UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Caio Kioshi Miyazaki

Redes neurais convolucionais para aprendizagem e reconhecimento de objetos 3D

São Carlos

2017

Caio Kioshi Miyazaki

Redes neurais convolucionais para aprendizagem e reconhecimento de objetos 3D

Monografia apresentada ao de Curso de Engenharia Elétrica com Ênfase em Sistemas de Energia e Automação, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Vitor Campanholo Guizilini

São Carlos

2017

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Miyazaki, Caio Kioshi
M685r Redes neurais convolucionais para aprendizagem e reconhecimento de objetos 3D / Caio Kioshi Miyazaki; orientador Vitor Campanholo Guizilini. São Carlos, 2017.

Monografia (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2017.

1. Aprendizado profundo. 2. Redes neurais convolucionais. 3. Aprendizado de máquina. 4. Objetos 3D. 5. Nuvem de pontos. 6. Voxel. 7. ModelNet10. I. Título.

FOLHA DE APROVAÇÃO

Nome: Caio Kioshi Miyazaki

Título: "Redes neurais convolucionais para aprendizagem e reconhecimento de objetos 3D"

Trabalho de Conclusão de Curso defendido e aprovado em 29 / 11 / 17,

com NOTA 9,5 (Nove e Meio), **pela Comissão Julgadora:**

Prof. Dr. Vitor Campanholo Guizilini - Orientador -
The University of Sydney

Prof. Dr. Valdir Grassi Júnior - SEL/EESC/USP

Prof. Dr. Fernando Santos Osório - SSC/ICMC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

Este trabalho é dedicado aos meus pais,
aos meus irmãos e minha namorada.

AGRADECIMENTOS

Aos meus pais, Celso e Ester, e meus irmãos, Caroline e Christian, por todo o apoio, carinho e suporte em toda minha vida. Sempre me incentivando à aprender e estudar.

Gostaria de agradecer à minha amada namorada e melhor amiga Nathalia, pelo companheirismo, conselhos e apoio. Apesar de estar passando por um período muito atarefado, realizando estágio e faculdade, conseguiu dar tempo para me apoiar neste trabalho.

Ao Vitor Campanholo Guizilini, pela imensa contribuição e orientação durante todas as etapas deste trabalho. Sempre me incentivando e me fascinando pela área de Aprendizado de Máquinas.

Ao professor Valdir Grassi Junior, pelo apoio e paciência devotada a mim, tornando possível a realização deste projeto.

Aos todos amigos que fiz durante esta parte da minha jornada.

À Escola de Engenharia de São Carlos, todos os professores e técnicos que contribuíram durante minha formação.

"A morte do homem começa no instante
em que ele desiste de aprender ..."
Albino Teixeira

RESUMO

MIYAZAKI, C. K. **Redes neurais convolucionais para aprendizagem e reconhecimento de objetos 3D**. 2017. 56p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2017.

O reconhecimento de objetos tridimensionais é um problema ainda pouco explorado, porém de extrema importância em diversas áreas da visão computacional, evolvendo da medicina até a robótica. Este trabalho propõe a utilização de Redes Neurais Convolucionais para a identificação de 10 classes de objetos 3D. Para isto foi utilizado a base de dados de modelos CAD 3D, o ModelNet10. Utilizando a representação dos objetos em nuvem de pontos uma série de topologias foram determinadas variando os parâmetros da rede. Cada topologia após configurada foi treinada e testada com conjunto de exemplos diferentes. Como resultado, é discutido a importância da escolha de cada parâmetro de uma Rede Neural Convolucional e sua influência no desempenho final. A topologia com o melhor resultado obteve 89% de acurácia no conjunto de teste.

Palavras-chave: Aprendizado Profundo. Objetos 3D. Redes Neurais Convolucionais. Aprendizado de Máquina. Nuvem de Pontos. Voxel ModelNet10

ABSTRACT

MIYAZAKI, C. K. **Convolutional neural network for learning and recognition of 3D objects**. 2017. 56p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2017.

The 3D objects recognition is a problem that has not been well explored yet, but it is extremely important in several fields of computer vision, it has been used in medicine to robotics. This paper uses Convolutional Neural Network for the identification of 10 classes of 3D objects. The ModelNet10 was the 3D CAD model dataset used. Using point cloud as the object representation, a set of topologies were defined by varying the parameters of the network. Each topology configuration was trained and tested with separated example of the dataset. As a result, the importance of selecting each parameter of the Convolutional Neural Network corretly and its influence on final performance is discussed. The topology with best result obtained 89% of accuracy on the test set.

Keywords: Machine Learning. Convolutional Neural Network. 3D Object. Deep Learning. Point Cloud. ModelNet10

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

TCC	Trabalho de Conclusão de Curso
USP	Universidade de São Paulo
GPU	Unidades de Processamento Gráfico (*Graphic Processing Unit*)
IA	Inteligência Artificial
ML	Aprendizado de Máquina (*Machine Learning*)
DL	Aprendizado Profundo (*Deep Learning*)
2D	Bidimensional
3D	Tridimensional
RNA	Rede Neural Artificial
MLNN	Rede Neural Multicamadas (*Multi-layer Neural Network*)
LR	Taxa de Aprendizagem (*Learning Rate*)
GD	Gradiente Descendente (*Gradient Descendent*)
SGD	Gradiente Descendente Estocástico (*Stochastic Gradient Descent*)
CNN	Redes Neurais Convolucionais (*Convolutional Neural Network*)
CL	Camada Convolucional (*Convolutional Layer*)
FCL	Camada Totalmente Conectada (*Fully Connected Layer*)

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

Os desafios mais recentes que a Inteligência Artificial tenta resolver são problemas com soluções intuitivas. Em outras palavras, são tarefas executadas de modo fáceis por pessoas, porém difíceis de serem descritas formalmente como o reconhecimento da fala e imagens. A intuição humana está relacionada com a habilidade de entender e interpretar conhecimentos passados para predizer e ou resolver problemas atuais. Com isso surgiu o Aprendizado de Máquina (do termo em inglês *Machine Learning* - ML), uma subárea da IA, que utiliza métodos computacionais para emular esse processo de intuição humana. Em ML, a aprendizagem é feita por meio de treinamentos em banco de dados, que representam eventos e experiências passadas, possibilitando a construção de sistemas capazes de aprender de forma automática (DUNDAS; CHIK, 2011).

Desde sua concepção, ML têm contribuído para o avanço de diversas áreas do conhecimento. Atualmente, a Visão Computacional foi a que mais se beneficiou com o surgimento do método, sendo um dos campos que vêm ganhado grande importância, tanto pelo desenvolvimento quanto pela forma como as informações estão sendo expostas. Sua importância pode ser vista em um estudo da Cisco, em que estima que em 2016 mais de 85 porcento de todo o tráfico da internet está na forma de *pixels* (LI, 2016).

Neste panorama geral da Visão Computacional e Aprendizado de Máquina, que surgiu a motivação deste trabalho. Reconhecimento de imagens são problemas que até pouco tempo atrás eram impossíveis de serem resolvidos pelas máquinas. Mas hoje, dada sua importância, grandes esforços estão sendo direcionados no desenvolvimento de novas técnicas para a aplicação em diversas áreas: desde imagens médicas, carros autônomos, automação industrial, realidade aumentada, etc.

1.2 Motivação

Reconhecimento de objetos continua sendo um dos desafios fundamentais de sistemas de visão computacional. Atualmente muitos esforços estão concentrados na classificação de categorias de objetos 2D. No entanto, há um constante crescimento na utilização de formas 3D, fazendo necessário o desenvolvimento de novos métodos para classifica-los. Um dos problemas mais recorrentes nestas formas é a obtenção de uma boa representação no espaço tridimensional, interferindo diretamente no reconhecimento da imagem. A utilização de Redes Neurais Convolucionais têm como objetivo suprir, em partes, esta necessidade de boa representações.

1.3 Objetivo Geral

Este trabalho tem como objetivo utilizar redes neurais convolucional para identificação de objetos 3D a partir do banco de dados fornecidos pelo projeto Princeton ModelNet10.

1.4 Objetivo Específico

- Estudar trabalhos correlatos e analisar o estado da arte;
- Entender o funcionamento de uma rede neural convolucional e suas diferentes topologias;
- Realizar o treinamento e aplicação de uma rede neural artificial para reconhecimento de objetos 3D.

1.5 Organização do Trabalho

O Capítulo 1 apresenta uma contextualização ao tema de identificação de objetos tridimensionais, trata da motivação e objetivos do trabalho, e por fim apresenta a estrutura organizacional do trabalho.

O Capítulo 2 apresenta uma breve revisão bibliográfica sobre representações tridimensionais e os métodos existentes de identificação de objetos 3D.

O Capítulo 3 abrange algumas áreas de Aprendizado de Máquina (ML) utilizadas neste projeto. Os fundamentos teóricos de Redes Neurais Artificiais e seus novos métodos, Aprendizado Profunda e Redes Neurais Convolucionais.

O Capítulo 4 descreve os materiais utilizados e a metodologia desenvolvida, baseada em aspectos teóricos e práticos.

O Capítulo 5 apresentada os resultados e análises da fase de pré-processamento dos objetos 3D, e da fase de treinamento das Redes Neurais Convolucionais.

Por fim, o Capítulo 6 traz conclusões do trabalho, e considerações acerca de trabalhos futuros.

2 RECONHECIMENTO DE OBJETOS 3D

Este capítulo apresenta uma breve revisão bibliográfica sobre representações tridimensionais e os métodos existentes de identificação de objetos 3D.

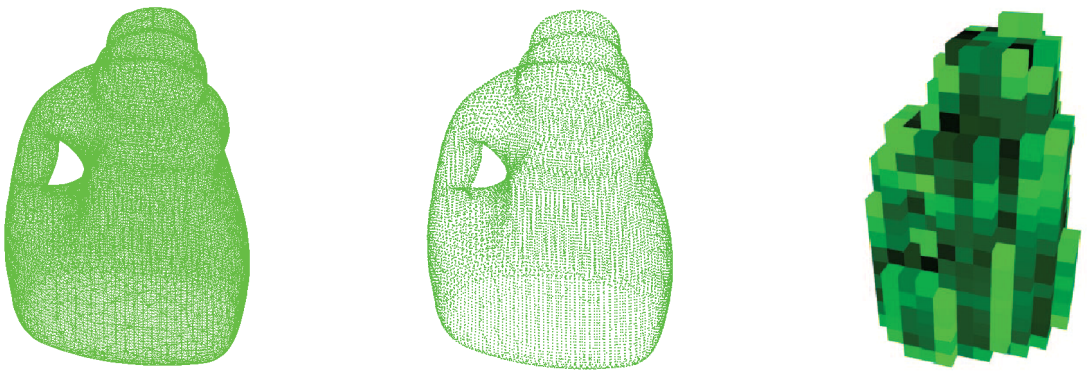
2.1 Representações Tridimensionais

Representações tridimensionais de objetos e cenários são cruciais para análises visuais do mundo, pois retratam de forma mais fiel e compacta quando comparada com a representação bidimensional (KRAUSE et al., 2013). Entretanto, são atualmente subutilizadas em sistemas de visão computacional, principalmente por causa da falta de boas representações genéricas (WU et al., 2015).

Modelos 3D podem ser divididos em duas categorias: sólidos e superfície. Modelos sólidos definem volumes dos objetos, são os mais realísticos porém são mais difíceis de construir. Comumente utilizados em simulações médicas e em engenharia, compõe as seguintes representações: *voxel*, BSP tree, CSG, Sweep e Octree. Modelos de superfície são mais fáceis de modelar pois representam somente a casca do objeto. Dentre alguns exemplos destas representações temos: nuvem de pontos, superfícies poligonais e *mesh*.

As três representações mais utilizados para problemas de reconhecimento de objetos 3D encontrados na literatura são: nuvem de pontos, *voxel* e *mesh*. Neste trabalho serão abordados a representação *mesh*, pois o banco de dados utilizado possui os objetos neste formato, a representação nuvem de pontos, que é a representação intermediária para o processo de discretização da representação e por fim é utilizada a representação *voxel*.

Figura 1: Exemplo de Representações 3D



(a) Mesh (b) Nuvem de pontos (c) voxel

Fonte: Garcia-Garcia et al. (2016)

2.1.1 Representação Mesh

Mesh são malhas poligonais, que são compostas por um conjunto de vértices e arestas. A conexão entre eles formam as faces, ou polígonos, que compõem a superfície da imagem. Os polígonos mais comuns utilizados são os triângulos e quadriláteros. Na Figura 1 exemplo (a) é possível observar o conjunto de pequenas faces, que ligadas foram a superfície do objeto.

2.1.2 Representação em Nuvem de Pontos

A nuvem de pontos é uma coleção de pontos colocados em um sistema de coordenadas tridimensional, em geral, adquiridos de scanners ou da conversão de outras representações 3D. Este tipo de dados são frequentemente encontrados em várias aplicações desde robótica e visão até cosmologia (RAVANBAKHSH; SCHNEIDER; POCZOS, 2016). A Figura 1 exemplo (b) ilustra esta representação que foi produzida a partir do exemplo (a).

2.1.3 Representação Voxel

Assim como o *pixel* sugere um elemento da imagem em um espaço bidimensional, o *voxel* analogamente sugere um elemento de volume em um espaço tridimensional. A representação em *voxel* consiste em decompor o objeto 3D em células em forma de cubos igualmente espaçados em uma grade (comumente referido em inglês como *grid*). A Figura 1 exemplo (c) mostra um exemplo desta representação em um cubo de 30x30x30 *voxels*. Cada *voxel* possui uma densidade diferente, nos quais graficamente as cores mais escuras indicam alta densidade e as cores mais claras, baixa densidade.

Apesar da representação *voxel* possuir propriedades interessantes, ela tende a ser computacionalmente mais custosa em termos de memória em comparação à nuvem de pontos (SIMONOVSKY; KOMODAKIS, 2017). Porém este custo traz resultados significativos, dado que trabalhos no estado da arte em reconhecimento de objetos 3D abordam a utilização desta representação.

2.2 Métodos de reconhecimento de padrões 3D

Uma das primeiras abordagens encontradas em reconhecimento de objetos se baseia na criação de modelos matemáticos para os objetos. O modelo *spline* é um exemplo desta abordagem, no qual o objeto pode ser deformando localmente e não afetar o resultado. A criação de modelos matemáticos necessitam de muitos parâmetros, tornando uma técnica muito lenta e de difícil implementação. São encontradas em aplicações cujos objetos possuem poucos parâmetros e compactamente caracterizado, em especial reconhecimento de modelos cilíndricos, esferas, pirâmides e prismas (BENVEGNÙ, 2017).

