Pedro Henrique de Oliveira Moreira
Victor Hugo Nascimento Rocha

Rede neural convolucional e recursiva para a estimação de profundidade de objetos em vídeos

São Paulo

2018

Pedro Henrique de Oliveira Moreira
Victor Hugo Nascimento Rocha

Rede neural convolucional e recursiva para a estimação de profundidade de objetos em vídeos

Trabalho apresentado como requisito para a conclusão do curso de graduação de Engenharia Mecatrônica na Escola Politécnica da Universidade de São Paulo

Orientador: Prof. Dr. Eduardo Lobo Lustosa Cabral

São Paulo

2018

Catalogação-na-publicação

Moreira, Pedro
Rede neural convolucional e recursiva para a estimaçãode profundidade de objetos em vídeos / P. Moreira, V. Rocha, E. L. Cabral -- São Paulo, 2018.
71 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Visão computacional 2.Redes neurais 3.Processamento de imagens 4.Visão estéreo 5.Estimação da profundidade de imagens I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t. III.Rocha, Victor IV.Cabral, Eduardo, Orient.

Esta monografia é apresentada como requisito para a conclusão do curso de graduação de Engenharia Mecatrônica na Escola Politécnica da Universidade de São Paulo. Ela é o resultado do trabalho dos autores, com exceção de onde é explicitamente indicado. Este trabalho pode ser copiado e distribuído livremente, desde que seja citado apropriadamente

Dedicado a ambas as nossas famílias.

AGRADECIMENTOS

Gostaríamos de agradecer primeiramente ao orientador deste trabalho, o Prof. Dr. Eduardo Lobo Lustosa Cabral. Ele esteve sempre disposto a nos ajudar, esclarecer dúvidas e, quando necessário, criticar de maneira construtiva. Sem ele este projeto nunca teria tido os bons resultados que obteve.

É importante mencionar a imensa contribuição obtida através do uso do supercomputador HPC da USP. Nós fomos orientados acerca de como utilizá-lo pela própria equipe do HPC, que solucionou rapidamente todas as nossas dúvidas que surgiram acerca do seu uso ao longo deste trabalho. Dessa forma, agradecemos à equipe do HPC por toda a ajuda fornecida e pela paciência com a qual todas as nossas perguntas foram respondidas.

Agradecemos também à todos os professores e funcionários da Escola Politécnica da USP, da Universidade Técnica de Darmstadt e da Universidade Técnica de Ilmenau por terem sido cruciais na nossa formação como engenheiros e pessoas e que, dessa maneira, permitiram que este trabalho pudesse ser executado.

Por fim, gostaríamos de agradecer à ambas as nossas famílias pela paciência que tiveram durante os longos fins de semana e feriados nos quais este texto foi escrito.

Obrigado,

Victor e Pedro

RESUMO

A tarefa de estimação de profundidades de objetos em vídeos tem ganhado importância crescente nas pesquisas relacionadas à área de visão computacional, devido às suas diversas aplicações. Muitos métodos utilizando desde visão estéreo até monocular já foram desenvolvidos e testados, mas poucos resultados existem para tratar do uso das relações temporais entre os quadros consecutivos de vídeos.

Ao longo deste trabalho, visando utilizar as mencionadas relações temporais, foi desenvolvida e treinaram uma rede neural convolucional, recursiva e siamesa para tratar do problema de estimação da profundidade de quadros de vídeos. Ela é composta por duas sub-redes, a de aproximação inicial e a de refinamento. A primeira tem uma arquitetura convolucional siamesa para extrair informações espaciais do par de imagens estéreo. A segunda utiliza camadas recursivas para relacionar as informações espaciais extraídas anteriormente com sua memória dos quadros precedentes, de forma a montar um mapa de profundidade da imagem.

Foram propostas três arquiteturas convolucionais distintas para a sub-rede de aproximação inicial, sendo cada uma baseada em uma rede convolucional consagrada. Foram treinadas versões da sub-rede com as partes convolucionais das redes VGG19, *Xception* e YOLO. Os resultados das três foram comparados e a arquitetura *Xception* foi a escolhida.

De forma análoga, três arquiteturas recursivas diferentes foram analisadas para a sub-rede de refinamento, sendo elas arquiteturas desenvolvidas pelos autores. Cada arquitetura se baseava no uso de camadas recursivas diferentes, sendo essas recursivas tradicionais, LSTM totalmente conectadas e LSTM convolucionais. Após a comparação dos resultados, a sub-rede recursiva tradicional foi a escolhida.

Todas as redes foram treinadas utilizando imagens da base de dados KITTI (GEIGER et al., 2013), organizadas em batches com sequências de quadros consecutivos dos vídeos presentes nessa biblioteca. Os resultados da rede final foram analisados e comparados com a tabela de colocações da base de dados KITTI para previsão de profundidade de imagens, mostrando que a arquitetura desenvolvida obteve ótimos resultados, sendo eles comparáveis e, em alguns pontos, superiores aos melhores colocados.

Devido a limitações na capacidade computacional disponível para o treinamento, a rede desenvolvida foi obrigada a operar apenas com tamanhos de imagens muito reduzidos. Mesmo assim, espera-se que o sucesso encontrado no trabalho com imagens pequenas pode ser expandido para imagens maiores e aplicações reais.

Palavras-chave: Visão computacional, Redes neurais convolucionais e recursivas, Processamento de imagens, Visão estéreo, Estimação da profundidade de imagens

ABSTRACT

The task of depth estimation of objects in videos has gained increasing importance in the computer vision field due to its various applications. Many methods using from stereo to monocular vision have already been developed and tested, but few results exist to address the use of temporal relationships between consecutive video frames.

Throughout this work, in order to use the mentioned temporal relations, the authors developed and trained a convolutional, recursive and siamese neural network to deal with the problem of depth estimation of video frames. It is composed of two subnets, the initial approximation network and the refinement network. The first consists in a siamese convolution architecture to extract spatial information from the stereo image pair. The second uses recursive layers to relate the spatial information extracted previously with its memory from the previous frames, in order to assemble a depth map of the image.

Three different convolutional architectures were proposed for the initial approximation sub-network, each one being based on a known convolution network. Versions of the subnet were trained with the convolutional parts of the VGG19, *Xception* and YOLO networks. The results of the three were compared and the architecture *Xception* was chosen.

Analogously, three different recursive architectures were analyzed for the refinement sub-network, and each of the architectures were developed by the authors. Each architecture was based on the use of different recursive layers, these being the traditional recursive, the fully connected LSTM and the convolutional LSTM. After the comparison of the results, the traditional recursive subnet was chosen.

All networks were trained using images from the KITTI dataset (GEIGER et al., 2013), organized into batches with consecutive frame sequences of the videos present in that library. The final network results were analyzed and compared with the KITTI dataset for predicting the depth of the images, showing that the architecture developed had excellent results, being comparable and in some points superior to the best ranked ones.

Due to limitations in the computational capacity available for the training, the developed network was forced to operate only with very small image sizes. Even so, it is expected that the success found in working with small images can be expanded to larger images and real-life applications.

Keywords: Computer vision, convolutional and recursive neural network, stereo vision, depth estimation

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

absErrorRel	Erro relativo absoluto (*Relative absolute Error*)
HPC	Computador de alto desempenho da USP (*High Performance Computing*)
iRMSE	Raiz do erro médio quadrado do inverso da profundidade (*Root-Mean Squared error of the inverse depth* em inglês)
KITTI	*Karlsruhe Institute of Tecnology and Toyota Technological Institute*
LSTM	Memória de longo curto prazo (*Long-Short-Term Memory*)
Relu	Unidade de Retificação Linear (*Rectified Linear Unit*)
SILog	Erro logarítmico de escala invariante (*Scale Invariant Logarithmic Error*)
sqErrorRel	Erro relativo quadrado (*Relative squared error*)
USP	Universidade de São Paulo
YOLO	"Você só olha uma vez"(*You Only Look Once*)
CPU	Unidade Central de Processamento (*Central Processing Unit*)
GPU	Unidade Processamento Gráfica (*Graphics Processing Unit*)

1 INTRODUÇÃO

1.1 Motivação

Este trabalho foi motivado pelo grande sucesso que redes neurais têm obtido nos últimos anos para solucionar problemas na área de visão computacional, tais como visão estéreo e fluxo ótico. Redes neurais têm melhorado o desempenho das mais diversas tarefas em diversas áreas do conhecimento, devido à sua alta capacidade de aproximação para problemas complexos e ao aumento da capacidade de processamento e da disponibilidade de dados em tempos recentes.

