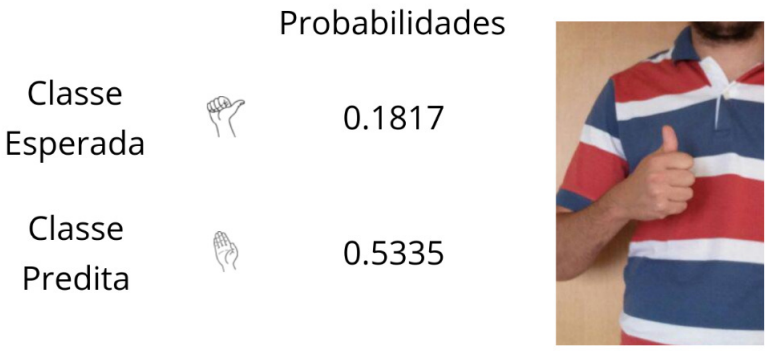


Figura 10 – Experimento onde a escala da mão é variada. As probabilidades são referentes à saída da rede

É possível perceber através da comparação entre as duas redes que houve uma melhora significativa na maioria das classes apresentadas, porém também se nota que algumas precisões diminuíram. Isso pode ser um indício de uma maior distribuição dos pesos entre as diversas classes.

Quanto à forma específica de treinamento do segundo modelo algumas informações podem ser inferidas. Segundo os parâmetros disponibilizados juntamente com o modelo pode se observar que a taxa de aprendizado foi mantida, foi aplicado um momento de 0.9 e uma regularização L2 com força 0.0005 foi utilizada.

Além disso é seguro afirmar que a arquitetura da rede não foi modificada significantemente. Dessa forma, pra entender o motivo de tal variação na precisão da rede (de 62.8% para 85.5%), é necessária uma investigação junto aos autores do trabalho, pois as informações existentes não apontam nenhum aspecto visivelmente responsável pela variação dos resultados.

												
98.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.1	0.0	0.0	0.0	0.5
0.0	89.9	0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.6	89.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.6	91.5	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	80.5	18.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.8	87.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	94.7	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	5.3	98.2	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.3	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	86.8	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	5.3	0.0	0.0	0.0	0.0	92.9	0.0
0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.0	54.5

Tabela 2 – Confusão de classes do novo modelo obtida através da reprodução.

4 Conclusão

No curso desse trabalho foram introduzidos o conceito e o funcionamento de redes neurais convolucionais, assim como uma revisão do estado-da-arte no reconhecimento de *handshapes* usando esse tipo de rede. Além disso foram reproduzidos os resultados apresentados por (KOLLER; NEY; BOWDEN, 2016), um artigo de referência no uso de redes neurais para esse tipo de reconhecimento.

Com o conhecimento adquirido sobre redes neurais convolucionais é possível entender o motivo pela sua alta capacidade discriminativa, assim como perceber como a variação de diversos parâmetros tem um impacto direto no desempenho desse tipo de rede e quais são os comportamentos esperados.

Esta introdução se fez necessária para entender as consequências relativas as escolhas feitas pelos autores do trabalho revisado, assim como evidenciar a importância do aumento no volume de dados rotulados proposto pelo artigo e também para obter uma maior compreensão dos passos para a reprodução dos resultados assim como os obstáculos discutidos.

A revisão da literatura apresentada demonstra a possibilidade da utilização de fontes ruidosas de dados, como vídeos ambiguamente anotados, para o treinamento de um classificador de *frames*. Também foi observado o desempenho desse tipo de classificador no contexto específico de reconhecimento de *handshapes*, evidenciando a viabilidade do uso de redes neurais convolucionais para este fim.

Quanto a reprodução dos resultados é possível identificar empiricamente as robustez e estabilidade do modelo treinado mesmo quando mudanças no ambiente de execução são realizadas. Também é notável a necessidade da disponibilização do mapeamento correto entre os rótulos legíveis dos dados e seus respectivos inteiros.

O sucesso da reprodução indica uma linha de base confiável para a exploração do problema de reconhecimento de *handshapes* à partir do modelo disponibilizado.

4.1 Trabalhos futuros

Possíveis sequências à este trabalho incluem:

- Uma investigação mais profunda da viabilidade da utilização do classificador em contextos menos controlados, onde variações do ambiente, das câmeras e dos sujeitos possam introduzir um maior ruído às amostras.

- O desenvolvimento de um algorítimo de rastreamento de mãos em tempo real para teste e utilização do classificador em uma maior variedade de contextos.
- A validação do classificador utilizando diferentes recortes e escalas para verificar a robustez do mesmo a esse tipo de perturbação, com a possibilidade de um refinamento da rede para que generalize sobre esse tipo de entrada.

Baseando-se nessas propostas o planejamento para o segundo trabalho foi realizado. Nesse planejamento a construção do rastreador em tempo real é tido como prioridade, pois se faz necessário para as demais investigações. O prazo estimado para a construção do mesmo é de 2 meses à partir do princípio do semestre. A segunda prioridade é a validação do classificador quanto a escalas e recortes, pois, se houver necessidade de refinação da rede, mais tempo pode ser demandado. Fechando o escopo pretendido, a terceira proposta será abordada por um prazo de um mês, culminando com um prazo dedicado à finalização do texto do trabalho, que também tem a duração prevista de um mês. Nesse período, também está prevista a preparação de um artigo científico descrevendo resultados, a ser submetido para uma conferência tal como SIBGRAPI.

z

Referências

AHN, S. H. *Convolution*. 2012. Disponível em: <http://www.songho.ca/dsp/convolution/convolution.html>. Citado na página 19.