O método *Scale-Invariant Feature Transform* (SIFT) proposto por Lowe (1999), marcou como um grande avanço para o reconhecimento de objetos. Esta técnica, comumente utilizada em imagens 2D é invariante à escala e rotação, com isso pode ser utilizado também em imagens 3D, apesar de possuir resultados pouco expressivos. Basicamente é composta por duas partes: o detector e o descritor. O detector é utilizado para identificar os extremos da imagem e utilizar a localização como pontos-chaves. O descritor define a orientação e descrição destes pontos-chave (BATISTA, 2012). O algoritmo SIFT extrai as características das imagens e transforma em vetores com características locais, e então estes são usados para fazer correspondência entre as imagens e classificá-las.

Em Osada et al. (2002) é proposto um método que calcula as assinaturas dos objetos utilizando modelos 3D poligonais arbitrários. Este método tem como objetivo reduzir problemas de comparações das formas 3D, utilizando comparações com distribuição probabilística, que são mais simples e não necessitam de métodos como registro de postura, correspondência das características ou correspondência de modelos. O método realiza a amostragem e normaliza o objeto, seguido da transformação do modelo arbitrário 3D em uma função parametrizada, esta função pode então ser facilmente comparada com outras funções.

Máquinas de Vetores de Suporte (do inglês *Support Vector Machine* - SVM) é um classificador binário baseado na aprendizagem estatística. É um classificador muito utilizado em classificação de padrões, reconhecimento de imagens, seleção de genes, e outros. O SVM tenta estabelecer uma série de princípios para a obtenção de uma boa generalização (LORENA; CARVALHO, 2007).

Por fim, o método de redes neurais artificiais é o mais amplamente utilizado hoje, possui boa capacidade de generalização com aprendizagem de padrões e vêm apresentando grandes resultados em tarefas de reconhecimento de objetos. Este método é utilizado neste projeto e será discutido mais profundamente no capítulo 3.

3 APRENDIZADO DE MÁQUINA

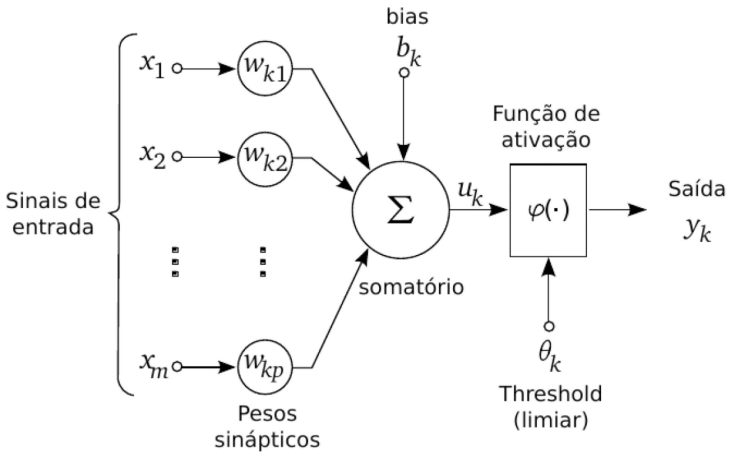
Este capítulo têm como objetivo abranger algumas áreas de Aprendizado de Máquina (ML) utilizadas neste projeto. Iniciando com uma breve introdução sobre Redes Neurais Artificiais, mostrando algumas técnicas e termos, como por exemplo, regressão logística, função *Softmax*, codificação *one-hot* e *Cross entropy*. Em seguida, será apresentado os fundamentos teóricos de Aprendizagem Profunda (DL) e Redes Neurais Convolucionais (CNN), com os principais métodos utilizados e seus benefícios.

3.1 Redes Neurais Artificiais

Redes Neurais Artificiais (RNA) são modelos computacionais de aprendizagem de máquina inspirados em redes neurais biológicas, que originaram em 1943, no trabalho de McCulloch e Pitts (1943). RNAs são muito utilizados em tarefas de reconhecimento de padrões, como por exemplo, reconhecimento de fala, reconhecimento de objetos, identificação de células cancerígenas, e outros (HAFEMANN, 2014).

Assim como uma rede neural biológica, as unidades básicas da RNA são os neurônios. Cada neurônio artificial pode ser representado graficamente como na Figura 2.

Figura 2: Neurônio Artificial



Fonte: (SILVA; SCHIMIDT, 2016)

Onde cada elemento possui as seguintes funções:

a) Sinais de entrada (x_m): conjunto de dados que servirão de base para o treinamento da rede.

b) Pesos Sinápticos (w_{kp}): cada sinal de entrada é associado à um peso diferente, de forma que durante o treinamento a RNA determina a influência da entrada

para o resultado final.

c) limiar de ativação (b_k): parâmetro que aumenta os graus de liberdade, permitindo melhor adaptação da rede.

d) Somatório: realiza a soma de todas entradas multiplicadas pelos pesos.

e) Função de ativação ($\varphi(.)$): aplica uma não-linearidade no valor do neurônio e determina a forma como ele deverá ser ativado.

f) Saída (y_k): resultado estimado pelo neurônio

Matematicamente, o neurônio artificial pode ser expresso por uma função com as variáveis de entrada x_m e saída y_k, como descrito pela Equação 3.1:

$$y_k = \varphi(\sum_{i=1}^{m} x_i w_{ki} + b_k) \tag{3.1}$$

3.1.1 Função de Ativação

As funções de ativação são funções não-lineares conectadas ao final da estrutura de um neurônio artificial (Figura 2), também são inspiradas biologicamente e definem a saída com base nos dados de entrada e o limiar de ativação.

Figura 3: Função Tangente Hiperbólica

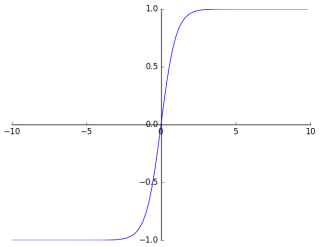


Figura 4: Função Sigmóide

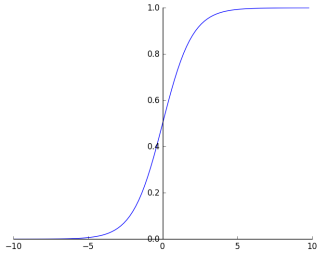
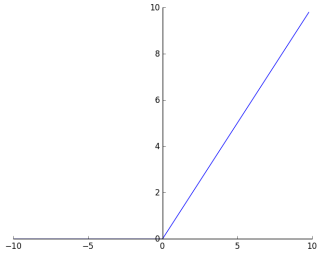


Figura 5: Função ReLU



Exemplos de funções de ativação - Fonte: Ferreira (2017)

A função sigmóide, Figura 4, é uma função do tipo:

$$\varphi(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

Muito utilizada em problemas de classificação, tem como saída valores entre 0 e 1, e resulta na probabilidade dos dados de entrada estarem contidos na classe analisada.

A função tangente hiperbólica, Figura 3, tem como saída valores entre -1 e 1 e é baseada na função sigmóide:

$$tanh(x) = 2\varphi(2x) - 1 \tag{3.3}$$

3.1.1.1 Função ReLU

Função Linear Retificada, do inglês *Rectified Linear Unit* (ReLU), é uma função de ativação mais eficiente que a sigmóide e tangente hiperbólica, pois não faz uso de expoentes. A Figura 5 representa esta função, onde é possível observar que é uma função linear por partes.

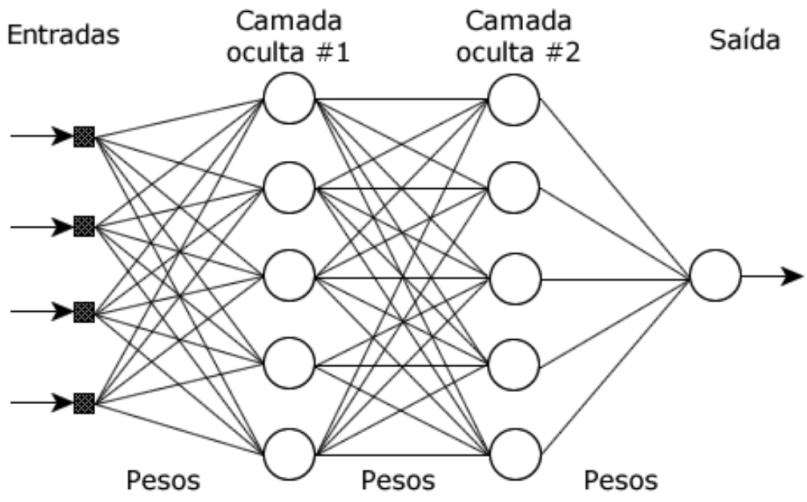
$$\varphi(x) = max(0, x) \tag{3.4}$$

3.1.2 Rede Neural Multicamadas

Um modelo básico de rede de apenas um neurônio é chamado Perceptron, ele é capaz de classificar padrões, porém de forma limitada, pois agrupa apenas dados linearmente separáveis. Como grande parte dos problemas existentes não são linearmente separável surgiu uma arquitetura mais robusta chamada Rede Neural Multicamadas (*Multi-layer Neural Network - MLNN*)(HAFEMANN, 2014).

MLNN é uma generalização do perceptron, no qual conjuntos de neurônios artificiais são distribuídos em camadas. Cada camada pode ser nomeada como entrada, oculta (também chamada de escondida) ou saída. A Figura 6 demonstra como cada perceptron é definido como um nó, e a distribuição das camadas, sendo a primeira camada chamada de entrada, a última de saída, e todas as camadas intermediárias são as ocultas.

Figura 6: Rede Neural Multicamadas



Fonte: (ALMEIDA,)

3.1.3 Função Softmax

Como explicado em Goodfellow, Bengio e Courville (2016), funções *Softmax* são comumente utilizados como classificadores na camada de saída, têm como objetivo representar a probabilidade de cada classe para cada valor de entrada.

Pela Equação 3.5 da Função *softmax* podemos notar que os resultados só podem assumir valores entre 0 e 1, e que a soma da probabilidade de todas as classes é igual a 1. Desta forma podemos determinar que a classe estimada pela rede é a que possui a maior probabilidade.

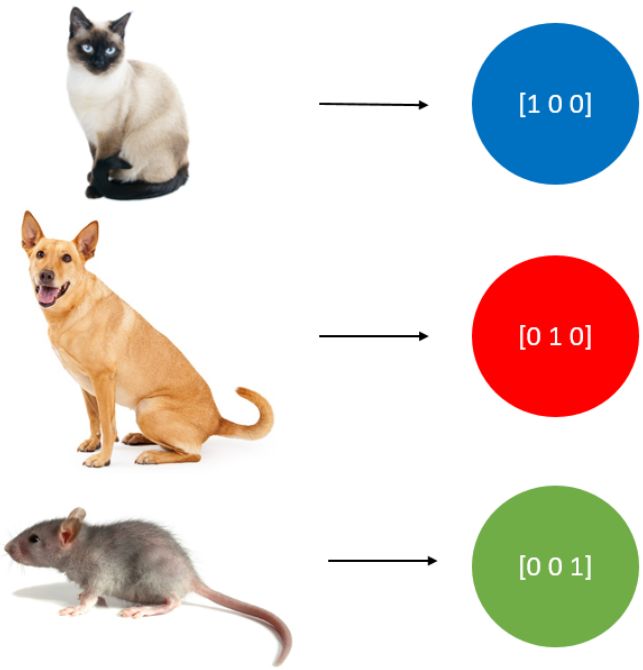
$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{3.5}$$

3.1.4 Codificação One-hot

Algoritmos de redes neurais artificiais possuem como entrada e saída valores numéricos, portanto é necessário converter os rótulos de cada classe de objetos de modo que a RNA consiga identificar na saída da rede.

Existem diversas formas de converter os rótulos das classes. Neste trabalho iremos utilizar a codificação *one-hot* que é o processo que converte os rótulos em vetores binários. Um exemplo desta conversão está representada na figura 7. Cada uma das classes: cachorro, gato e rato, possuem um vetor único.

Figura 7: One-hot encoding



Fonte: Elaborado pelo autor

3.1.5 Função Custo

A função custo (em inglês *loss*) têm como objetivo parametrizar o quão longe a rede está do resultado esperado. Ela é a média das distâncias dos vetores de saída e saída

desejada de toda a rede. Como podemos observar na equação do custo 3.6, o cálculo é através da média de cada *Cross Entropy*.

$$L = \frac{1}{N} \sum_{i} D(S(wx_i + b), L_i) \tag{3.6}$$

3.1.6 Cross Entropy

Cross Entropy é o método que mede a distância entre os valores da codificação *one-hot* com a saída da rede. Na equação 3.7 o parâmetro *Cross Entropy* é representado por "D", "S" é o vetor de saída do *Softmax* e "L" é o vetor da codificação *one-hot*.

$$D(S, L) = -\sum_{i}^{n} L_i log(S_i) \tag{3.7}$$

3.1.7 Método do Gradiente Descendente

O gradiente descendente (ou *Gradient Descent* - GD em inglês) é um método de otimização que tenta minimizar o erro da rede neural. Esta minimização é realizada modificando os pesos e os limiares de ativação com o objetivo de encontrar o mínimo local da função perda. As equações 3.8 e 3.9 determinam o modo como os pesos e *bias* são atualizados:

$$w \leftarrow w - \alpha \, \Delta_w \, L \tag{3.8}$$

$$b \leftarrow b - \alpha \, \Delta_b \, L \tag{3.9}$$

O parâmetro α das equações representa a taxa de aprendizado (em inglês *learning rate* - LR), ou seja, a taxa de variação dos pesos e *bias* para cada iteração. Esta taxa é então multiplicada pela derivada da função perda com relação à cada peso e *bias*.

3.1.8 Sobre-ajuste e Sub-ajuste

De acordo com Goodfellow, Bengio e Courville (2016), o que separa Aprendizado de Máquina de um problema de otimização é a necessidade de técnicas capazes de treinar modelos que generalizem exemplos nunca antes processados. Ou seja, não é suficiente que o modelo desenvolvido possua boa acurácia apenas na base de dados de treinamento, ele necessita passar por uma validação do treinamento.

Para determinar o desempenho de generalização de uma rede neural é necessário ter dois conjunto de dados: dados de treinamento e dados de teste. O objetivo é diminuir tanto o erro de treinamento como de teste. Quando estes dois erros não caminham juntos,

ocorrem dois fenômenos definidos a seguir: subajuste (referenciado em muitas literaturas no inglês *underfitting*) e sobre-ajuste (em inglês *overfitting*).