Um dos problemas amplamente pesquisados atualmente é o de estimação de profundidade de objetos em imagens e vídeos. Ele se fez relevante devido às suas inúmeras aplicações diretas e indiretas, principalmente na área de veículos autônomos, robôs móveis, reconstrução 3D de ambientes, realidade virtual e jogos eletrônicos. A aplicação de redes neurais para a resolução desse tipo de problema não é um conceito novo. Existem atualmente uma grande quantidade de pesquisas que utilizam desde os métodos mais tradicionais, que baseiam-se comumente na visão estéreo, usando duas imagens, de forma análoga à visão humana, conforme explicado por (SCHARSTEIN; SZELISKI, 2002) até a chamada visão monocular, que utiliza apenas uma imagem para extrair profundidade, como, por exemplo, a rede de (EIGEN; PUHRSCH; FERGUS, 2014).

1.2 Objetivos

Atualmente existem poucos trabalhos que exploram plenamente a informação temporal fornecida por quadros consecutivos de vídeos, sendo os poucos exemplos extremamente recentes, como o trabalho de (KUMAR; BHANDARKAR; PRASAD, 2018). Este projeto propõe, portanto, um novo método baseado em *deep learning*, que utiliza, além de camadas convolucionais, camadas recursivas, visando assim utilizar as mencionadas relações temporais para estimar a profundidade de objetos em vídeos em tempo real. A proposta dos autores ainda não foi amplamente testada pela comunidade científica e apresenta certo grau de inovação.

1.3 Organização do trabalho

Neste trabalho apresenta-se primeiramente uma análise de requisitos e critérios de avaliação da ideia proposta, de modo a verificar a eficiência da arquitetura de rede idealizada com

relação a outros métodos já testados. A análise e discussão dos objetivos e métricas escolhidos são detalhados no capítulo 2 deste texto.

Uma análise detalhada do Estado da Arte para resolução do problema proposto é mostrada no capítulo 3 deste trabalho. Para o desenvolvimento deste trabalho foram estudados desde os conceitos mais básicos de redes neurais, até os artigos mais especializados no problema de estimação de profundidades. A partir desse estudo, definiu-se o uso da base de dados KITTI (GEIGER et al., 2013) para o treinamento e validação do método proposto, devido à sua grande popularidade entre os pesquisadores da área, ao seu grande acervo de imagens e também à sua excelente documentação.

No capítulo 4, a proposta de arquitetura de rede deste projeto é detalhada e as suas possíveis variações explicadas. No capítulo 5, uma metodologia baseada no uso de redes convolucionais pré-treinadas, diferentes arquiteturas e etapas de treinamento distintas é detalhada de modo a obter-se o melhor resultado possível.

Por fim, os resultados obtidos são analisados e comparados aos requisitos estabelecidos no capítulo 2. A eficácia da ideia proposta é verificada dentro das limitações presentes durante a execução deste projeto. O capítulo 6 mostra de maneira detalhada os resultados deste trabalho, as conclusões são apresentadas no capítulo 7.

2 ANÁLISE DE REQUISITOS

O principal objetivo deste trabalho é definir se a proposta de arquitetura de rede que faz uso de camadas recursivas, capazes de estabelecer relações temporais, em conjunto com as comumente utilizadas para processamento de imagens redes convolucionais apresenta melhores resultados na estimação de profundidade de objetos em vídeos do que redes convencionais que analisam cada quadro do vídeo individualmente.

Os requisitos a serem atendidos pela rede neural para alcançar o objetivo proposto são relacionados à precisão das profundidades estimadas pela rede e o tempo de resposta da mesma depois de treinada. Uma precisão mínima nas previsões de profundidade é necessária para qualquer aplicação na qual a rede seja utilizada, mas essa precisão pode variar muito de uma aplicação para outra.

No caso do tempo de resposta, o requisito exigido depende muito da aplicação, uma vez que para aplicações de tempo real, como no caso de carros autônomos, qualquer ação do sistema que dependa das informações de profundidade só poderá ser efetuada após o cálculo das mesmas pela rede. Para o caso de filmagens com 10 quadros por segundo (como é o caso da base de dados utilizada neste trabalho), um tempo de processamento de cada quadro superior a 0,1 segundos faria com que as informações do quadro anterior não estivessem disponíveis a tempo para calcular a profundidade do posterior. Desse modo, caso a rede demore mais do que 0,1 segundos no processamento de cada quadro, não seria possível fornecer uma resposta em tempo real para todos os quadros dessa filmagem.

Mesmo assim, nem todas as aplicações exigem uma frequência tão alta no fornecimento das informações de profundidade pela rede, sendo que muitas delas, como a reconstrução 3D de ambientes, não precisam sempre de respostas em tempo real.

Dessa forma, para que seja possível avaliar esses dois requisitos de maneira objetiva, os resultados obtidos por este trabalho são comparados àqueles que apresentaram o melhor desempenho na base de dados KITTI (GEIGER et al., 2013). Na Tabela 1, retirada do site da mesma, é mostrada a colocação dos melhores resultados obtidos na tarefa de estimação de profundidade quando do desenvolvimento deste projeto.

Nessa tabela, as colunas *SILog*, *sqErrorRel*, *absErrorRel* e *iRMSE* mostram diferentes parâmetros para medir o desempenho dos métodos, sendo *SILog* o erro logarítmico de escala invariante, *sqErrorRel* o erro relativo quadrado, *absErrorRel* o erro relativo absoluto e *iRMSE* a raiz do erro médio quadrado do inverso da profundidade. Esses erros são definidos na seção 3.3. A coluna Tempo mostra o tempo médio necessário para o método estimar a profundidade de um único quadro. É importante acrescentar que todos o métodos apresentados foram processados em GPUs. É relevante ressaltar também que

na análise do desempenho de cada um dos métodos é dada maior importância para o desempenho com relação aos erros do que ao mencionado tempo.

É também considerado na análise do cumprimento dos requisitos a limitação da capacidade computacional disponível para a execução do método proposto neste projeto.

Por fim, como requisito adicional, é avaliada a precisão da estimativa de profundidade em cada quadro de um mesmo vídeo, de modo a verificar se a inclusão de camadas recursivas capazes de guardar relações temporais ajuda na redução do erro com o passar do tempo, como seria o esperado.

Tabela 1 – Os vinte e seis métodos mais bem cotados na base de dados KITTI 2015 de previsão de profundidade em outubro de 2018

Método	SILog	sqErrorRel	absErrorRel	iRMSE	Tempo
DL_61 (DORN)	11.77	2.23 %	8.78 %	12.98	0.5 s
FUSEBS_ROB	13.41	2.86 %	10.60 %	15.06	0.16 s
DORN_ROB	13.53	3.06 %	10.35 %	15.96	2 s
FUSION_ROB	13.90	3.14 %	11.04 %	15.69	0.2 s
BMMNet	14.37	5.10 %	10.92 %	15.51	0.1 s
DABC_ROB	14.49	4.08 %	12.72 %	15.53	0.7 s
APMoE_base_ROB	14.74	3.88 %	11.74 %	15.63	0.2 s
CSWS_E_ROB	14.85	3.48 %	11.84 %	16.38	0.2 s
VNET_ROB	14.89	3.73 %	12.77 %	17.43	1 s
SW_ROB	14.94	3.75 %	12.81 %	17.53	1 s
UReUFo_ROB	15.20	3.83 %	11.54 %	16.15	0.04 s
DHGRL	15.47	4.04 %	12.52 %	15.72	0.2 s
FCRN_ROB	15.93	4.06 %	12.10 %	16.51	0.2 s
semiDepth_pure	16.25	4.65 %	12.33 %	17.57	0.02 s
MMD_ROB	16.89	4.32 %	12.09 %	834.59	0.1 s
AI Mono Tech.	17.21	6.98 %	13.60 %	16.80	0.04 s
TASKNetV1_ROB	17.29	4.75 %	13.39 %	18.20	0.18 s
Modu_selfdriving_ROB	17.54	7.69 %	14.61 %	17.77	0.1 s
semiDepth_ROB	17.78	5.02 %	13.09 %	23.70	0.02 s
BRNet_ROB	18.16	3.90 %	13.53 %	17.95	0.19 s
FCRN	22.91	10.95 %	18.33 %	24.96	0.1 s
UNET_depth_ROB	24.90	7.58 %	21.24 %	35.75	0.1 s
DSA	31.09	6.09 %	14.19 %	65.97	0.1 s
RVGNet_ROB	37.71	10.66 %	23.39 %	62.48	0.3 s
RRCNN_ROB	39.13	20568.70 %	1325.26 %	100.09	1s
RVGNet	40.91	13.35 %	28.03 %	44.54	0.3 s

Fonte: (UHRIG et al., 2017)

3 ESTADO DA ARTE

A extração de informações de profundidade de imagens é um campo de estudo muito importante devido às suas inúmeras aplicações que incluem: carros autônomos, navegação de robôs, reconstrução 3D de ambientes, realidade virtual e jogos eletrônicos. Diversos algoritmos já foram desenvolvidos com esse objetivo, porém apenas mais recentemente, devido ao aumento do poder de processamento dos computadores e da disponibilidade de dados (NG, 2018), os métodos de redes neurais e *deep learning* passaram a ser utilizados em tais tarefas.