APHEX34. *Input volume connected to a convolutional layer*. 2015. Licencianda sob CC-BY-SA-4.0. Disponível em: <https://commons.wikimedia.org/wiki/File:Conv_layer.png>. Citado 2 vezes nas páginas 9 e 20.

BENGIO, Y.; BOULANGER-LEWANDOWSKI, N.; PASCANU, R. Advances in optimizing recurrent networks. In: IEEE. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. [S.l.], 2013. p. 8624–8628. Citado na página 25.

CHU, H. Mdb: A memory-mapped database and backend for openldap. In: *Proceedings of the 3rd International Conference on LDAP, Heidelberg, Germany*. [S.l.: s.n.], 2011. Citado na página 35.

DEMPSTER, A. P.; LAIRD, N. M.; RUBIN, D. B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, JSTOR, p. 1–38, 1977. Citado na página 29.

DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, v. 12, n. Jul, p. 2121–2159, 2011. Citado na página 25.

GHEMAWAT, S.; DEAN, J. *LevelDB*. 2011. Citado na página 35.

HUBEL, D. H.; WIESEL, T. N. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, Wiley Online Library, v. 195, n. 1, p. 215–243, 1968. Citado na página 18.

JIA, Y. et al. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. Citado na página 35.

KARPATHY, A. *Illustration of spatial arrangement.* 2015. Licenciada sob MIT. Disponível em: <https://github.com/cs231n/cs231n.github.io/blob/master/assets/cnn/stride.jpeg>. Citado 2 vezes nas páginas 9 e 20.

KARPATHY, A. *CS231n Convolutional Neural Networks for Visual Recognition*. 2016. Disponível em: <http://cs231n.github.io/>. Citado 3 vezes nas páginas 22, 24 e 26.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. Disponível em: <http://arxiv.org/abs/1412.6980>. Citado na página 25.

KOLLER, O. *Deep Hand: How to Train a CNN on 1 Million Hand Images When Your Data Is Continuous and Weakly Labelled*. 2016. Disponível em: <http://www-i6.informatik.rwth-aachen.de/~koller/1miohands/>. Citado na página 47.

KOLLER, O.; NEY, H.; BOWDEN, R. Deep hand: How to train a cnn on 1 million hand images when your data is continuous and weakly labelled. In: *IEEE Conference on Computer Vision and Pattern Recognition*. Las Vegas, NV, USA: [s.n.], 2016. p. 3793–3802. Citado 9 vezes nas páginas 5, 7, 9, 27, 29, 30, 32, 35 e 41.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, v. 5, n. 4, p. 115–133, 1943. ISSN 1522-9602. Disponível em: <http://dx.doi.org/10.1007/BF02478259>. Citado na página 17.

OBERWEGER, M.; WOHLHART, P.; LEPETIT, V. Hands deep in deep learning for hand pose estimation. *arXiv preprint arXiv:1502.06807*, 2015. Citado na página 27.

POTAMIAS, M.; ATHITSOS, V. Nearest neighbor search methods for handshape recognition. In: ACM. *Proceedings of the 1st international conference on PErvasive Technologies Related to Assistive Environments*. [S.l.], 2008. p. 30. Citado na página 27.

ROBERTS, E. *Neural Networks*. 2000. Disponível em: <http://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/index.html>. Citado 2 vezes nas páginas 17 e 18.

SRIVASTAVA, N. et al. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, v. 15, n. 1, p. 1929–1958, 2014. Citado 2 vezes nas páginas 9 e 26.

SZEGEDY, C. et al. Going deeper with convolutions. In: *Computer Vision and Pattern Recognition (CVPR)*. [s.n.], 2015. Disponível em: <http://arxiv.org/abs/1409.4842>. Citado 3 vezes nas páginas 9, 30 e 31.

WIKIPEDIA. *Machine learning — Wikipedia, The Free Encyclopedia*. 2017. [Online; accessed 16-January-2017]. Disponível em: <https://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=759789517>. Citado na página 16.

Anexos

ANEXO A – Tabela de mapeamento das classes

#	Descrição	#	Descrição
1	1	31	l_hook
2	2	32	middle
3	3	33	m
4	3_hook	34	n
5	4	35	o
6	5	36	index
7	6	37	index_flex
8	7	38	index_hook
9	8	39	pincet
10	a	40	ital
11	b	41	ital_thumb
12	b_nothumb	42	ital_nothumb
13	b_thumb	43	ital_open
14	cbaby	44	r
15	obaby	45	s
16	by	46	write
17	c	47	spoon
18	d	48	t
19	e	49	v
20	f	50	v_flex
21	f_open	51	v_hook
22	fly	52	v_thumb
23	fly_nothumb	53	w
24	g	54	y
25	h	55	ae
26	h_hook	56	ae_thumb
27	h_thumb	57	pincet_double
28	i	58	obaby_double
29	jesus	59	m2
30	k	60	jesus_thumb

Tabela 3 – Mapeamento de inteiros para rótulos segundo a taxonomia da linguagem de sinais dinamarquesa. (KOLLER, 2016)