O sub-ajuste ocorre quando o erro de treinamento não reduz. Ou seja o modelo não foi capaz de determinar uma relação entre os dados de treinamento. Geralmente quando os modelos e/ou bases de dados são simples a rede não consegue aprender o padrão.

O sobre-ajuste ocorre quando o erro de treinamento decresce mas o de teste continua alto. Ou seja, a rede não é capaz de generalizar para exemplos não vistos. Isto pode ocorrer devido a utilização de pequena base de treinamento ou uso excessivo de características (nós).

3.1.9 Treinamento de uma Rede Neural Artificial de Mútliplas Camadas

O treinamento de uma rede neural artificial consiste no ajuste dos pesos sinápticos e *bias* de modo que o vetor de saída se aproxime da saída esperada. O processo ocorre em duas etapas principais: a propagação (*forward-propagation*) e a retro-propagação (*back-propagation*) (RUSSELL; NORVIG, 1995).

A primeira etapa de propagação ocorre quando aplicamos a Equação 3.1 de neurônio artificial para cada um dos nós da rede. Como explicado por Hafemann (2014), para tarefas de classificação, o fluxo de propagação segue da camada de entrada, passa pelas camadas ocultas e termina na camada de saída. Nesta última camada é comum utilizar a Função 3.5 de *Softmax*, que produz uma saída adequada para compararmos com o rótulo esperado da codificação *one-hot*.

Terminada a etapa de propagação, a rede entrega como saída os valores estimados por ela. Estes valores de saída são comparados com o desejado e a função perda é definida, estabelecendo então o desempenho da rede no instante. Se este desempenho não foi o suficiente, é iniciada então a fase de retro-propagação, onde se deseja minimizar o erro da estimação. Para isto é necessário calcular a função *Cross Entropy* e função perda descritas pelas equações 3.7 e 3.6 e utilizar o método Gradiente Descendente.

A cada passo, de propagação e retro-propagação, o vetor de pesos é alterado na direção que produz a maior queda ao longo da superfície de erro. Este processo continua até atingir um erro mínimo local.

3.2 Aprendizado Profundo

O desenvolvimento de Aprendizado Profundo (do inglês *Deep Learning* - DL) foi motivado em parte pela falha de algoritmos tradicionais em generalizar tarefas de IA como reconhecimento de fala e objetos (GOODFELLOW; BENGIO; COURVILLE, 2016). Aprendizagem Profunda refere-se à modelos de MLNN com mais de duas camadas ocultas e técnicas que treinam este modelo de forma eficiente.

O aumento de camadas em uma RNA causa um crescimento considerável de parâmetros que devem ser ajustados no algoritmo de aprendizagem, então dois fatores são fundamentais para a viabilidade da ferramenta: alta capacidades de processamento e extensos banco de dados (FERREIRA, 2017). Portanto, esta ferramenta de aprendizado só se tornou viável recentemente, com o barateamento de sensores de qualidade, que geram enormes volume de dados para o treinamento e o aumento da capacidade processamento das máquinas com unidades de processamento gráfico (GPU - *Graphic Processing Unit*).

3.2.1 Método do Gradiente Descendente Estocástico

O método do Gradiente Descendente Estocástico (do inglês *Stochastic Gradient Descent - SGD*) é um dos algoritmos de otimização em aprendizado de máquina e aprendizado profunda mais utilizados (GOODFELLOW; BENGIO; COURVILLE, 2016). SGD é uma adaptação do método GD apresentado na Subseção 3.1.7, no qual procura resolver o problema do GD em conjuntos de dados muito grandes. Visto que o GD calcula a função perda com o gradiente para cada dado de treinamento, em redes neurais com muitas camadas e nós a utilização deste método pode torna-lo inviável. O método SGD tenta contornar este problema calculando uma estimativa da perda, utilizando uma pequena parte do conjunto de treinamento.

3.2.2 Dropout

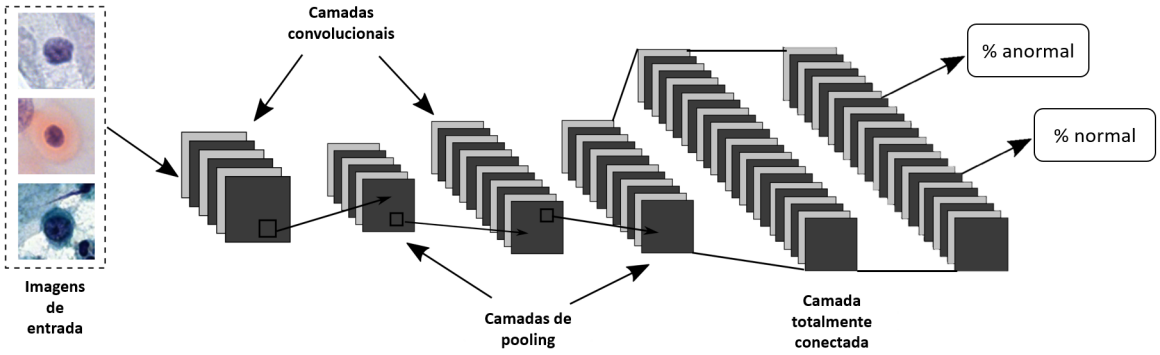
O *Dropout* é uma técnica de regularização que consiste em remover aleatoriamente a cada iteração de treinamento uma determinada porcentagem dos neurônios de uma camada, re-adicionando-os na iteração seguinte. Essa técnica também confere à rede a habilidade de aprender atributos mais robustos, uma vez que um neurônio não pode depender da presença específica de outros neurônios (ARAÚJO; CARNEIRO; SILVA, 2017).

3.3 Redes Neurais Convolucionais

As Redes Neurais Convolucionais (*Convolutional Neural Network - CNN*) são arquiteturas de aprendizado profundo que subdividem os dados para tentar extrair características de cada conjunto. Um dos objetivos da CNN é reduzir o número de parâmetros que deverão ser ajustados pela rede, e então melhorar o processo de treinamento. Uma característica importante desta arquitetura está relacionada com sua invariância a escala, a translação e outras transformações, ou seja, ela consegue reconhecer padrões de forma mais robusta e automática.

As CNNs são MLNN muito utilizadas em tarefas com estruturas em grades, como por exemplo, processamento de fala e entendimento da linguagem natural (1D, convoluções temporais), segmentação e classificação de imagens (2D, convolução espacial), e análise de

Figura 8: Arquitetura de uma Rede Neural Convolucional

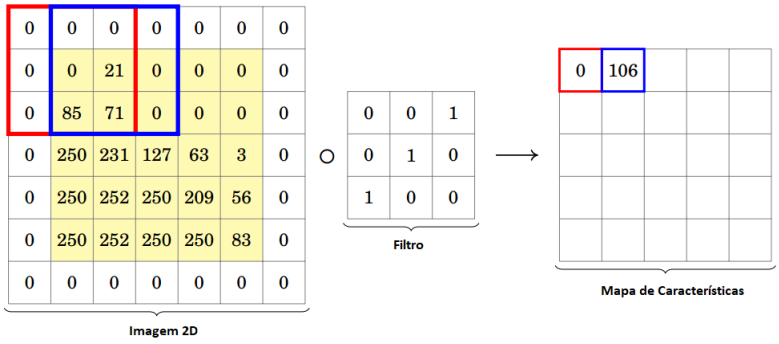


Fonte: Araújo, Carneiro e Silva (2017)

vídeos (3D, convolução volumétrica) (SIMONOVSKY; KOMODAKIS, 2017) (LECUN; BENGIO; HINTON, 2015). As principais camadas e as quais utilizamos neste trabalho são: convolucionais, de *pooling* e totalmente conectadas.

A Figura 8 ilustra um exemplo de uma CNN, a LeNet (**??**), que classifica as imagens de entrada em células anormais ou normais. Na arquitetura, as camadas convolucionais são responsáveis por extrair as características. As camadas de *pooling* reduzem a dimensionalidade da rede. As camadas totalmente conectadas estão no fim da rede, ligam todas saídas da camada anterior e determinam utilizando de funções de ativação a saída da CNN.

Figura 9: Operação de Convolução



Fonte: Imagem adaptada do site https://www.vaetas.cz/blog/intro-convolutional-neural-networks/

3.3.1 Camada Convolucional

As camadas convolucionais (*Convolutional layer* - CL) são conjuntos de filtros não lineares que percorrem sequencialmente os dados de entrada (ou camada anterior) e então produzem matrizes chamadas mapas de características (*feature maps)*. A Figura 9 exemplifica a operação de convolução, onde o filtro de tamanho 3x3 sobrepõe uma região dos dados, a multiplicação matricial entre eles é computada e então os valores somados são passados para o mapa de características.

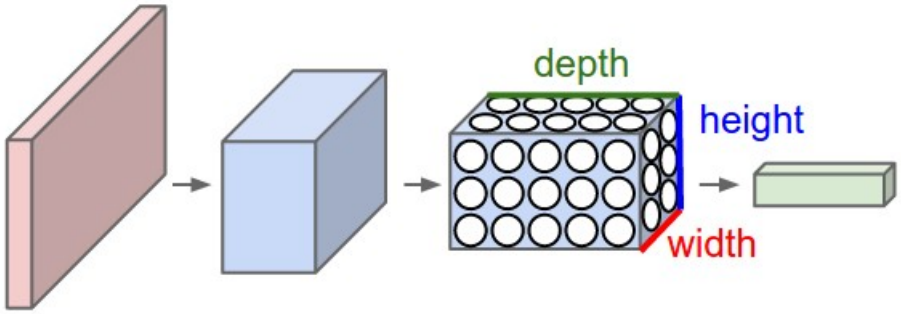
Durante o processo de treinamento, esses filtros são ajustados automaticamente para que sejam ativados na presença de características relevantes, como orientação de bordas ou manchas de cores (KARPATHY, 2017). Em cada camada convolucional diversos filtros são usados, e os mapa de características produzidos são então empilhados, formando uma matriz 3D para imagens 2D ou uma matriz 4D para imagens 3D. A Figura 10 demonstra a forma como estes mapas são empilhados e como cada camada então aumenta sua profundidade (*depth*) em uma CNN com imagem 2D de entrada e matrizes 3D nas camadas subsequentes.

Ainda observando a Figura 10, cada camada convolucional varia a largura (*width*) e a altura (*height*) conforme os dados percorrem a CNN. Esta variação ocorre com a mudança de dois parâmetros na operação de convolução: o passo dos filtros (*stride*) e o preenchimento na camada que sofrerá a convolução (*padding*).

3.3.1.1 Passo (*stride*)

Conforme explicado anteriormente os filtros da operação de convolução deslizam por toda matriz de forma sequencial. Este deslizamento ocorre em passos, passando de pixels para pixel em uma imagem ou de posição para outra em uma matriz. Quando o passo é igual a um, a altura e largura da camada de saída será igual a entrada. Quando for dois, a saída possuirá metade do tamanho da entrada.

Figura 10: Camadas Convolucionais



Fonte: Karpathy (2017)

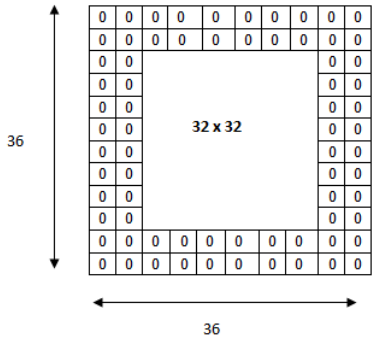
3.3.1.2 Preenchimento (*padding*)

Durante o deslizamento dos filtros, há diversas formas de lidar com as bordas, as mais utilizadas são: ***valid padding*** ou ***same padding***. No *valid padding* as bordas do filtro não ultrapassam as bordas da imagem, enquanto no *same padding* as fronteiras da imagem são preenchidas com 0 de modo a controlar a altura e largura na camada de saída. Este preenchimento é determinado com a Equação 3.10 e podemos observar na Figura 11 como é feito para uma imagem de 32x32 e filtro de 5x5.

$$P = \frac{K - 1}{2} \tag{3.10}$$

onde K é o tamanho do filtro.

Figura 11: Preenchimento para filtro 5x5



Fonte: Deshpande (2016)

Após definir os parâmetros da convolução, é possível determinar a dimensão de saída da CL utilizando a Equação 3.11.

$$O = \frac{(W - K - 2P)}{S} + 1 \tag{3.11}$$

onde O é a dimensão da saída, W é a dimensão da entrada, K é o tamanho do filtro, P é o preenchimento e S é o passo do filtro.

3.3.2 Camada de Pooling

A camada *pooling*, geralmente utilizada após uma camada convolucional, têm como objetivo reduzir a dimensão da camada de entrada, para diminuir o custo computacional e evitar *overfitting*. O método mais comum, chamado de *Max Pooling*, consiste em reduzir a dimensão das camadas pegando o valor máximo de cada região. Desta forma, ele elimina valores desprezíveis, criando uma invariância a pequenas mudanças e distorções locais (ARAÚJO; CARNEIRO; SILVA, 2017).

3.3.3 Camada Totalmente Conectada - FCL

As camadas totalmente conectadas (em inglês Fully Connected Layer - FCL) são camadas iguais às MLNNs convencionais, onde todos os neurônios da camada anterior estão conectados com cada neurônio desta camada. Nesta camada, as características extraídas nas camadas convolucionais e de *pooling* são classificadas e na última camada utiliza-se a função de ativação *softmax* para predizer a classe do objeto de entrada.

4 IMPLEMENTAÇÃO

Neste capítulo serão descritos os materiais e métodos utilizados para a solução do problema apresentado. A metodologia consiste no treinamento e validação de uma rede neural convolucional para aprendizado de classificação de objetos tridimensionais a partir de um banco de dados ModelNet10, para isto o experimento foi dividido em duas etapas principais: pré-processamento da base de dados e treinamento da CNN.

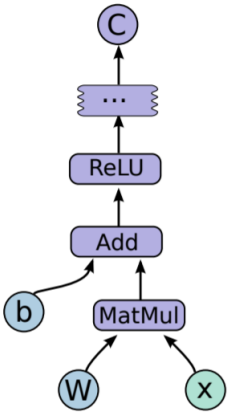
4.1 Materiais

A rede neural convolucional e pré-processamento dos dados foram implementados em Python, utilizando bibliotecas como TensorFlow, cuDNN e Numpy. Todos os códigos foram feitos e rodados em um computador com CPU Intel(R) Core(TM) i7-5500U 2,40 GHz, GPU NVIDIA GeForce 840M e 8GB de memória RAM. Vale destacar a utilização de processamento paralelo através da GPU e com auxílio as bibliotecas cuDNN e TensorFlow com o objetivo de diminuir o tempo computacional.