Este capítulo tem por objetivo apresentar uma breve introdução teórica sobre de redes neurais artificiais, assim como apresentar uma visão geral sobre a extração de informações de profundidade de imagens, como encontrado atualmente na literatura, tanto para o caso de visão estéreo como monocular.

3.1 Técnicas de redes neurais e deep learning

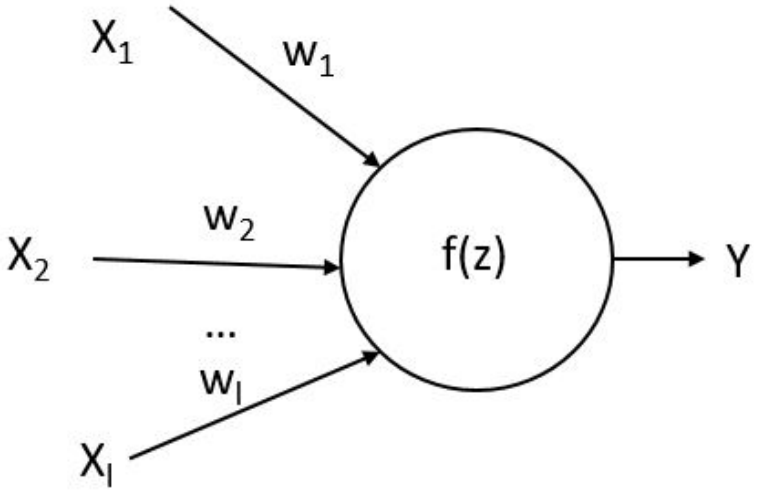
Redes Neurais e *deep learning* consistem de métodos consagrados de aprendizado de máquinas, cujas principais inspirações provém da biologia do cérebro humano e do seu modo de aprender. Naturalmente que os modelos matemáticos que surgiram têm pouco a ver com o funcionamento real das respectivas inspirações biológicas, porém os resultados e aplicações desses algoritmos são muitos e bastante significativos, além de terem revolucionado diversos campos, tais como processamento de imagens e fala. O objetivo dessa seção é apresentar uma introdução ao tema baseada no curso *online* de (NG, 2018) e no texto de (ADAMY, 2015). Como na biologia, a estrutura mais básica desses algoritmos de aprendizado são os neurônios, cujo modelo é apresentado na Figura 1.

Um neurônio possui várias entradas que são ponderadas por pesos e enviesadas por um viés, antes de passar por uma função de ativação que retorna uma saída. Portanto, a função de um neurônio pode ser descrita pela seguinte equação, onde $\mathbf{x}$ é um vetor com as variáveis de entrada, $\mathbf{w}$ é um vetor com os pesos, g representa a função de ativação, b é o viés e a é a sua saída numérica.

$$a = g(\mathbf{w}^T\mathbf{x} + b) = g(z) \tag{3.1}$$

As funções de ativação podem assumir diversas formas, sendo possível provar que ao se utilizar uma de característica não-linear, a rede como um todo pode aproximar e aprender comportamentos extremamente complexos. As funções que normalmente são

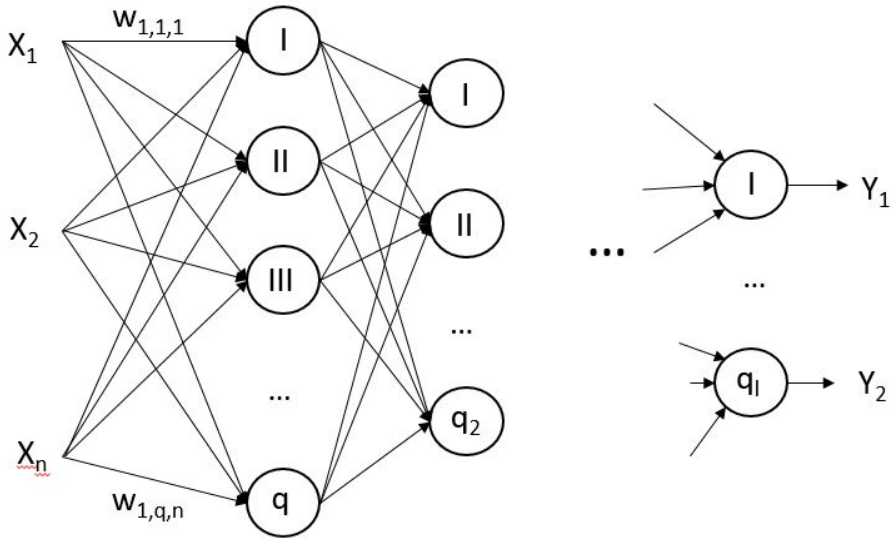
Figura 1 – O modelo de um neurônio



mais utilizadas são a sigmoide, a tangente hiperbólica, a unidade de retificação linear conhecida como *ReLu* (*Rectified Linear Unit* em inglês) e a sua variação a *Leaky ReLu*.

Ao se aglomerar diversos neurônios com as mesmas entradas, cada um gerando saídas diferentes, cria-se uma camada da rede neural. A complexidade dos problemas que a rede pode resolver está normalmente associada ao número de camadas da mesma. Quando da utilização de um grande número de camadas (Figura 2) usa-se o termo *deep learning*.

Figura 2 – Rede Neural e suas camadas



Na terminologia de redes é importante que se diferenciem os termos conhecidos como parâmetros da rede, sendo eles os pesos $\mathbf{w}$, os vieses b, as entradas $\mathbf{X}$ e as saídas $\hat{\mathbf{y}}$, daqueles conhecidos como hiperparâmetros, sendo eles o número de camadas da rede, o número de neurônios por camada e até mesmo a taxa de aprendizagem α, que será

explicada no próximo item.

3.1.1 Propagação para trás - o processo de aprendizagem

Com o objetivo de que as saídas $\hat{\mathbf{y}}$ apresentem o comportamento desejado $\mathbf{y}$, é necessário o treinamento da rede neural. Ele consiste na determinação dos valores dos pesos $\mathbf{w}$ e b para todos os neurônios da rede com o auxílio de dados de aprendizado. A forma mais usual de realização deste processo é descrita na literatura como Propagação para trás (ou *Backpropagation* em inglês).

Uma explicação sucinta desse método é dada a seguir, sendo ela baseada novamente nas duas referências do item anterior: (NG, 2018) e (ADAMY, 2015).

Primeiramente, para facilitar as notações daqui para frente, define-se n, que corresponde ao número de entradas dos neurônios da camada l. Assim, para o neurônio i da camada l, tem-se as seguintes equações que representam o seu funcionamento:

$$\begin{aligned} \mathbf{w_i}^{l^T} &= (w^l_{i,1}, w^l_{i,2}, ..., w^l_{i,n}) \\ \mathbf{x_i}^{l^T} &= (x^l_{i,1}, x^l_{i,2}, ..., x^l_{i,n}) \\ z^l_i &= \mathbf{w_i}^{l^T} \mathbf{x_i}^{l^T} + b^l_i \end{aligned} \tag{3.2}$$

onde o índice T denota a operação de transposição do vetor, os índices l e i definem, respectivamente, a camada e o neurônio daquela camada ao qual a variável ou vetor pertence.

Em seguida, supondo uma rede neural como a da Figura 2, pode ser dito, de forma simplificada, que a rede deve aprender a realizar a operação da função f abaixo:

$$\mathbf{y} = f(\mathbf{X}) \tag{3.3}$$

onde f representa uma função que relaciona corretamente o vetor com todas as entradas da rede $\mathbf{X}$ com as saídas desejadas de todos os neurônios da última camada da rede $\mathbf{y}$.

Deve-se definir uma função que calcula o erro entre o resultado desejado $\mathbf{y}$ e o que a rede neural realmente fornece $\hat{\mathbf{y}}$. Os dois erros mais frequentemente utilizados são o quadrático (3.4) e o logarítmico (3.5).

$$J = \frac{1}{2m} \sum_{i=0}^{m} (\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)})^2 \tag{3.4}$$

$$J = -\frac{1}{m} \sum_{i=0}^{m} (\mathbf{y}^{(i)} \log \hat{\mathbf{y}}^{(i)} - (1 - \mathbf{y}^{(i)}) \log(1 - \hat{\mathbf{y}}^{(i)})) \tag{3.5}$$

onde J representa a função de erro, m é o número de exemplos de treinamento e i é um subscrito que define um exemplo específico.