4.1.1 TensorFlow

A biblioteca escolhida para a implementação da rede neural foi o *TensorFlow*. Ele é um *framework* de código aberto para computação numérica e *Machine Learning* implementado em *Python* (ABADI et al., 2016). Foi disponibilizado pelo *Google Brain Team* em novembro de 2015 e está sendo amplamente utilizada em *Machine Learning* por diversas empresas como: Google, Twitter, Intel, Dropbox, Ebay, entre outras.

Figura 12: Exemplo de Fluxo de Dados Gráficos do TensorFlow



Fonte: Abadi et al. (2016)

Como explicado em Zaccone (2016), a computação do *TensorFlow* pode ser descrita como um fluxo de dados gráficos (*Data Flow Graph*), onde cada nó representa uma instância de operação (multiplicação, ReLU, convolução, e outros) e os cantos (*edges*) são vetores multidimensionais de dados chamados de **tensores**.

A Figura 12 representa um exemplo de fluxo de dados gráficos de uma rede neural *Perceptron*, com dados entradas (tensor "x") multiplicados pelos pesos sinápticos (tensor "W") e somados pelo limiar de ativação (tensor "b"). O tensor de custo de saída (C) é obtido da função de ativação *ReLU*.

4.1.2 Base de Dados

A base de dados utilizada foi retirada do projeto Princeton ModelNet10. Este projeto têm como objetivo fornecer modelos de objetos CAD 3D para pesquisadores em visão computacional, robótica, cientistas cognitivos e outros (PRINCETON, 2017). A base de dados ModelNet10 contêm 4899 modelos CAD 3D de diferentes objetos, separados em 10 categorias e em conjuntos de treinamento e teste. A Tabela 1 enumera a quantidade de objetos em cada categoria da base de dados.

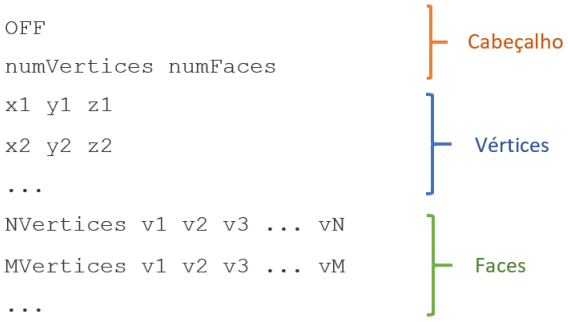
Tabela 1: Conjunto de dados de treinamento e teste

Objeto	**Dados de treinamento**	**Dados de teste**
Cama	515	100
Cadeira	889	100
Banheira	106	50
Escrivaninha	200	86
Cabeceira	200	86
Cômoda	200	86
Monitor	465	100
Sofa	680	100
Mesa	392	100
Vaso sanitário	344	100
Total	**3991**	**908**

4.1.2.1 Formato OFF

A base de dados disponibilizada pelo ModelNet10, está em formato OFF (*Object File Format*). Este formato representa geometricamente modelos 3D através de polígonos em cada superfície do objeto. Estruturalmente cada arquivo de objeto segue os padrões indicados na Figura 13. No cabeçalho são definidos o formato dos dados, o número de vértices e número de faces. Logo abaixo do cabeçalho são listados os vértices com as coordenadas x, y e z. Por fim, as faces são determinadas pelo número de vértices que serão ligados, seguido do índice da lista de vértices.

Figura 13: Estrutura de Arquivo .off



Fonte: Elaborado pelo autor

Em Python, foi produzido um programa utilizando as bibliotecas Numpy, Matplotlib e STL para produzir graficamente os objetos no formato OFF. A Figura 14 apresenta alguns exemplos de cada classe de objetos do ModelNet10.

4.2 Pré-processamento

4.2.1 Conversão da Representação 3D

Conforme discutido no capítulo 2, existem diversas formas de representação 3D. Neste projeto optou-se por trabalhar com a nuvem de pontos, dado sua facilidade e aplicabilidade em CNN. Para isto, é necessário o pré-processamento da base de dados convertendo da representação *mesh* para a nuvem de pontos. Esta conversão consiste em povoar a superfície com pontos de ocupação em uma grade.

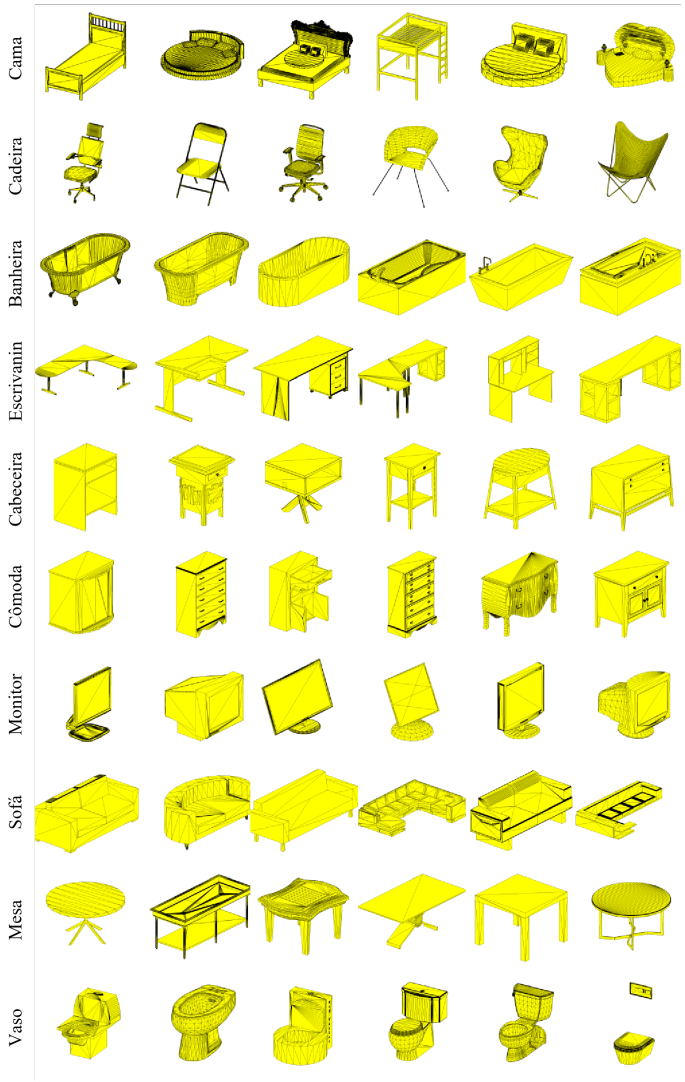
4.2.2 Grade

A nuvem de pontos é representada por um sistema de coordenadas tridimensionais definido como **grade**. Neste trabalho, a grade receberá o valor de "1" quando a coordenada está ocupada pelo objeto e o valor de "0" quando a coordenada não está ocupada. Os objetos serão então convertidos para uma matriz de três dimensões delimitada pelo tamanho e resolução desejados.

4.2.3 Normalização

A primeira etapa do pré-processamento consiste em normalizar o conjunto de dados. A normalização dos dados de entrada, é um método muito comum e essencial para evitar problemas de estabilidade numérica. Redes Neurais Artificiais com valores de dados de entrada muito altos e/ou muito baixos não conseguem obter bons resultados de generalização.

Figura 14: Exemplos dos objetos em .off



Fonte: Elaborado pelo autor

Neste trabalho foram utilizados as Equações 4.1, 4.2, 4.3 para a normalização da figura de acordo com o tamanho da grade desejado. Cada coordenada é normalizada para um intervalo de [0,1] e multiplicada pelo tamanho de grade. De modo a manter as proporções do objeto, o denominador das equações deve ser a maior distância dentre os eixos X, Y e Z (max(X,Y,Z)), definido pela Equação 4.4.

$$\hat{X} = \frac{X - min(X)}{max(X, Y, Z)} \cdot grade \tag{4.1}$$

$$\hat{Y} = \frac{Y - min(Y)}{max(X, Y, Z)} \cdot grade \tag{4.2}$$

$$\hat{Z} = \frac{Z - min(Z)}{max(X, Y, Z)} \cdot grade \tag{4.3}$$

$$max(X, Y, Z) = max(max(X) - min(X)), (max(Y) - min(Y)), (max(Z) - min(Z))) \tag{4.4}$$

4.2.4 Povoamento das Faces

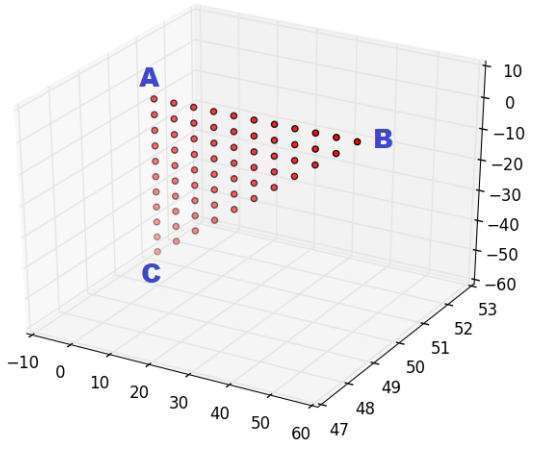
O método de conversão utilizado, consiste em preencher as faces dos objetos com pontos em uma grade. Como os polígonos das faces dos dados do ModelNet10 são triângulos, o povoamento das faces se baseou no sistema de coordenadas baricêntricas, definida pela Equação 4.5:

$$P = (1 - u - v)A + uB + vC \tag{4.5}$$

$$u + v = 1 \tag{4.6}$$

onde **P** na Equação 4.5 representa um ponto no espaço delimitado pelos vértices do triângulo **A**, **B** e **C**, e (u, v) são escalares que formam as coordenadas baricêntrica do ponto **P**. Assim computacionalmente, por interações, podemos encontrar todos os pontos **P** dentro do triângulo, variando (u, v) dentro da condição da Equação 4.6.

Figura 15: Povoamento de um triângulo



Fonte: Elaborado pelo autor

Para cada ponto contido no triângulo calculado pelo programa, a coordenada é aproximada para um ponto na grade, recebendo então o valor "1". O dado de cada objeto de resolução previamente definida da forma do exemplo da Figura 15.

4.2.5 Codificação One-hot

Conforme explicado anteriormente na Subseção 3.1.4, RNAs não podem operar com rótulos diretamente. Por causa disso foi desenvolvido a Tabela 2 com a codificação *one-hot* dos rótulos das classes de objetos. Na primeira coluna temos os rótulos de cada classe de objetos, em seguida as linhas formam os vetores que irão representá-los.

Tabela 2: Codificação One-hot das classes de objetos

Cama	**1**	0	0	0	0	0	0	0	0	0
Cadeira	0	**1**	0	0	0	0	0	0	0	0
Banheira	0	0	**1**	0	0	0	0	0	0	0
Escrivaninha	0	0	0	**1**	0	0	0	0	0	0
Cabeceira	0	0	0	0	**1**	0	0	0	0	0
Cômoda	0	0	0	0	0	**1**	0	0	0	0
Monitor	0	0	0	0	0	0	**1**	0	0	0
Sofa	0	0	0	0	0	0	0	**1**	0	0
Mesa	0	0	0	0	0	0	0	0	**1**	0
Vaso sanitário	0	0	0	0	0	0	0	0	0	**1**

4.2.6 Conjunto de Dados de Treinamento e Teste

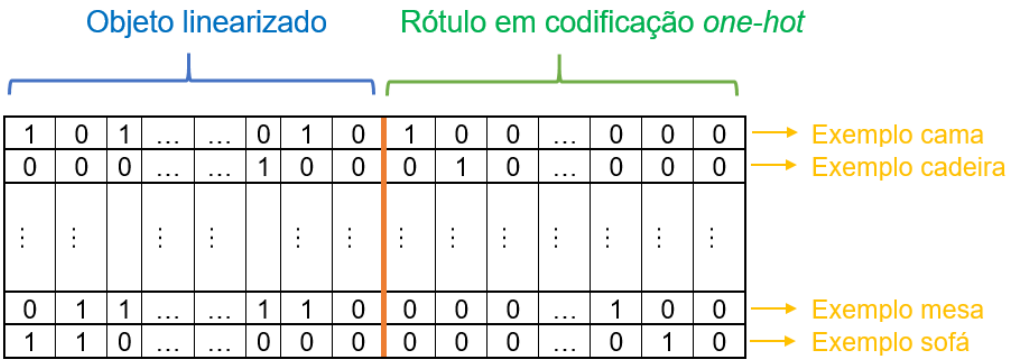
Após o preprocessamento, cada objeto representado em uma matriz 3D é linearizado em um vetor e concatenado com o devido vetor do rótulo da classe do objeto. O vetor resultante é então empilhado com os outros vetores de objetos, formando a matriz do conjunto de dados. No total 4 arquivos de dados são produzidos, 2 arquivos com a matriz dos dados de treinamento com grade de 16 e 32, e mais 2 arquivos para teste.

A Figura 16 exemplifica a estrutura de cada arquivo produzido pelo preprocessamento. Cada linha representa um objeto com seu rótulo, para o arquivo dos dados de treinamento com grade de 32, a matriz possui 3991 linhas e $32^3 + 10$ colunas.

4.3 Treinamento da Rede Neural Convolucional

Após o pré-processamento dos dados de entrada da rede neural convolucional, a rede esta apta para ser treinada. A determinação da topologia de CNN mais adequada para resolução do problema é um processo empírico, portanto diversas topologias com técnicas diferentes foram testadas, a fim de determinar a topologia que obtêm o melhor resultado. Neste trabalho foram testados topologias variando o tamanho da grade dos conjunto de dados, o número de camadas convolucionais, número de camadas totalmente conectadas, tamanho dos filtros e taxa de aprendizagem.