O objetivo do treinamento da rede é minimizar a função de erro escolhida para os dados de treinamento. Isso é feito ao se achar a derivada dos erros em função dos pesos e igualá-la a zero, como mostrado em (3.6), onde $\mathbf{W}$ representa a matriz que contém todos os pesos da rede.

$$\frac{\partial J}{\partial \mathbf{W}} = 0 \tag{3.6}$$

O cálculo dessa derivada é feito de maneira iterativa através do conhecido Método do Gradiente Descendente. Na equação (3.7) está ilustrado o processo de atualização gradual dos pesos da rede, no qual α representa a taxa de aprendizagem, que é definida em função do problema a ser tratado, influenciando a precisão e a velocidade do processo de aprendizagem:

$$\mathbf{w^l} = \mathbf{w^l} - \alpha \frac{\partial J}{\partial \mathbf{w^l}} \tag{3.7}$$

Naturalmente, para o caso frequente de uma rede neural com mais de uma camada, a derivada do erro deve ser propagada para cada neurônio das diferentes camadas. Este processo é conhecido como Propagação para trás. Não é o objetivo deste texto deduzir as fórmulas para a execução desse processo, dado que isso é facilmente encontrado na literatura. Abaixo encontra-se um esquema que ilustra o método de acordo com (ADAMY, 2015), onde l representa o número da camada atual, L o número total de camadas da rede, n_l o número de entradas dos neurônios da camada l, j a posição do neurônio na camada e N a quantidade de neurônios em uma camada.

Dada a regra de aprendizagem para cada peso de cada neurônio:

$$w_{i,j}^l = w_{i,j}^l - \alpha \frac{\partial J}{\partial w_{i,j}^l} \tag{3.8}$$

com

$$\frac{\partial J}{\partial w_{i,j}^l} = \delta_i^l y^{l-1} \tag{3.9}$$

onde

$$\delta_i^l = \begin{cases} \frac{\partial J}{\partial \hat{y}_i} g_L^{'}(z_i^N), \text{ para a última camada} \\ (\sum_{l=1}^{N} \delta_i^{l+1} w_{j,n}^{l+1}) g_L^{'}(z_i^l), \text{ para as demais} \end{cases} \tag{3.10}$$

É possível então calcular iterativamente os pesos w_i^l até que o erro seja considerado mínimo ou dentro de uma margem aceitável estipulada previamente.

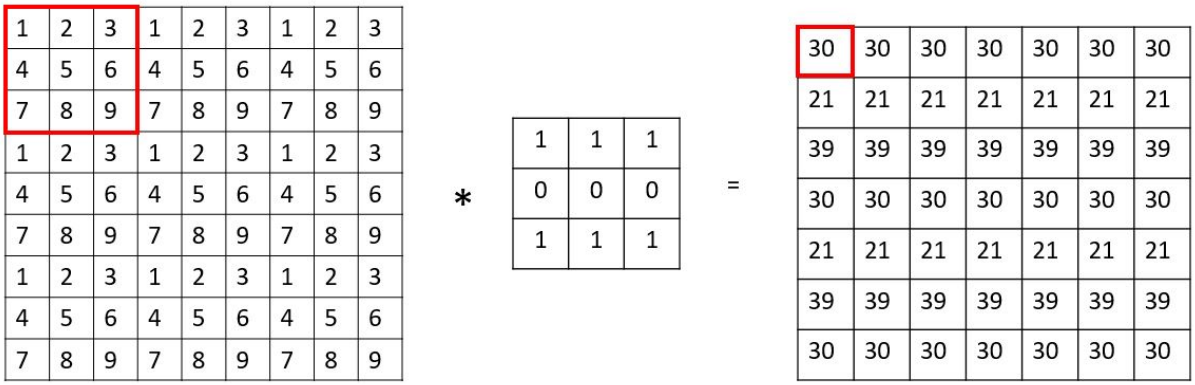
A quantidade de dados de treinamento deve ser suficientemente grande para que seja possível um aprendizado efetivo. É necessário, por vezes, ter também uma grande quantidade de dados diferentes dos usados para treinamento, para testar a eficiência de treinamento.

Ao se utilizar o método aqui descrito torna-se possível a rede neural para realizar tarefas de grande complexidade. Porém, para o caso deste trabalho, uma combinação de diferentes tipos de redes e métodos de otimização será utilizada, sendo todos esses explicados nas próximas seções.

3.1.2 Redes Convolucionais

Uma das aplicações mais famosas de redes tipo *deep learning* se dá no âmbito do processamento de imagens. Desde reconhecimento de faces até a estimação de distâncias e padrões em imagens, o desenvolvimento desse campo avançou muito nos últimos anos e isso se deveu principalmente ao uso das redes neurais convolucionais. Uma breve explicação do seu funcionamento, baseada em (NG, 2018), é dada a seguir.

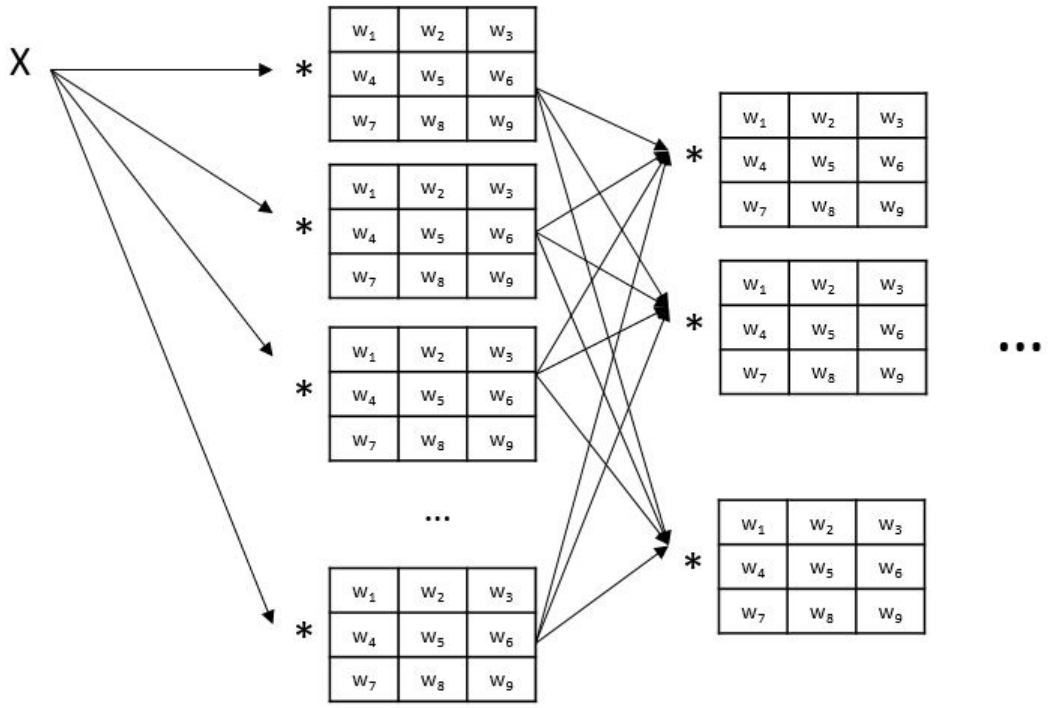
Figura 3 – Convolução de uma Imagem com filtro de dimensão 3x3



Toda a estrutura das redes de convolução é baseada na operação matemática apresentada na Figura 3. Como exemplo a imagem, no caso de dimensão 9x9, e um filtro, de dimensão 3x3. Primeiramente multiplicam-se todos os termos no quadrado vermelho com os pesos do filtro na posição correspondente e soma-os, configurando o resultado do primeiro pixel da nova imagem na posição equivalente ao pixel da posição central onde o filtro está sendo aplicado. Ao se reproduzir o mesmo processo e se deslocar o filtro através da imagem, é possível obter todos os resultados mostrados na Figura 3. É possível observar que na operação de convolução a imagem de saída é menor que a original. Essa redução

depende do tamanho do filtro e pode ser mitigada usando uma técnica conhecida como *padding*, que não será explicada, uma vez que esse não é o foco deste trabalho.