Figura 16: Estrutura do conjunto de dados de entrada após preprocessados



Fonte: Elaborado pelo autor

Durante a configuração das CNNs os seguintes parâmetros são definidos como fixos e não foram mudados no decorrer de todo o experimento:

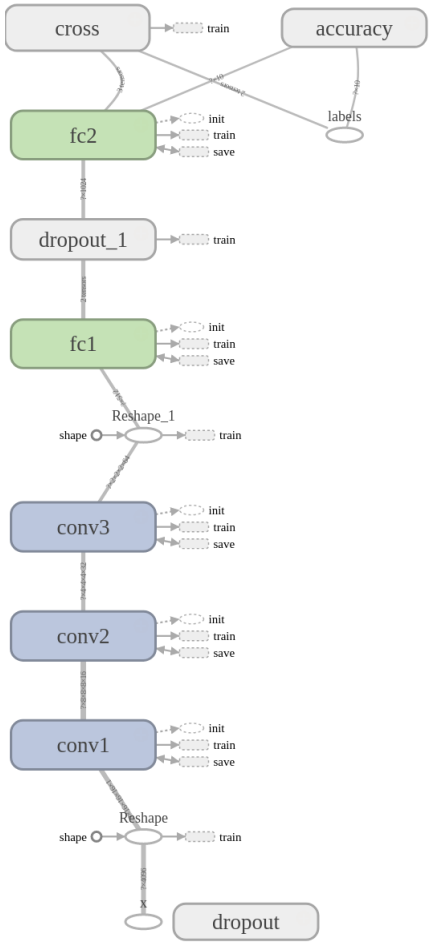
1. Passo da camada convolucional = 1;
2. Passo da camada de *Max Pooling* = 2;
3. Tamanho do filtro do *Max Pooling* = 2;
4. Número de épocas = 1950;
5. *Dropout* = 0,50;
6. Número de mapas de características:
   a) com 2 camadas convolucionais:
      i. 1° CL = 32
      ii. 2° CL = 64
   b) com 3 camadas convolucionais:
      i. 1° CL = 16
      ii. 2° CL = 32
      iii. 3° CL = 64

Fixado estes parâmetros, utilizou-se os 3991 dados de treinamento dos objetos 3D com grade de 32 para o treinamento de diversas CNNs. Neste caso 16 topologias diferentes foram testadas, compostas pela variação dos 4 parâmetros listados a seguir:

- Com taxa de aprendizado de $1 \cdot 10^{-3}$ ou $1 \cdot 10^{-4}$;

- Com 2 camadas convolucionais ou 3 camadas;
- Com 1 camada totalmente conectada ou 2 camadas;
- Com tamanho de filtro de 3x3 ou 5x5.

Figura 17: Topologia completa da CNN utilizada



Fonte: Elaborado pelo autor

A Figura 17 exemplifica a topologia mais completa com 3 camadas convolucionais e 2 FC. Outras topologias descritas anteriormente são pequenas variações desta. Após o treinamento de todas as topologias com os objetos com grade de 32, o mesmo processo foi realizado, porém para grade de 16. Totalizando 32 topologias treinadas, os desempenhos de cada uma foram gravados e serão apresentados no capítulo 5.

5 EXPERIMENTOS E RESULTADOS

Neste capítulo são apresentados de forma gráfica os resultados do pré-processamento dos objetos 3D, com a análise da conversão para representação em nuvem de pontos e as diferenças entre as imagens com diferentes tamanhos de grade. Será também apresentado o desempenho de cada uma das arquiteturas das CNNs treinadas e discutidos a influência de cada parâmetro variado.

5.1 Pré-processamento

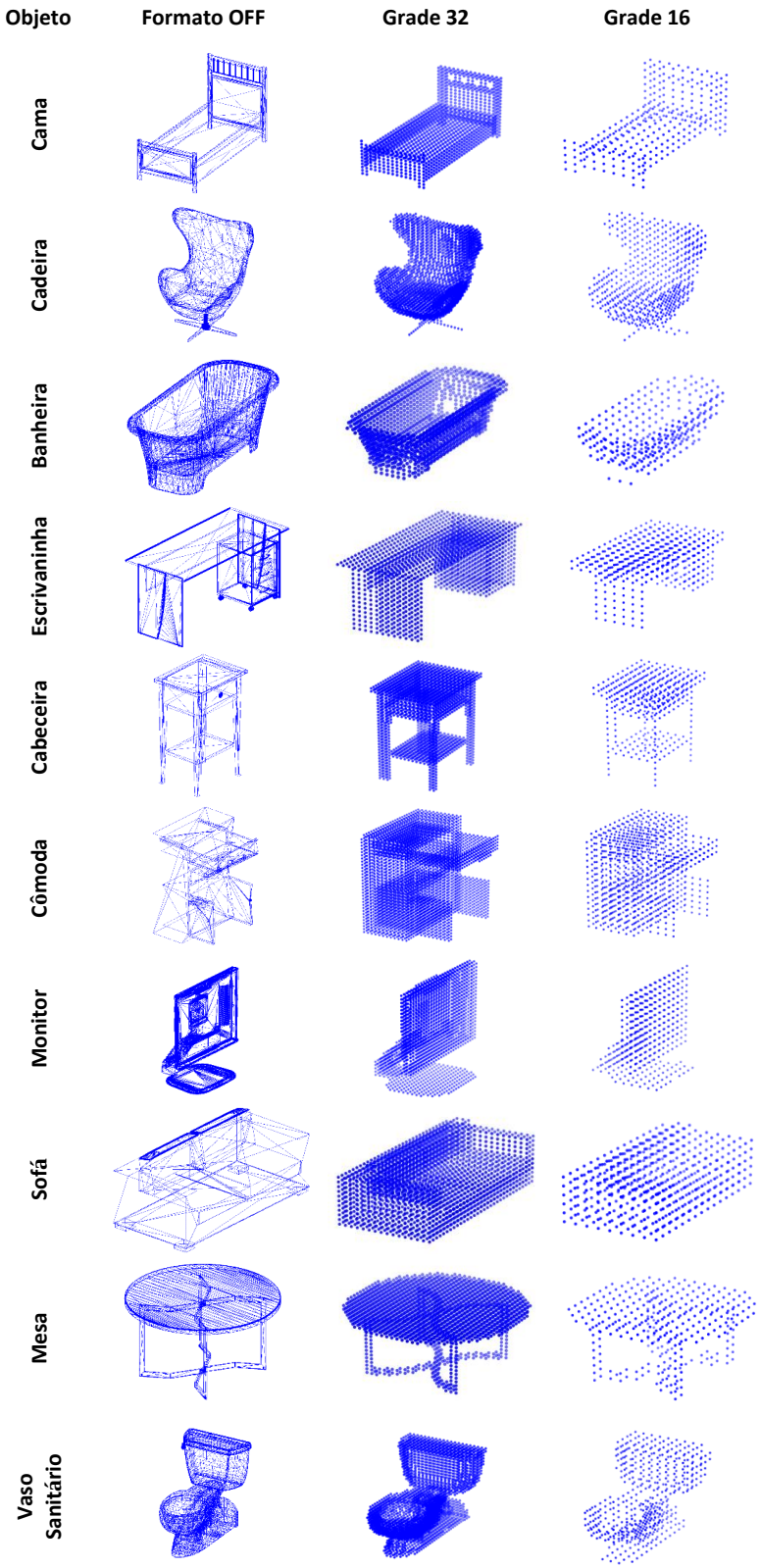
Conforme definido no Capítulo 4, 4899 imagens foram pré-processadas para o conjunto de objetos de treinamento e teste com grade 32 e mais 4899 imagens para os objetos com grade de 16, totalizando 9798 imagens 3D. A Figura 18 compõe o resultado deste processo com um exemplo de cada objeto. Na coluna do formato OFF estão presentes os objetos com a representação original retirada do banco de dados ModelNet10. Na coluna seguinte estão presentes os objetos processados em nuvem de pontos com grade de 32 e na última coluna os objetos na grade de 16.

Devido a baixa resolução da nuvem de pontos (pequeno tamanho da grade), é possível observar que as imagens preprocessadas não possuem o mesmo nível de detalhes das imagens originais.

5.2 Treinamento e Teste da Rede Neural Convolucional

Para o treinamento de todas as topologias, foi desenvolvido um *script* em *Python* para que uma sequência de treinamentos seja realizada, onde para cada treinamento os parâmetros desejados eram alterados. O número de épocas de todos os treinamentos foi determinado a partir de alguns treinamentos preliminares, onde 1950 épocas foi o valor onde a função perda e a acurácia possuem variação menor que 0,1% por época. Definido então o número de épocas e executado o treinamento, os resultados com valores de acurácia e função perda de cada topologia são salvos e plotados utilizando a ferramenta do TensorFlow chamada de Tensorboard.

Figura 18: Exemplos de conversão da representação dos objetos



Fonte: Elaborado pelo autor

Durante a gravação dos resultados adotou-se uma nomenclatura para distinguir cada topologia utilizada. Cada resultado foi salvo em um arquivo, cujo nome contêm todos os parâmetros da topologia. Os parâmetros e sua nomenclaturas são enumeradas abaixo, onde n é o valor atribuído ao parâmetro:

- Taxa de aprendizado: "lr_"+n;
- Número de camadas convolucionais: "conv="+n;
- Número de camadas totalmente conectadas: "fc="+n;
- Tamanho da grade do objeto: "img="+n;
- Tamanho do filtro: "ksize="+n.

Os treinamentos e testes da CNN foram separados em dois conjuntos: o treinamento com objetos 3D com grade de 32 e o treinamento com objetos 3D com grade de 16. Nas sub-seções a seguir serão apresentados os resultados de cada um destes conjuntos.

5.2.1 Objetos 3D com grade de 32

Para treinamento de todas as topologias com a grade de 32 foi utilizado o código fonte presente no Apêndice B. Utilizando o hardware descrito anteriormente, o treinamento levou 47 horas 36 minutos e 30 segundos para ser completado. Na Tabela 3 são apresentados os resultados de cada topologia treinada.

Tabela 3: Desempenho das topologias de CNN para objetos com grade de 32

Taxa de Aprendizagem	**Número de Camandas Convolucionais**	**Número de Camadas FC**	**Tamanho do Filtro**	**Treinamento**		**Teste**	
				Acurácia	**Custo**	**Acurácia**	**Custo**
1E-3	2	1	3x3x3	0,3800	1,7960	0,2080	2,0430
			5x5x5	0,1420	2,3030	0,1140	2,3030
		2	3x3x3	0,9980	0,0004	0,8740	0,8716
			5x5x5	0,1100	2,3030	0,0840	2,3030
	3	1	3x3x3	0,5300	1,084	0,3900	1,5570
			5x5x5	03540	1,8420	0,2080	2,1780
		2	3x3x3	**0,9940**	**0,0008**	**0,8900**	**0,3500**
			5x5x5	0,1020	2,3030	0,1200	2,3030
1E-4	2	1	3x3x3	0,7600	0,5849	0,5500	1,2370
			5x5x5	0,3300	1,8330	0,2943	1,9380
		2	3x3x3	0,9820	0,0437	0,8557	0,6206
			5x5x5	0,1320	2,3030	0,1071	2,3030
	3	1	3x3x3	0,5480	1,224	0,5157	1,4430
			5x5x5	0,6360	0,9446	0,5114	1,9160
		2	3x3x3	0,9880	0,0032	0,8871	0,6068
			5x5x5	0,2480	2,095	0,2057	2,0260

Analisando os resultados, é observado que quatro topologias obtiveram acurácia de treinamento próximo à 1, todas elas com 2 FCL e tamanho de filtros de 3x3x3. As topologias que utilizaram filtros 5x5x5 em geral foram os que obtiveram os piores resultados. O desempenho da CNN é analisado pelos resultados no conjunto de teste, sendo assim a topologia "lr_1E-3,conv=3,fc=2,img=32,ksize=3" foi a que obteve melhor resultado, com 89% de acurácia e 0,35 de perda.

São ainda produzidos com auxílio da ferramenta Tensorboard os gráficos das Figuras 19, 20, 21 e 22. Estas figuras mostram o comportamento do treinamento de acordo com o número de épocas. Em cada uma, é possível visualizar a influência da taxa de aprendizagem e o desempenho das topologias para cada época.

Figura 19: Acurácia do treinamento para Objetos - Grade 32

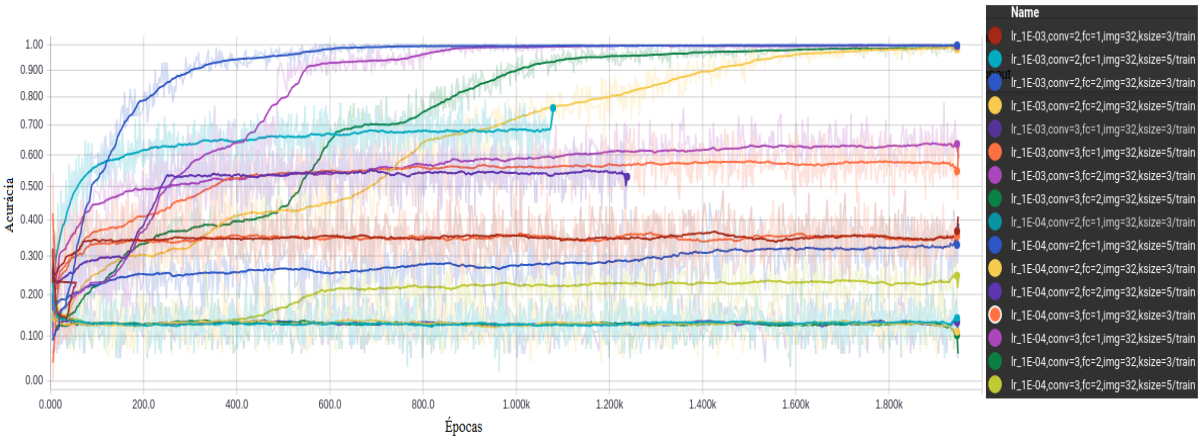


Figura 20: Acurácia do teste para Objetos - Grade 32

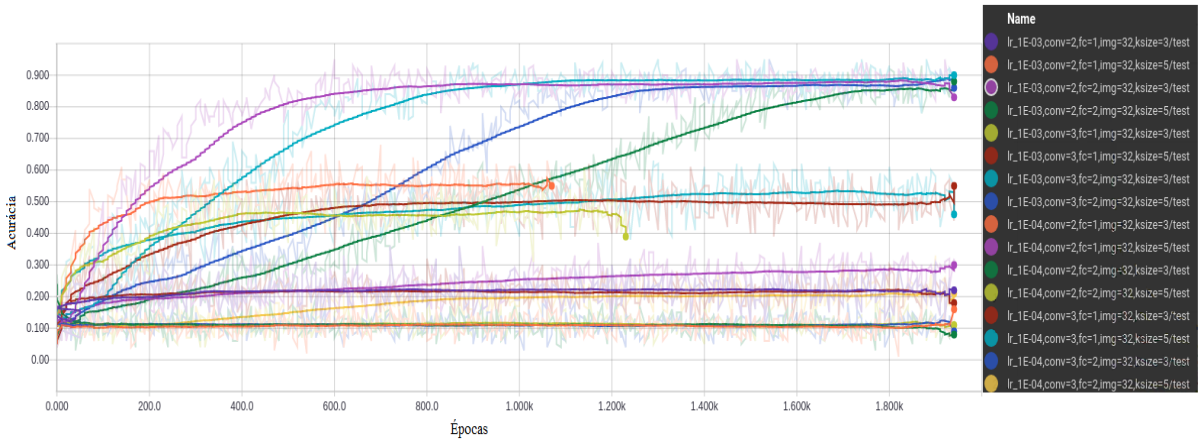


Figura 21: Perda do treinamento para Objetos - Grade 32

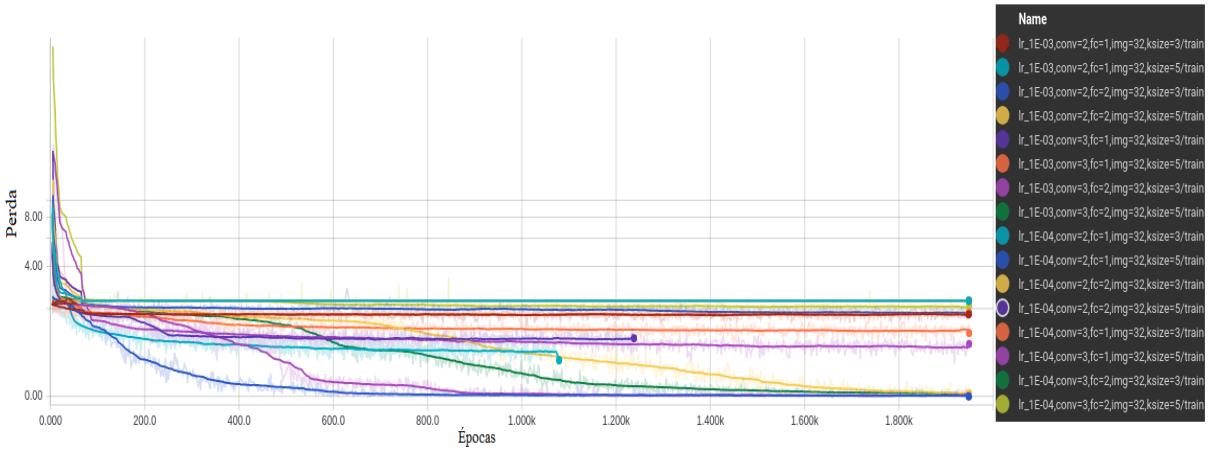
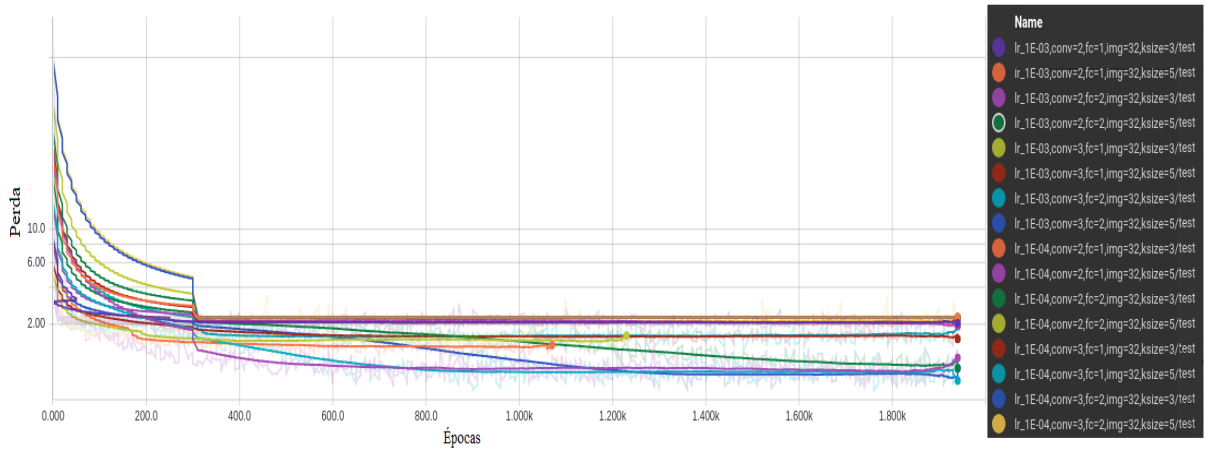


Figura 22: Perda do teste para Objetos - Grade 32



5.2.2 Objetos 3D com grade de 16

Da mesma forma que o treinamento dos objetos com grade de 32, foi utilizado o código fonte presente no Apêndice B, para o treinamento dos objetos 3D com grade de 16. O treinamento de todas as topologias com o grade de 16 levou 5 horas 10 minutos e 15 segundos para ser completada. A Tabela 4 apresenta o resultado do treinamento destas topologias.

A topologia com melhor desempenho das CNNs descritas na Tabela 4 é a "lr_1E-4,conv=2,fc=2,img=16,ksize=3". Com 88,86% de acurácia e 0,3526 de perda, obteve um resultado apenas 0,1% abaixo da melhor CNN para objetos com grade de 32, "lr_1E-3,conv=3,fc=2,img=32,ksize=3".

Tabela 4: Desempenho das topologias de CNN para objetos com grade de 16

Taxa de Aprendizagem	Número de Camadas Convolucionais	Número de Camadas FC	Tamanho do Filtro	Treinamento		Teste	
				Acurácia	Custo	Acurácia	Custo
1E-3	2	1	3x3x3	0,3044	1,6080	0,2083	1,8980
			5x5x5	0,4206	1,6470	0,3156	1,8590
		2	3x3x3	0,9333	0,1451	0,8029	0,6473
			5x5x5	0,1311	2,3030	0,1257	2,3030
	3	1	3x3x3	0,4400	1,3370	0,3000	1,8870
			5x5x5	0,1400	2,3030	0,0828	2,3030
		2	3x3x3	0,1244	2,3030	0,1086	2,3030
			5x5x5	0,1667	2,3030	0,1143	2,3030
1E-4	2	1	3x3x3	0,6556	0,9061	0,6514	0,8833
			5x5x5	0,1444	2,3030	0,1143	2,3030
		2	3x3x3	**0,9667**	**0,0840**	**0,8886**	**0,3526**
			5x5x5	0,1200	2,3030	0,1114	2,3030
	3	1	3x3x3	0,8356	0,6015	0,6486	1,1060
			5x5x5	0,8178	0,5699	0,6229	1,2340
		2	3x3x3	0,9422	0,1455	0,8143	0,4970
			5x5x5	0,2133	2,072	0,1743	2,0590

Os gráficos das Figuras 23, 24, 25 e 26, mostram os resultados para cada época da acurácia de treinamento, acurácia de teste, erro do treinamento e erro de teste, respectivamente. Nestas figuras é possível verificar de modo qualitativo a influência da taxa de aprendizagem e o desempenho das topologias para cada época de treinamento e teste.

Figura 23: Acurácia do treinamento para Objetos - Grade 16

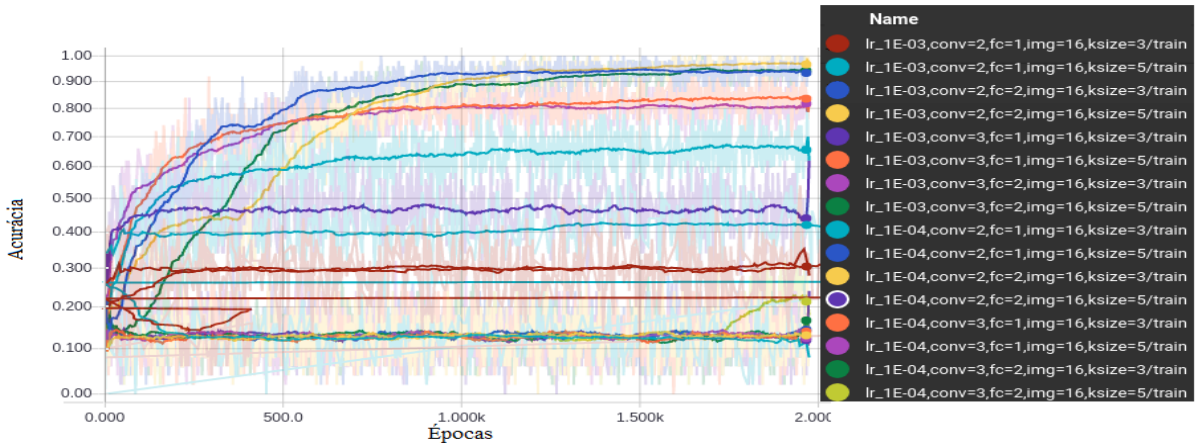


Figura 24: Acurácia do teste para Objetos - Grade 16

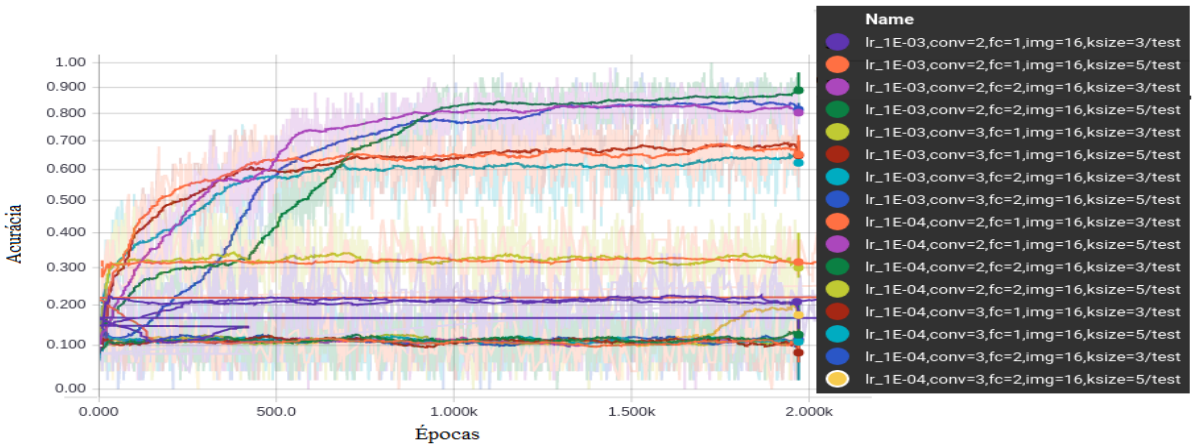


Figura 25: Perda do treinamento para Objetos - Grade 16

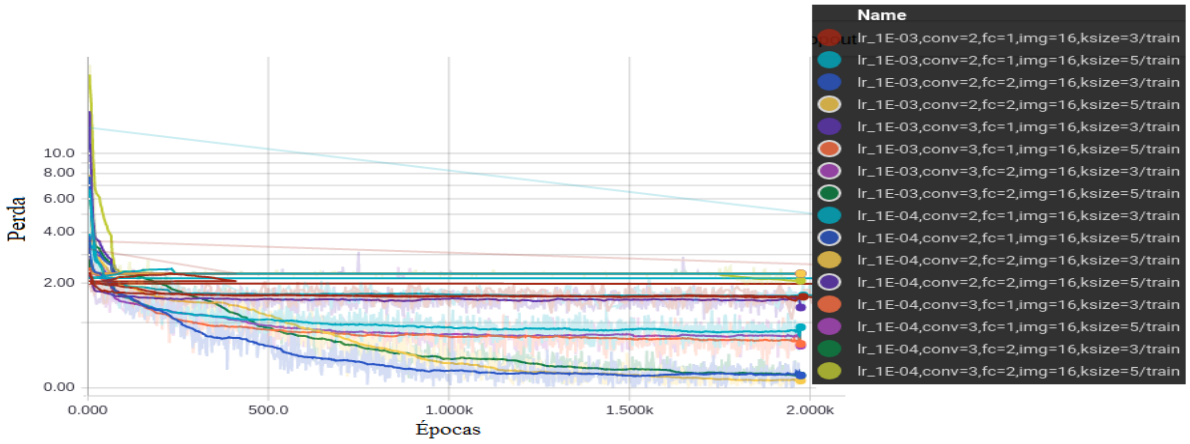
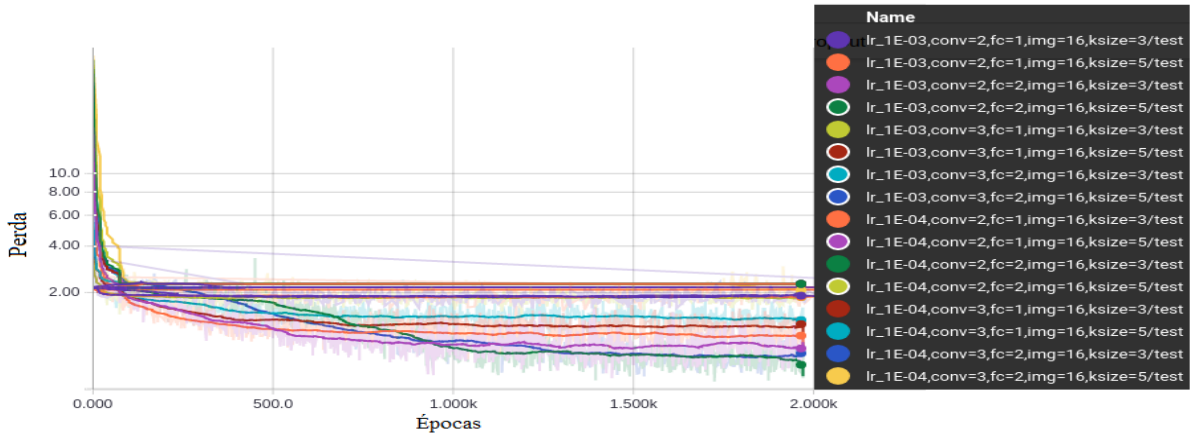


Figura 26: Perda do teste para Objetos - Grade 16



5.3 Análise da Variação dos Parâmetros

Nesta secção é analisada a influência da variação dos parâmetros no resultado final das CNNs. Pelas tabelas e gráficos mostrados na secção anterior, estes parâmetros são fundamentais para um bom desempenho no treinamento de uma CNN e não há uma fórmula para determinar a melhor topologia para cada problema.

5.3.1 Tamanho da Grade

Os tamanhos da grade testada são estabelecidos como dois problemas diferentes, com a finalidade de mostrar a robustez das redes neurais convolucionais. Apesar da origem dos objetos 3D antes do pré-processamento ser a mesma, os diferentes resultados obtidos durante o processo de treinamento, mostram que os dados de entrada influenciam na topologia à ser escolhida.

5.3.2 Número de Camadas Convolucionais

O desempenho geral com a variação do número de CLs pode ser determinado com a média dos resultados com o mesmo número de camadas. A Tabela 5 faz a síntese dos cálculos das médias, e nela é possível observar que no conjunto de dados com grade 32, o aumento do número de camadas contribui positivamente para a acurácia e diminuição da perda das CNNs. Para os objetos com grade 16, por outro lado, influencia negativamente nos resultados.