Figura 4 – Esquema de rede convolucional



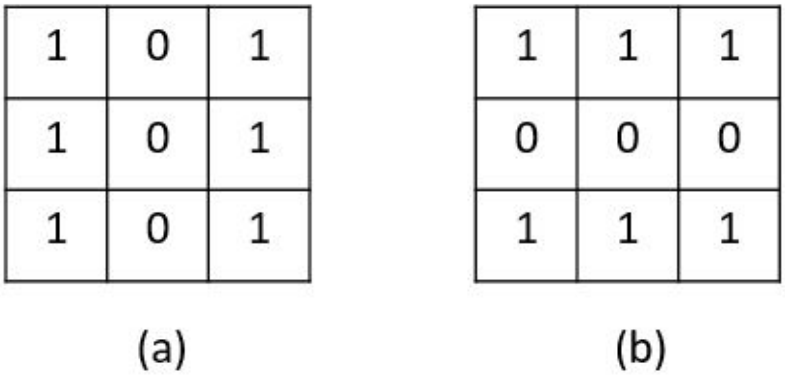
Se considerarmos uma situação como a da Figura 4, na qual cada um dos filtros constitui um neurônio, cujos valores de **w** configuram os pesos dos mesmos, e **X** uma imagem de entrada, produzimos uma rede neural convolucional. Cada camada da rede então será treinada para reconhecer diferentes padrões nas imagens de acordo com as necessidades do problema. Exemplos de filtros comums na área de processamento de imagens, com dimensão 3x3, são mostrados na Figura 5. É importante notar que esse tipo de filtro pronto não é usado nesse trabalho uma vez que as redes neurais aprendem de forma sozinhas durante o seu treinamento os pesos presentes em cada posição dos seus filtros de convolução.

Em aplicações reais, as dimensões tanto das imagens como dos filtros podem ser maiores, o que torna o tempo de processamento um fator determinante para o treinamento de redes convolucionais. Além disso, mais uma vez, é necessário ter disponível muitas imagens de treinamento para garantir a eficiência das redes.

Este tipo de arquitetura é exaustivamente utilizado atualmente para realizar o processo de correspondências em imagens estéreo e, assim, calcular as profundidades da cena, como será mostrado na seção 3.2.

Existem alguns métodos específicos usados para se obter melhores resultados em redes convolucionais, sendo que dois deles são explicados abaixo.

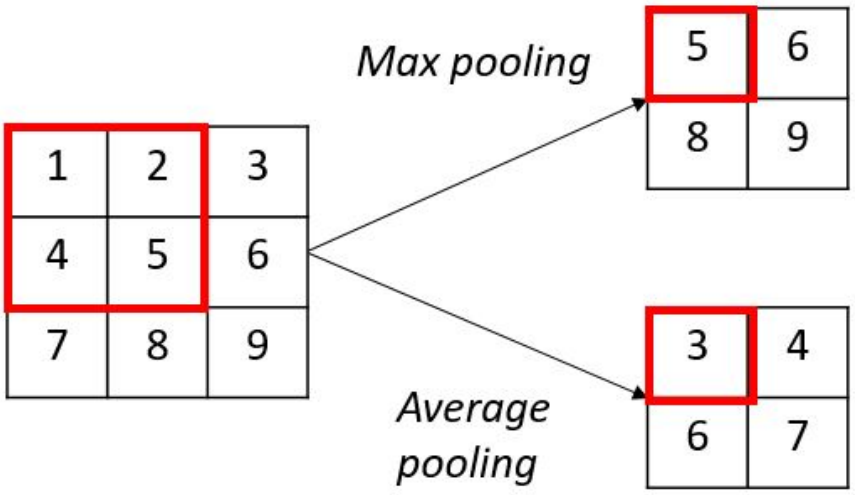
Figura 5 – (a) Filtro para detecção de bordas verticais; (b) Filtro para detecção de bordas horizontais



Pooling

Em uma rede neural convolucional é possível utilizar operações de *pooling* para reduzir o número de neurônios necessários nas suas camadas posteriores e, desse modo, aumentar a velocidade de processamento e tornar a detecção de padrões mais robusta (NG, 2018). Existem dois tipos principais de *pooling* que são bastante utilizados, nota-se que o motivo de seu bom funcionamento ainda é bastante discutido. Eles são *max pooling* e *average pooling*. Os dois casos são mostrados na Figura 6 e explicados a seguir.

Figura 6 – Operações de *pooling*



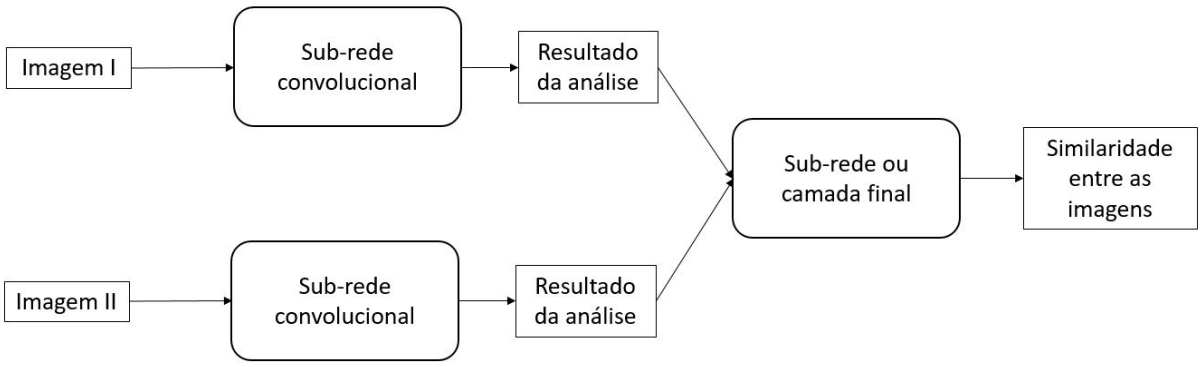
Na operação de *max pooling* ilustrada na Figura 6, o valor de saída de uma janela é assumido como o valor máximo correspondente na imagem resultante, enquanto no *average pooling* a saída é a média dos valores de imagem dentro da janela. Assim a imagem resultante se torna menor e algumas de suas características, no caso de *max pooling*, são ressaltadas.

É importante também dizer que o tamanho da janela escolhida e o passo dela (número de pixels que a janela se desloca sobre a imagem para novamente executar a operação desejada) são hiperparâmetros da rede que devem ser escolhidos junto com os demais, tal como o número de camadas, antes de se realizar qualquer treinamento.

Redes siamesas

Redes convolucionais siamesas são bastante utilizadas atualmente quando se pretende verificar a similaridade entre um par de imagens. Nelas, a rede neural é dividida entre duas sub-redes que realizam o processamento individual de cada uma das imagens. Por fim, o resultado da análise de cada uma das sub-redes é usado como entrada para uma única rede (ou camada) que realiza a desejada verificação de similaridade. Na Figura 7 é ilustrada de maneira simplificada a ideia acima explicada.

Figura 7 – Esquema de redes siamesas



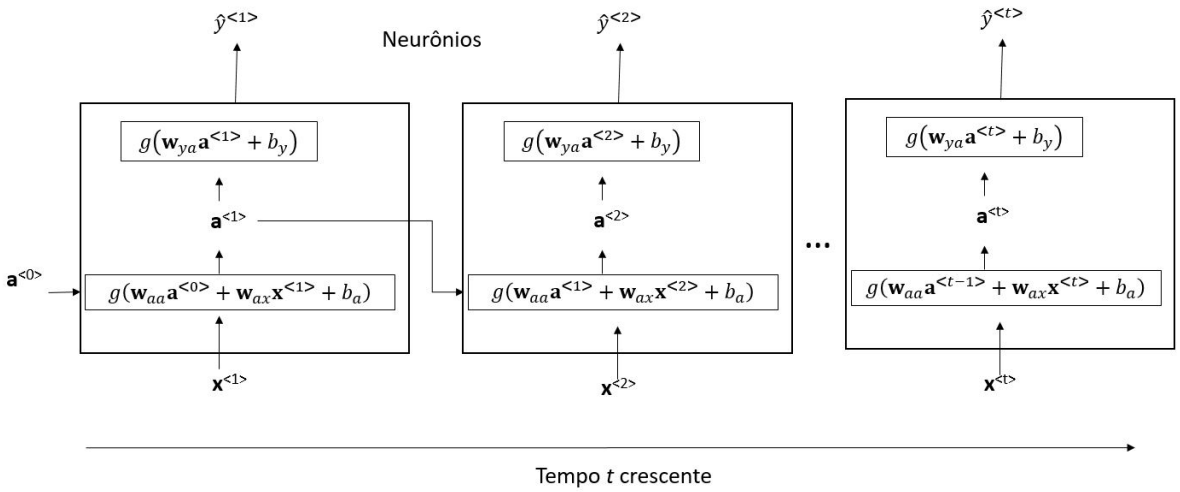
3.1.3 Redes Recursivas

Redes tipo *deep learning* também são hoje muito utilizadas para processar dados que tem uma dependência inerente ao tempo, seja áudio, fala ou vídeo. Reconhecimento de voz, construção de frases e traduções feitas por computador são feitas com o uso de redes neurais recursivas. Uma descrição sucinta do funcionamento deste tipo de rede neural é dada a seguir.

Para um vetor de entrada $\mathbf{x}$ que varia no tempo e uma saída $\mathbf{y}$ com as mesmas propriedades, ocorre que os resultados de um tempo t+1 são dependentes daqueles de um tempo anterior t. Na Figura 8 é ilustrado o modelo de uma rede neural recursiva que lida com esse tipo de problema.

O vetor $\mathbf{a}^{<t>}$ representa a ativação dos neurônios da rede no instante de tempo t.

Figura 8 – Rede neural recursiva



As equações generalizadas para esse tipo de rede neural são definidas a seguir:

$$\begin{aligned} \mathbf{a}^{<t>} &= g(\mathbf{w}_{aa}\mathbf{a}^{<t-1>} + \mathbf{w}_{ax}\mathbf{x}^{<t>} + b_a) \\ \hat{\mathbf{y}}^{<t>} &= g(\mathbf{w}_{ya}\mathbf{a}^{<t>} + b_y) \end{aligned} \tag{3.11}$$

Como é possível observar, a informação adquirida em um tempo anterior é propagada para as iterações posteriores através das ativações no instante de tempo anterior $\mathrm{a}^{<\mathrm{t\text{-}1}>}$, o que resolve o problema da dependência temporal das entradas. Da mesma forma que é feita para outros tipos de redes, é possível concatenar mais de uma camada com neurônios recursivos, porém, como mencionado por (NG, 2018), poucas camadas já são o suficiente para resolver satisfatoriamente a grande maioria dos problemas.

Para o treinamento desse tipo de rede, aplica-se o método de Propagação para trás através do tempo que é similar ao método de propagação para trás apresentado anteriormente, sendo que novamente os pesos $\mathbf{w}$ e b são determinados no processo.

3.1.4 Redes Recursivas LSTM (Long Short-Term Memory)

Um outro tipo de camada recursiva mais complexa é a chamada LSTM, *Long Short-Term Memory*, também conhecida como LSTM totalmente conectada. Elas foram desenvolvidas especificamente para compensar um problema comum das redes recursivas tradicionais: aprender dependências temporais de longo prazo (HOCHREITER; SCHMIDHUBER, 1997). As redes LSTM são capazes de aprender esse tipo de dependência temporal devido à sua capacidade de armazenar informações por um tempo arbitrariamente longo aprendido pela própria rede.

Uma camada LSTM é composta por quatro elementos: a célula, o portão de entradas, o portão de saídas e o portão do esquecimento. O portão de entradas controla

quanto a entrada mais recente da LSTM afeta os valores armazenados na célula. O portão de saídas controla como os valores atualmente armazenados na célula afetam a saída da camada LSTM. O portão do esquecimento controla quanto dos valores armazenados na célula serão esquecidos (descartados). A célula armazena os valores das entradas anteriores que passaram pelo portão de entradas e eles lá permanecem até serem removidos pelo portão do esquecimento. Dessa forma, a rede é capaz de manter informações específicas armazenadas na sua célula por um tempo muito mais longo do que o observado em redes recursivas tradicionais (HOCHREITER; SCHMIDHUBER, 1997).

As equações a seguir controlam o funcionamento de uma rede LSTM:

$$\begin{aligned}
\mathbf{f}^t &= \sigma(\mathbf{W}_f[\mathbf{h}^{t-1}, \mathbf{X}^t] + b_f) \\
\mathbf{i}^t &= \sigma(\mathbf{W}_i[\mathbf{h}^{t-1}, \mathbf{X}^t] + b_0) \\
\mathbf{O}^t &= \sigma(\mathbf{W}_O[\mathbf{h}^{t-1}, \mathbf{X}^t] + b_O) \\
\mathbf{C}^t &= \mathbf{f}_t\mathbf{C}^{t-1} + \mathbf{i}^t \tanh\left(\mathbf{W}_C[\mathbf{h}^{t-1}, \mathbf{X}^t] + b_C\right) \\
\mathbf{h}^t &= \mathbf{O}^t \tanh\left(\mathbf{C}^t\right)
\end{aligned} \tag{3.12}$$

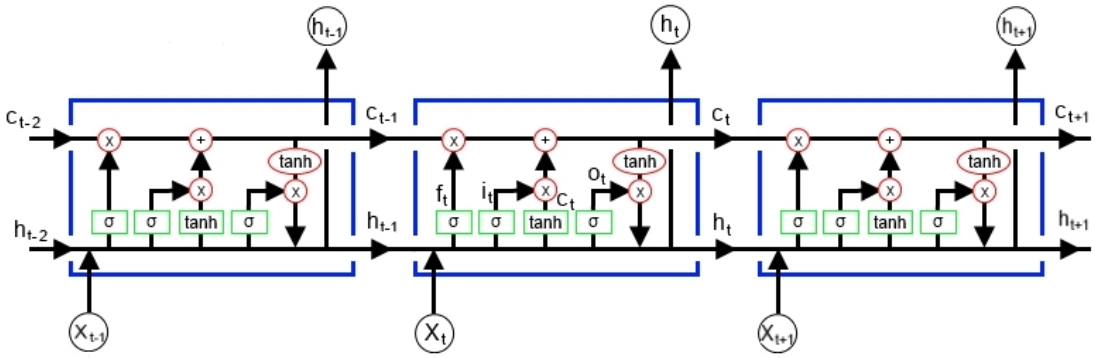
onde $\mathbf{X}^{\mathrm{t}}$ representa as entradas e $\mathbf{h}^{\mathrm{t}}$ representa as saídas da rede LSTM no instante de tempo t, as operações vetoriais identificadas como $\mathbf{f}^{\mathbf{t}}$, $\mathbf{i}^{\mathbf{t}}$, $\mathbf{O}^{\mathbf{t}}$ e $\mathbf{C}^{\mathbf{t}}$ representam respectivamente o portão de esquecimento, portão de entradas, portão de saídas e célula no instante t. As interações entre os portões e a célula são controladas por pesos aprendidos pela própria rede ($\mathbf{W}_{\mathrm{f}}$, $\mathbf{W}_{\mathrm{i}}$, $\mathbf{W}_{\mathrm{O}}$, $\mathbf{W}_{\mathrm{C}}$) de forma análoga ao aprendizado de redes tradicionais. Os portões costumam operar com funções de ativação sigmoide σ, enquanto a célula utiliza funções de ativação tanh.

Na Figura 9 é ilustrado graficamente o funcionamento de uma rede LSTM. Os círculos vermelhos representam operações termo a termo das grandezas vetoriais. Os retângulos verdes representam funções de ativação da rede com parâmetros que podem ser aprendidos, sendo elas sigmoide σ ou tanh. Assim como na representação utilizada para descrever redes recursivas convencionais, na Figura é ilustrada apenas uma única camada LSTM e suas interações com as entradas $\mathbf{X}$, as saídas $\mathbf{h}$ e com os seus valores de célula $\mathbf{C}$ e de saída $\mathbf{h}$ anteriores e futuros.

Uma desvantagem das redes LSTM é que a sua elevada complexidade introduz muito mais pesos que devem ser aprendidos pela rede do que em outros tipos de camadas, como convolucionais ou até outros tiipos de camadas recursivas. Esse fato contribui para tornar mais lento e oneroso o treinamento das mesmas comparado ao treinamento de redes recursivas tradicionais.

No treinamento dessas redes o método de propagação para trás através do tempo pode ser utilizado assim como em redes recursivas tradicionais.

Figura 9 – Ilustração esquemática do funcionamento de uma rede LSTM



3.1.5 Redes Recursivas LSTM convolucionais

A camada convolucional LSTM é distinta da LSTM tradicional (totalmente conectada), pois ela considera as relações espaciais entre as posições da entrada, assim como, uma camada de convolução tradicional, além de apresentar as capacidades de memória das camadas LSTM tradicionais. Ela foi desenvolvida para aplicações que fazem uso de informações espaciais juntamente com informações de memória. Para essas aplicações, costumava-se utilizar camadas convolucionais intercaladas com camadas LSTM tradicionais, mas percebeu-se que as camadas LSTM comuns apresentam muita redundância quando lidam com informações espaciais (SHI et al., 2015).