Analisando estes resultados, no conjunto de objetos com grade 32, a melhoria do desempenho das topologias com o aumento do número de CL já era esperado, pois o aumento de uma terceira camada possibilita a extração de outras características.

Tabela 5: Valores Médios da Acurácia e Perda das Topologias para mesmo Número de Camadas Convolucionais

		Grade 32		**Grade 16**	
		Conv = 2	**Conv = 3**	**Conv = 2**	**Conv = 3**
Acurácia	**Treinamento**	0,4792	0,5500	0,4595	0,4600
	Teste	0,3859	0,4659	0,4023	0,3582
Custo	**Treinamento**	1,3958	1,1871	1,4124	1,5686
	Teste	1,7024	1,5475	1,4544	1,7115

O mesmo efeito não é observado nos objetos com grade 16. Isto porque após 2 camadas convolucionais e 2 *max pooling* o tamanho dos mapas de características passam a ser 4x4x4. Com a adição de uma terceira camada convolucional e o preenchimento da imagem com zeros, utilizando *same padding*, leva a extração de características onde 33% (nos filtros de 3x3x3) dos valores não pertencem à imagem, e sim às bordas com zeros do preenchimento. Como resultado, são gerados na saída mapas de características 2x2x2 que não conseguem estabelecer um padrão para uma boa classificação.

5.3.3 Número de Camadas Totalmente Conectadas

A mesma análise foi realizada para a variação do número de camadas totalmente conectadas. A Tabela 6 demonstra que independente dos objetos de entrada os desempenhos durante a fase de teste foram melhores com duas camadas. Resultado já esperado, pois segundo (LECUN et al., 1998), apesar de teoricamente uma única FCL ser suficiente para classificar os objetos, foi observado que em situações práticas o uso de duas FCL pode produzir melhores desempenhos.

Tabela 6: Valores Médios da Acurácia e Perda das Topologias para mesmo Número de Camadas Totalmente Conectadas

		Grade 32		**Grade 16**	
		FC = 1	**FC = 2**	**FC = 1**	**FC = 2**
Acurácia	**Treinamento**	0,4600	0,5692	0,4698	0,4497
	Teste	0,3489	0,5029	0,3680	0,3925
Custo	**Treinamento**	1,4514	0,5029	1,4094	1,4573
	Teste	1,8269	1,4230	1,6842	1,5960

5.3.4 Variação do Tamanho do Filtro

A Tabela 7 produzida com a média dos resultados de cada topologia, mostra o efeito da aplicação de diferentes filtros na camada convolucional. Por causa dos tamanhos das grades utilizadas, já era esperado que o filtro 5x5x5 está sobredimensionado para os problemas. Este sobredimensionamento foi previsto da análise da segunda CL, onde as imagens de entrada são 16x16x16 (grade 32) e 8x8x8 (grade 16), e a utilização de filtros 5x5x5 pode ser considerado muito grande para a extração de alguma característica nestas imagens.

Tabela 7: Valores Médios da Acurácia e Perda das Topologias para mesmo Tamanho do Filtro

		Grade 32		**Grade 16**	
		Filtro 3x3x3	**Filtro 5x5x5**	**Filtro 3x3x3**	**Filtro 5x5x5**
Acurácia	**Treinamento**	0,7725	0,2568	0,6503	0,2692
	Teste	0,6463	0,2056	0,5528	0,2077
Custo	**Treinamento**	0,5921	1,9908	0,8913	1,9755
	Teste	1,0911	2,1588	1,1968	2,0834

5.3.5 Variação da Taxa de Aprendizagem

Na Tabela 8 é observado que a variação da taxa de aprendizagem contribui para a melhora geral dos resultados. Mas vale enfatizar que esta melhora corresponde ao desempenho médio de todas as topologias, e como foi determinado nos treinamentos há topologias com taxa de aprendizagem de 10^{-3} melhores que taxas de 10^{-4}.

Tabela 8: Valores Médios da Acurácia e Perda das Topologias para mesma Taxa de Aprendizagem

		Grade 32		**Grade 16**	
		LR = 1E-3	**LR = 1E-4**	**LR = 1E-3**	**LR = 1E-4**
Acurácia	**Treinamento**	0,4513	0,5780	0,3326	0,5870
	Teste	0,3610	0,4909	0,2573	0,5032
Custo	**Treinamento**	1,4540	1,1289	1,7436	1,1231
	Teste	1,7386	1,5113	1,9379	1,3422

5.4 Desempenho de outros trabalhos

Existem diversos trabalhos de classificação de objetos, que utilizaram o projeto Princeton ModelNet10 como base de dados. No site do projeto, estão listados alguns destes trabalhos e os respectivos resultados obtidos, a maioria deles utilizam CNNs como métodos de classificação, mas utilizam diferentes modos de representação tridimensional. Na Tabela 9 são mostrados os resultados obtidos por estes trabalhos em comparação com o resultado obtido neste experimento.

Tabela 9: Resultados obtidos por outros trabalhos

Algoritmo	**Abordagem**	**Representação 3D**	**Ano**	**Acurácia**
VRN Ensemble	CNN	Voxel	2016	97,14 %
Klokov and Lempitsky	Kd network	Nuvem de pontos	2017	94,00 %
ORION	CNN	Voxel	2017	93,80 %
LightNet	CNN	Voxel	2016	93,39 %
FusionNet	CNN	Voxel/Pixel	2016	93,11 %
Pairwise	CNN	Image Pair	2016	92,80 %
VoxNet	CNN	Nuvem de pontos	2015	92,00 %
Zanuttigh and Minto	CNN	Depth Maps	2017	91,50 %
PANORAMA-NN	CNN	Panorama	2017	91,10 %
3D-GAN	CNN e Redes Adversariais	Voxel	2017	91,00 %
ECC	CNN	Nuvem de pontos	2017	90,00 %
Presente Trabalho	CNN	Voxel	2017	**89,00 %**
Geometry Image	CNN	Mesh	2016	88,40 %
Xu and Todorovic	CNN	Voxel	2016	88,00 %
DeepPano	CNN	Panoramic view	2015	85,45 %
3DShapeNets	CNN	Voxel	2015	83,50 %
PointNet	CNN	Nuvem de pontos	2017	77,60 %

6 CONCLUSÃO

Reconhecimento de objetos tridimensionais é um problema emergente em visão computacional, que só está se tornando possível com a facilidade na geração de grandes bancos de dados e capacidade de processamento computacional cada vez maiores. Para um bom reconhecimento de objetos são necessários dois processos principais: geração de uma boa representação digital do objeto e métodos com capacidade de generalização para a classificação deste objeto. Este trabalho demonstrou a importância destes dois processos com a representação em nuvem de pontos e o método de aprendizagem de máquina, Rede Neural Convolucional.

De forma geral a utilização de CNN para classificação de objetos 3D obteve resultados muito acima do esperado. Apesar do baixo volume de exemplos do banco de dados, e a escolha de grades de 32 e 16, devido a limitação de hardware disponível pelo autor. O método se mostrou bastante robusto, com grande capacidade de generalização e conseguiu classificar os objetos com bom nível de acurácia.

Um resultado interessante neste trabalho, está no fato das CNNs conseguirem uma acurácia de 88% com o conjunto de objetos na grade de 16. Durante a fase de conversão para a representação em nuvem de pontos, foi observado que muito objetos ficam visualmente indistinguíveis. Demonstrando desta forma que CNNs são ferramentas eficientes de extração de características.

6.1 Trabalhos Futuros

Este projeto tinha como objetivo a validação da ferramenta de aprendizagem de máquina e abordar as técnica mais utilizadas para o treinamento da CNN. Além de muitas técnicas não abordadas, o treinamento com objetos 3D com tamanho de grade maior, pode proporcionar maior desempenho na classificação dos objetos. Para possíveis trabalhos futuros são sugeridos os seguintes tópicos:

- Treinamento com grade da nuvem de pontos maior;
- Aumento artificial da base de dados, utilizando técnicas como: oclusão, cortes parciais da imagem, rotação e translação;
- Utilização de outras técnicas para redução do sobre-ajuste: declínio da taxa de aprendizagem, normalização L2;
- Validação das topologias de CNN no banco de dados ModelNet40, no qual possui 40 classes de objetos 3D.

REFERÊNCIAS

ABADI, M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. **arXiv preprint arXiv:1603.04467**, 2016.

ALMEIDA, T. Uma metodologia de reconhecimento de caracteres manuscritos utilizando redes neurais embarcadas. Monografia - Faculdade 7 de Setembro, Fortaleza, Brazil. 2014.

ARAÚJO, F. H.; CARNEIRO, A. C.; SILVA, R. R. Redes neurais convolucionais com tensorflow: Teoria e prática. 2017. III Escola Regional de Informática do Piauí. Livro Anais - Artigos e Minicursos, v. 1, n. 1, p. 382-406.

BATISTA, J. N. F. Sistema de reconhecimento de objetos para demonstrador de condução robótica autónoma. 2012. Tese de mestrado integrado. Engenharia Informática e Computação. Faculdade de Engenharia. Universidade do Porto.

BENVEGNÙ, L. 3d object recognition without cad models for industrial robot manipulation. 2017. Tese de mestrado. Corso Di Laurea Magistrale In Ingegneria Informatica. Univerisita Degli Studi Di Padova.

DESHPANDE, A. **A Beginner's Guide To Understanding Convolutional Neural Networks**. 2016. Disponível em: <https://adeshpande3.github.io/A-Beginner\%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/>.

DUNDAS, J.; CHIK, D. Implementing human-like intuition mechanism in artificial intelligence. **arXiv preprint arXiv:1106.5917**, 2011.

FERREIRA, A. E. T. Estimação do ângulo de direção por vídeo para veículos autônomos utilizando redes neurais convolucionais multicanais. 2017. Trabalho de Conclusão de Curso (Graduação) — Universidade de Brasília, Instituto de Ciências Exatas, Departamento de Ciência da Computação.

GARCIA-GARCIA, A. et al. Pointnet: A 3d convolutional neural network for real-time object class recognition. In: IEEE. **Neural Networks (IJCNN), 2016 International Joint Conference on**. [S.l.], 2016. p. 1578–1584.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

HAFEMANN, L. G. An analysis of deep neural networks for texture classification. 2014. Master's degree Dissertation. M.Sc. Program in Informatics, Universidade Federal do Paraná.

KARPATHY, A. **Convolutional neural networks for visual recognition**. 2017. Disponível em: <http://cs231n.github.io/convolutional-networks/>.

KRAUSE, J. et al. 3d object representations for fine-grained categorization. In: **Proceedings of the IEEE International Conference on Computer Vision Workshops**. [S.l.: s.n.], 2013. p. 554–561.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, Nature Research, v. 521, n. 7553, p. 436–444, 2015.

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

LI, F.-F. **CS231n Lecture 1 - Introduction and Historical Context**. 2016. Disponível em: <https://youtu.be/yp9rwI_LZX8>.

LORENA, A. C.; CARVALHO, A. C. de. Uma introdução às support vector machines. **Revista de Informática Teórica e Aplicada**, v. 14, n. 2, p. 43–67, 2007.

LOWE, D. G. Object recognition from local scale-invariant features. In: IEEE. **Computer vision, 1999. The proceedings of the seventh IEEE international conference on**. [S.l.], 1999. v. 2, p. 1150–1157.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, v. 5, Dec 1943. ISSN 1522-9602. Disponível em: <https://doi.org/10.1007/BF02478259>.

OSADA, R. et al. Shape distributions. **ACM Transactions on Graphics (TOG)**, ACM, v. 21, n. 4, p. 807–832, 2002.

PRINCETON, V. Princeton modelnet project. In: . [s.n.], 2017. Disponível em: <http://modelnet.cs.princeton.edu/>. Acesso em: 04 nov. 2017.

RAVANBAKHSH, S.; SCHNEIDER, J.; POCZOS, B. Deep learning with sets and point clouds. **arXiv preprint arXiv:1611.04500**, 2016.

RUSSELL, S.; NORVIG, P. Artificial intelligence: A modern approach. **Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs**, 1995.

SILVA, S. R. e.; SCHIMIDT, F. Redução de Variáveis de Entrada de Redes Neurais Artificiais a partir de Dados de Análise de Componentes Principais na Modelagem de Oxigênio Dissolvido. **Química Nova**, v. 39, 04 2016. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0100-40422016000300273&nrm=iso>.

SIMONOVSKY, M.; KOMODAKIS, N. Dynamic edge-conditioned filters in convolutional neural networks on graphs. **arXiv preprint arXiv:1704.02901**, 2017.

WU, Z. et al. 3d shapenets: A deep representation for volumetric shapes. In: **Proceedings of 28th IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2015.

ZACCONE, G. **Getting Started with TensorFlow**. [S.l.]: Packt Publishing Ltd, 2016.