Para solucionar esse problema foi desenvolvida a camada LSTM convolucional, na qual as transformações de entrada e transformações recorrentes são ambas convolucionais. Para atingir esse resultado são utilizados operadores convolutivos nas transições de estado para estado e entrada para estado. Dessa forma, os quatro elementos principais da camada LSTM original permanecem quase inalteradas. As equações abaixo definem o funcionamento da rede LSTM convolucional de forma análoga a rede LSTM tradicional.

$$
\begin{aligned}
\mathbf{f}^t &= \sigma(\mathbf{W}_{hf} * \mathbf{h}^{t-1} + \mathbf{W}_{xf} * \mathbf{X}^t + \mathbf{W}_{cf} * C^{t-1} + b_f) \\
\mathbf{i}^t &= \sigma(\mathbf{W}_{hi} * \mathbf{h}^{t-1} + \mathbf{W}_{xi} * \mathbf{X}^t + \mathbf{W}_{ci} * C^{t-1} + b_i) \\
\mathbf{O}^t &= \sigma(\mathbf{W}_{hO} * \mathbf{h}^{t-1} + \mathbf{W}_{xo} * \mathbf{X}^t + \mathbf{W}_{co} * C^{t-1} + b_o) \\
\mathbf{C}^t &= \mathbf{f^t C}^{t-1} + \mathbf{i^t} \tanh(\mathbf{W}_{hc} * \mathbf{h}^{t-1} + \mathbf{W}_{xc} * \mathbf{X}^t + b_c) \\
\mathbf{h}^t &= \mathbf{O}^t \tanh(C^t)
\end{aligned}
\tag{3.13}
$$

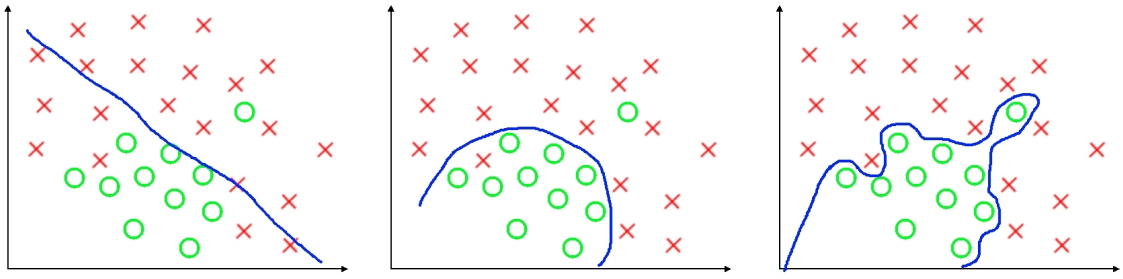
As únicas diferenças com relação à camada LSTM original são a presença das operações denotadas por *, que representam operações de convolução e que todas as entradas $\mathbf{X}^t$, valores de célula C^t, saídas $\mathbf{h}^t$ e portões de entrada i^t e de esquecimento f^t são tensores 3D onde duas dimensões são espaciais (colunas e linhas). O aprendizado e a operação dessa camada se dá de forma análoga à camada LSTM tradicional.

3.1.6 Métodos de regularização e otimização de treinamento

Ao longo do tempo foram desenvolvidos diversos métodos que modificam os sistemas tradicionais de aprendizado de redes neurais, com o objetivo de reduzir o erro obtido no conjunto de dados usados para testar a rede. A principal causa de erros é o chamado *overfitting*. Esses métodos são conhecidos como métodos de regularização (GOODFELLOW; BENGIO; COURVILLE, 2016).

O *overfitting* consiste em uma adaptação excessiva dos pesos da rede em especificidades da base de dados utilizada para o treinamento, o que acaba reduzindo a sua capacidade de generalização e deteriorando o seu resultado na base de dados de teste, mesmo que aumente o seu sucesso na base de treinamento. Em geral o problema de *overfitting* é maior quando a base de treinamento não é grande o bastante para conter toda a variedade do campo analisado ou quando grande parte da base de treinamento apresenta um viés que não está presente na situação generalizada na qual a rede deverá trabalhar. A estratégia tradicional usada no treinamento de redes neurais de tentar minimizar o erro no treinamento, esperando que isso reduza o erro nos testes, não consegue lidar com o problema de *overfitting*.

Figura 10 – Representação simplificada num espaço 2D do funcionamento de uma rede neural com underfitting (esquerda), sem overfitting ou underfitting (centro) e com overfitting (direita)



Na Figura 10 é ilustrado de forma simplificada o funcionamento de uma rede neural com *overfitting*, com *underfitting* e sem nenhum desses problemas. *Underfitting* consiste na situação inversa do *overfitting*, quando uma rede não é capaz de se adaptar à base de treinamento obtendo resultados ruins tanto no treinamento quanto nos testes. Ele geralmente significa que a arquitetura de rede utilizada não é apropriada para o problema em questão.

No exemplo da Figura 10, a tarefa da rede neural é aprender a diferenciar os círculos verdes (casos positivos) dos xis vermelhos (casos negativos), utilizando apenas as coordenadas X e Y de ambos no espaço 2D. Para isso, a rede seria treinada utilizando

a base de dados mostrada na Figura, ou seja, o conjunto de círculos e xis que podemos observar. A rede da esquerda não foi capaz de traçar uma fronteira que acompanhasse satisfatoriamente a distribuição de círculos e xis na base de treinamento, apresentando um resultado ruim no treinamento e provavelmente em testes. A rede da direita foi capaz de traçar uma fronteira extremamente complexa de forma a obter um resultado excelente na base de treinamento. Pode-se observar que a fronteira separa perfeitamente os círculos dos xis. Porém essa fronteira provavelmente não se sairia tão bem fora do espaço de treinamento pois, supondo que a distribuição real de xis e círculos seja razoavelmente regular, a "península" criada pela fronteira para capturar o círculo anômalo mais distante dos outros provavelmente englobaria vários xis, reduzindo a precisão da rede. Da mesma forma, a "baía" formada para evitar o xis anômalo no meio dos círculos pode fazer com que a rede perdesse vários círculos presentes na região. A rede central, porém, definiu uma fronteira complexa o suficiente para englobar a maioria dos xis, mas ainda muito mais simples do que a fronteira da direita. Se supusermos uma distribuição regular de xis e círculos, é fácil imaginar que a rede central se sairia melhor do que a rede da direita, mesmo que o resultado da rede central na base de treinamento tenha sido inferior.

Por causa desse tipo de problema, foram desenvolvidos diversos métodos capazes de combatê-los. Dentre eles, os métodos *Dropout* e Normalização de *Batch* são utilizados nesse trabalho. Ambos são explicados nos tópicos a seguir.

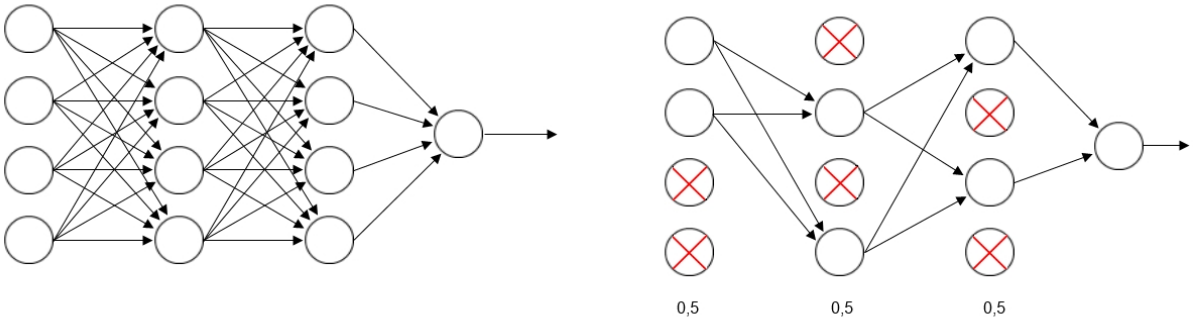
Dropout

O método de regularização *Dropout* é muito utilizado na área de visão computacional. Ele consiste em "remover" de forma aleatória vários neurônios da rede durante cada passo de treinamento. Para cada camada da rede neural, é definida a probabilidade de cada um de seus neurônios serem removidos. Essa remoção de vários neurônios durante cada passo de treinamento obriga a rede a não depender exclusivamente dos valores de saída de nenhum neurônio específico o que torna a rede mais robusta e mais capaz de generalizar seu aprendizado, diminuindo assim o problema de *overfitting* (NG, 2018).