Apêndices

APÊNDICE A – CÓDIGO FONTE PARA O PRÉ-PROCESSAMENTO DOS DADOS

```
'''
Preprocessamento Mesh to Voxel v9
Autor : Caio Kioshi Miyazaki
'''

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from stl import mesh
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
import time
import os

# Convert formato off para arquivo de nuvem de pontos de
    treinamento e teste
def off2pc(grid_size,n_classes,step):

 obj = ['bed', 'chair', 'bathtub', 'night_stand', 'dresser', '
    desk', 'monitor', 'sofa', 'table', 'toilet']
 ext = ['train','test']

 for obj_c in obj:
  for ext_c in ext:
   path = obj_c+'/'+ext_c+'/'
   filenames = next(os.walk(path))[2]
   train_data = np.zeros(shape=(len(filenames),grid_size**3),
       dtype=bool)
   example_n=0

   for obj_n in filenames:
    if not obj_n.startswith("."):
     # Import object points
     print("Lendo... "+ext_c+"/"+obj_n)

```

```
33      dirFile = path+obj_n
34      file = open(dirFile)
35      file.readline()
36
37      n_vertices, n_faces, n_edges, = [int(s) for s in file.
            readline().strip().split(' ')]
38
39      verts = []
40      for x_vertices in range(n_vertices):
41          verts.append([float(s) for s in file.readline().strip()
                .split(' ')])
42
43      faces = []
44      for i_face in range(n_faces):
45          faces.append([int(s) for s in file.readline().strip().
                split(' ')][1:])
46
47      file.close()
48
49      # Controi pontos
50      v = np.array(verts)
51      f = np.array(faces)
52
53      x = v[:, 0]
54      y = v[:, 1]
55      z = v[:, 2]
56
57      minX = min(x)
58      maxX = max(x)
59      minY = min(y)
60      maxY = max(y)
61      minZ = min(z)
62      maxZ = max(z)
63
64      maxCoord = max((maxX-minX),(maxY-minY),(maxZ-minZ))
65      # Desloca para zero, centraliza e normaliza objeto pelo
            tamanho do grid e centraliza
66
67      v[:,0] = (v[:,0]*(grid_size-1)/(maxCoord)+(grid_size-1-maxX
```

```
        -minX)/2)
68      v[:,1] = (v[:,1]*(grid_size-1)/(maxCoord)+(grid_size-1-maxY
        -minY)/2)
69      v[:,2] = (v[:,2]*(grid_size-1)/(maxCoord)+(grid_size-1-maxZ
        -minZ)/2)
70
71      obj3D = mesh.Mesh(np.zeros(f.shape[0], dtype=mesh.Mesh.
        dtype))
72      for i, fc in enumerate(f):
73          for j in range(3):
74              obj3D.vectors[i][j] = v[fc[j], :]
75
76      # Inicializa 3D grid
77      grid = np.zeros(shape=(grid_size, grid_size, grid_size),
        dtype=bool)
78
79      # Popula os triangulos - Superficie
80      for trian in obj3D:
81          A = trian[0:3]
82          B = trian[3:6]
83          C = trian[6:9]
84          xA, yA, zA = A
85          xB, yB, zB = B
86          xC, yC, zC = C
87
88          distAC = int(np.linalg.norm(C - A))
89          distAB = int(np.linalg.norm(B - A))
90
91          for u in np.arange(0, distAC, step):
92              for v in np.arange(0, distAB, step):
93                  if (u / distAC) + (v / distAB) < 1:
94                      # P = A + u*(C-A)+v*(B-A)
95                      x = int(xA + u / distAC * (xC - xA) + v /
                            distAB * (xB - xA))
96                      y = int(yA + u / distAC * (yC - yA) + v /
                            distAB * (yB - yA))
97                      z = int(zA + u / distAC * (zC - zA) + v /
                            distAB * (zB - zA))
98                      grid[x][y][z] = 1
```

```
99                      else:
100                         break
101
102         train_data[example_n] = grid.flatten()
103         example_n+=1
104         # train_data.append = grid.flatten()
105
106     # Salva dados
107     file = 'data'+str(grid_size)+'/'+obj_c+'_'+ext_c+'_data'
108     print("Saving image..."+file)
109     print(train_data.shape)
110     np.save(file,train_data)
111
112 def img_gen(img_size,n_classes):
113  marker_size = 30
114  # marker = '.'
115  marker_color = 'white'
116
117  obj = ['bed','chair','night_stand','dresser','monitor','sofa','
       table','toilet']
118  ext = ['train','test']
119
120  # Import object points
121  dir = '/data32/'
122
123  for obj_i in obj:
124   for ext_i in ext:
125    file = dir+obj_i+'_'+ext_i+'_data.npy'
126    data = np.load(file)
127
128    for i in range(data.shape[0]):
129     print('Reading %s %s %d' %(obj_i,ext_i,i))
130     print('Data shape: ',data.shape)
131     print('Data size', data.shape[0])
132     data_i = np.reshape(data[i],(img_size,img_size,img_size))
133     x,y,z = data_i.nonzero()
134
135     fig = plt.figure()
136     ax = fig.add_subplot(111, projection='3d')
```

```
137       ax.scatter(x, y, z, zdir='z', marker='.',c= marker_color,s=
              marker_size)
138       ax.set_xlim([-1,img_size+1])
139       ax.set_ylim([-1,img_size+1])
140       ax.set_zlim([-1,img_size+1])
141
142       save_img_dir = save_dir+obj_i+'/'+ext_i+'/'
143       if not os.path.exists(save_img_dir):
144        os.makedirs(save_img_dir)
145
146       img_file = save_img_dir+obj_i+str(i)+'.png'
147       plt.savefig(img_file)
148       plt.close()
149
150  def mergeData(img_size,num_classes)
151   data_size = {'train':3991,'test':908} # [train,test]
152
153   obj = ['bed','chair','bathtub','night_stand','dresser','desk','
         monitor','sofa','table','toilet']
154   ext = ['train','test']
155
156   for ext_c in ext:
157    label_pos = 0
158    all_data=np.zeros((data_size[ext_c],img_size**3+num_classes))
159    tmp=0
160    for obj_c in obj:
161     # Load files
162     file = 'data'+str(img_size)+'/'+obj_c+'_'+ext_c+'_data.npy'
163     data = np.load(file)
164     print("Lendo... "+file)
165     num_examples = data.shape[0]
166     print('numero de exemplos: ',num_examples)
167     # Label
168     label = np.zeros((num_examples,num_classes))
169     label[:,label_pos] = 1
170     # Adiciona label
171     print('data shape ', data.shape)
172     print('label shape ', label.shape)
173     data_wLabel = np.concatenate((data,label),axis=1)
```

```
174     print('data com label: ', data_wLabel.shape)
175     print(tmp)
176     all_data[tmp:tmp+num_examples,:] = data_wLabel
177     print('label: ',label[0])
178     print('label_pos',label_pos)
179     label_pos+=1
180     tmp=tmp+num_examples
181     print('tmp',tmp)
182     print('
            ------------------------------------------------------------------------
            ')
183
184   print(type(all_data))
185   # Shuffle linhas
186   np.random.shuffle(all_data)
187
188   # all_data=np.array(all_data)
189   print(all_data.shape)
190   # Salva valores
191   file = 'data'+str(img_size)+'/'+ext_c+'_data'
192   print("Saving image..."+file)
193   np.save(file,all_data)
194
195
196 def main():
197   off2pc(32,010,.5)
198   img_gen(32,10,save_dir)
199   mergeData(32,save_dir)
200
201 if __name__ == '__main__':
202   main()
```

APÊNDICE B – CÓDIGO FONTE PARA O TREINAMENTO DAS CNNS

```
'''
Treinamento CNN v4
Autor: Caio Kioshi Miyazaki
'''
import tensorflow as tf
import numpy as np
import os, time

os.environ['CUDA_VISIBLE_DEVICES']=str(1)

# Parametros do modelo
n_classes = 10  # Numero de classes de objetos
batch_size = 100
n_conv_strides = 1
n_pool_strides = 2
n_pool_ksize = 2
n_epochs = 50  # Numero de epocas
drop=0.5

# Max pooling
def max_pool3d(channels_in):
    return tf.nn.max_pool3d(channels_in, ksize=[1, n_pool_ksize,
        n_pool_ksize, n_pool_ksize, 1],
                            strides=[1, n_pool_strides,
                                n_pool_strides, n_pool_strides, 1],
                                padding='SAME')

# Camada Convolucional
def conv3d_layer(input, n_conv_ksize, channels_in, channels_out,
    name="conv"):
  with tf.name_scope(name):
   w = tf.Variable(tf.truncated_normal([n_conv_ksize,n_conv_ksize
       ,n_conv_ksize,channels_in,channels_out],
     stddev=0.1),tf.float32,name="W")
   b = tf.Variable(tf.constant(0.1,shape=[channels_out]),tf.
       float32,name="B")
```

```
31    conv = tf.nn.conv3d(input,w, strides=[1,n_conv_strides,
          n_conv_strides,n_conv_strides,1],
32     padding='SAME')
33    act = tf.nn.relu(conv+tf.cast(b,tf.float32))
34    tf.summary.histogram("weights",w)
35    tf.summary.histogram("biases",b)
36    tf.summary.histogram("activations",act)
37    return max_pool3d(act)
38
39 # Camada Totalmente Conectada
40 def fc_layer(input, channels_in, channels_out, name="fc"):
41  with tf.name_scope(name):
42   w = tf.Variable(tf.truncated_normal([channels_in, channels_out
          ],stddev=0.1),tf.float32,name="W")
43    b = tf.Variable(tf.constant(0.1,shape=[channels_out]),tf.
          float32,name="B")
44    act = tf.nn.relu(tf.matmul(input,w)+tf.cast(b,tf.float32))
45    tf.summary.histogram("weights",w)
46    tf.summary.histogram("biases",b)
47    tf.summary.histogram("activations",act)
48    return act
49
50
51 def CNN3D_model(learning_rate,n_conv,n_fc,img_size,n_conv_ksize,
       hparam):
52
53  # Diretorios dos arquivos
54  LOGDIR = 'log/' #dir para logs
55  dir = 'data'+str(img_size)+'/' #dir dos dados
56  train_file = 'train_data.npy'
57  test_file = 'test_data.npy'
58  train_data = np.load(train_file)
59  test_data = np.load(test_file)
60  num_of_examples = train_data.shape[0]
61  n_batches = int(num_of_examples/batch_size)
62  n_test = test_data.shape[0]
63
64  tf.reset_default_graph()
65
```

```
66  # Placeholders and reshape data
67  x = tf.placeholder(tf.float32, [None, img_size**3],name="x")
68  y = tf.placeholder(tf.float32, [None, n_classes],name="labels")
69  x_image = tf.reshape(x, [-1,img_size,img_size,img_size,1])
70
71  with tf.name_scope('dropout'):
72   keep_prob = tf.placeholder(tf.float32)
73   tf.summary.scalar('dropout_keep_probability',keep_prob)
74  # Numero de camadas convolucionais
75  if n_conv == 1:
76   conv1 = conv3d_layer(x_image,n_conv_ksize,1,64,"conv1")
77   conv_out = max_pool3d(conv1)
78  elif n_conv == 2:
79   conv1 = conv3d_layer(x_image,n_conv_ksize,1,32,"conv1")
80   conv_out = conv3d_layer(conv1,n_conv_ksize,32,64,"conv2")
81  elif n_conv == 3:
82   conv1 = conv3d_layer(x_image,n_conv_ksize,1,16,"conv1")
83   conv2 = conv3d_layer(conv1,n_conv_ksize,16,32,"conv2")
84   conv_out = conv3d_layer(conv2,n_conv_ksize,32,64,"conv3")
85
86  size_out_layer = int((np.ceil(img_size/n_pool_strides**n_conv))
       **3*64)
87  flattened = tf.reshape(conv_out, [-1,size_out_layer])
88
89  # Numero de camadas totalmente conectadas
90  if n_fc == 1:
91   logits = fc_layer(flattened,size_out_layer,n_classes,"fc")
92  elif n_fc == 2:
93   fc1 = fc_layer(flattened,size_out_layer,1024,"fc1")
94   tf.summary.histogram("fc1/relu",fc1)
95   fc1_drop = tf.nn.dropout(fc1,drop)
96   logits = fc_layer(fc1_drop,1024,n_classes,"fc2")
97
98  with tf.name_scope("cross"):
99   cross_entropy = tf.reduce_mean(
100      tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=
           logits), name="cross")
101  tf.summary.scalar("cross",cross_entropy)
102
```

```
103  with tf.name_scope("train"):
104   train_step = tf.train.GradientDescentOptimizer(learning_rate).
        minimize(cross_entropy)
105
106  with tf.name_scope("accuracy"):
107   correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(
        y, 1))
108   accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.
        float32))
109   tf.summary.scalar("accuracy",accuracy)
110
111 summ = tf.summary.merge_all()
112  train_writer = tf.summary.FileWriter(LOGDIR + hparam+'/train')
113  test_writer = tf.summary.FileWriter(LOGDIR + hparam+'/test')
114
115  saver = tf.train.Saver()
116  with tf.Session() as sess:
117   sess.run(tf.global_variables_initializer())
118
119   for epoch in range(n_epochs):
120    np.random.shuffle(train_data)
121    for j in range(n_batches):
122     batch_xs = train_data[j*batch_size:(j+1)*batch_size,:-
          n_classes] # data
123     batch_ys = train_data[j*batch_size:(j+1)*batch_size,-
          n_classes:] # label
124     [summary, acc] =sess.run([summ, train_step],feed_dict={x:
          batch_xs, y: batch_ys, keep_prob: drop})
125     e = epoch*n_batches+j
126     train_writer.add_summary(summary,e)
127
128     if e % 5 == 0: # print accuracy
129     # train_accuracy = accuracy.eval(feed_dict={x: batch_xs, y_:
          batch_ys, keep_prob: 1.0})
130      [train_accuracy, acc] = sess.run([summ,accuracy],feed_dict
          ={x: batch_xs, y: batch_ys, keep_prob: 1.0})
131      print('Epoch %s, training accuracy %s' % (e, acc))
132
133     if e % 10 == 0: # test model
```

```
134         np.random.shuffle(test_data)
135         for test_slice in range(int(n_test/batch_size/4)):
136          batch_test_xs = test_data[test_slice*batch_size:(
                test_slice+1)*batch_size,:-n_classes] # data
137          batch_test_ys = test_data[test_slice*batch_size:(
                test_slice+1)*batch_size,-n_classes:] # label
138          [summary, acc] = sess.run([summ,accuracy],feed_dict={x:
                batch_test_xs, y: batch_test_ys, keep_prob: 1.0})
139
140          test_writer.add_summary(summary,e)
141          print('Epoch %s, testing accuracy %s' % (e, acc))
142
143     model_name = LOGDIR + hparam+'/'
144     saver.save(sess,os.path.join(model_name,"model.ckpt"),epoch*
            int(n_batches))
145
146 def make_hparam_string(learning_rate, n_conv, n_fc,img_size,
        n_conv_ksize):
147   conv_param = "conv="+str(n_conv)
148   fc_param = "fc="+str(n_fc)
149   size_param = "img="+str(img_size)
150   ksize_param = "ksize="+str(n_conv_ksize)
151   return "lr_%.0E,%s,%s,%s,%s" % (learning_rate, conv_param,
          fc_param, size_param, ksize_param)
152
153 def main():
154
155   for img_size in [16,32]:
156     for learning_rate in [1E-3, 1E-4]:
157       for n_conv in [2,3]:
158         for n_fc in [1,2]:
159           for n_conv_ksize in [3,5]:
160             start_time = time.time()
161             print('Start Time: %s' % start_time)
162             # Nome do arquivo baseando na nomenclatura (exemplo: "
                    lr_1E-3,fc=2,conv=2")
163             hparam = make_hparam_string(learning_rate, n_conv, n_fc,
                    img_size,n_conv_ksize)
164             print('Starting run for %s' % hparam)
```

```
165
166         # Roda a topologia de CNN
167         CNN3D_model(learning_rate,n_conv,n_fc,img_size,
                n_conv_ksize,hparam)
168         elapsed_time = time.time() - start_time
169         print('Elapsed Time: %s' % elapsed_time)
170
171   print('Done training!')
172   print('Run `tensorboard --logdir=%s` to see the results.' %
        LOGDIR)
173
174 if __name__ == '__main__':
175   main()
```