Na Figura 11 é ilustrado graficamente o efeito da eliminação de neurônios pelo método *Dropout* nas conexões entre as camadas da rede durante a fase de treinamento da rede. É importante ressaltar que os neurônios são removidos apenas durante o treinamento da rede, sendo que durante a sua operação final todos os neurônios são mantidos e os pesos obtidos no treinamento são utilizados.

A efetividade do método *Dropout* se deve ao efeito que a remoção de neurônios tem na rede. Esse pode ser compreendido considerando-se que a remoção de combinações diferentes de neurônios da rede equivale a criação de sub-redes diferentes. Nesse experimento mental, considera-se uma sub-rede como uma versão reduzida da rede principal na qual alguns de seus neurônios foram removidos, mas todo o restante permanece igual, incluindo os seus pesos. Durante o treinamento, cada uma dessas sub-redes é treinada por apenas um

Figura 11 – À esquerda a rede neural antes da aplicação do Dropout e à direita a rede após aplicação do método *Dropout* com probabilidade de eliminar cada neurônio de 0,5



passo de treinamento utilizando os pesos obtidos pelo treinamento das sub-redes anteriores. Esse treinamento de diversas sub-redes diferentes faz com que a rede final apresente pesos com discrepâncias de valor menores entre si, ou seja, a diferença de amplitude entre os maiores e os menores pesos de cada camada é menor do que o obtido sem o uso de *Dropout*. Essa menor discrepância de pesos faz com que a rede neural não dependa excessivamente de um único caminho ao longo da rede para sua tomada de decisão, reduzindo a tendência dela em se adaptar excessivamente à peculiaridades indesejadas dos dados de treinamento.

Nesse trabalho é utilizado o método de *Dropout* Invertido no qual são definidas as probabilidades de se manter cada neurônio ao invés das probabilidades de removê-lo. Primeiramente define-se para cada camada da rede uma probabilidade de manutenção de seus neurônios. Antes de cada passo de treinamento um número aleatório entre 0 e 1 é gerado para cada neurônio de cada camada e, caso esse número seja menor do que a probabilidade de manutenção da camada, o neurônio é removido.

Após a remoção dos neurônios, a sub-rede resultante passa por um passo de treinamento exatamente como se ela fosse a rede original, com as etapas de propagação para frente e para trás executadas normalmente.

Como existem menos neurônios ativos em cada camada, o valor médio do vetor de saídas de cada camada é menor, uma vez que para os neurônios inativos podemos considerar uma saída nula. Pode ser provado que essa redução da média das saídas em uma camada tende a ser diretamente proporcional a probabilidade de manutenção dos neurônios naquela camada (NG, 2018). Para recuperar as médias da rede original, deve-se realizar uma divisão das ativações de cada camada pela probabilidade de manutenção dos neurônios da mesma.

3.1.6.1 Normalização de Batch

O método de Normalização de *Batch* trabalha em mini-*batches*. Um mini-*batch* é um vetor que contém um número arbitrário de pares **X** e **Y** de treinamento. Os mini-*batches* utilizados no treinamento não precisam ser todos do mesmo tamanho. O uso dos mesmos é recomendado quando a base de dados de treinamento é muito grande, pois nesses casos a separação do treinamento em múltiplos mini-*batches* é mais rápida do que o treinamento da rede com toda a base de testes de uma única vez. Além disso, o uso de mini-*batches* permite a aplicação de diversos métodos de regularização como por exemplo Normalização de *Batch*.

O método de regularização Normalização de *Batch* consiste em normalizar as médias e variâncias de cada um dos vetores $\mathbf{z}^{[l]}$ para cada mini-*batch* de treinamento. Onde:

$$\mathbf{z}^{[l]} = \mathbf{w}^{[l]T}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} \tag{3.14}$$

onde $\mathbf{z}^{[l]}$ é o vetor de pré-ativações da camada l da rede, $\mathbf{a}^{[l-1]}$ é o vetor de saídas das funções de ativação da camada l-1 da rede e $\mathbf{w}^{[l]}$ e $\mathbf{b}^{[l]}$ são os pesos sendo treinados para a camada l.

No método de Normalização de *Batch*, os valores das entradas dos neurônios são normalizados depois de serem ponderadas pelos pesos daquela camada. Depois de serem normalizadas, elas devem passar pela função de ativação conforme acontece nas redes tradicionais.

Para normalizar esses valores, primeiramente devemos calcular a média e a variância do vetor de pós-ativação, ou seja **z** da camada em questão.

$$\mu = \frac{1}{m}\sum_{i=1}^{m}\mathbf{z}^{[l](i)} \tag{3.15}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{z}^{[l](i)} - \mu) \tag{3.16}$$

onde m é o número de exemplos de treinamento, μé a média dos m valores de **z** da camada l da rede neural e σ^2 é a variância desses mesmos valores. Com μe σ^2 calculados, pode-se calcular os vetores de ativação normalizados $\mathbf{z}_{\mathrm{norm,\,i}}{}^{\mathrm{l}}$, fazendo:

$$\mathbf{z}^{l}_{norm,i} = \frac{\mathbf{z}^{l}_{i} - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{3.17}$$

onde ε é uma constante pequena arbitrária para impedir que ocorra uma divisão por zero quando a variância for nula. Os valores de $\mathbf{z}_{\mathrm{norm,\,i}}{}^{\mathrm{l}}$ calculados apresentam média zero e variância unitária. Porém o objetivo desse método é permitir que a rede possa

aprender os valores de média e variância ótimos para cada camada. Para isso, uma última transformação tem que ser feita.

$$\tilde{\mathbf{z}}_{\mathbf{i}}^{\mathbf{l}} = \gamma \mathbf{z}_{norm,i}^{l} + \beta \tag{3.18}$$

onde $\tilde{\mathbf{z}}_{\mathrm{i}}^{1}$ é o valor normalizado de z para o neurônio i da camada l da rede neural e $\boldsymbol{\gamma}$ e $\boldsymbol{\beta}$ são parâmetros a serem aprendidos pela rede, assim como $\mathbf{w}$ e b em redes tradicionais.

A normalização dos valores de $\mathbf{z}$ reduz a amplitude de variação dos valores calculados, acelerando o aprendizado e ajudando a rede a generalizar seus resultados. Além disso, a normalização permite que os pesos no interior da rede se tornem mais robustos em relação a mudanças nos pesos iniciais da rede. Isso ocorre uma vez que mesmo que os pesos iniciais mudem significativamente ao longo do treinamento, a distribuição das entradas das camadas internas da rede continua com a mesma média e variância fixa. Além disso essa normalização combate o problema de gradientes tendendo a zero na propagação para trás, muito comum em redes com muitas camadas.

Como calcular a média e variância exata dos valores de z antes do treinamento é uma tarefa computacionalmente muito pesada, é preferível utilizar uma média móvel exponencial ponderada (NG, 2018). Essa pode ser calculada de forma iterativa durante o próprio treinamento e é capaz de estimar a média real de forma suficientemente precisa para a aplicação em Normalização de *Batch*. O algoritmo para o cálculo da média móvel é apresentado a seguir:

$$\mu_t = \alpha \mu_{t-1} + (1 - \alpha) f_t \tag{3.19}$$

onde μ_{t} é a estimativa atual da média, $\mu_{\mathrm{t\text{-}1}}$ é a estimativa da média no instante imediatamente anterior, e $\mathrm{f_t}$ é o valor atual da função da qual estamos calculando a média. Além disso, $\boldsymbol{\alpha}$ é um hiperparâmetro arbitrário que controla o número de termos da função f para os quais a média está sendo estimada de acordo com a relação a seguir:

$$T \approx \frac{1}{1 - \alpha} \tag{3.20}$$

Dessa forma, ao decidir o valor de $\boldsymbol{\alpha}$ pode-se definir o intervalo da função considerado pela média móvel.

3.2 Métodos de visão computacional

Tradicionalmente, na área de visão computacional para a estimação da profundidade, são utilizados métodos "artesanais", nos quais algoritmos matemáticos traduzem as propriedades e características presentes na imagem. Como mostrado por (SCHARSTEIN; SZELISKI,

2002), esse processo é bastante complexo e, por essa razão, a introdução de redes neurais capazes de aprenderem relações e padrões para auxiliar na estimação da profundidade tem sido bem-sucedida.

Nos itens a seguir são apresentadas duas classes de métodos diferentes que fazem uso de redes neurais tipo *deep learning* para gerar estimativas de profundidade, sendo eles a visão estéreo e a visão monocular.

3.2.1 Visão estéreo

Uma das formas mais estudadas de se extrair informações de profundidade de uma imagem é a visão estéreo (CHENG; HE; ZHANG, 2018). Seu sucesso e potencial podem ser observados pela presença de dois olhos em inúmeros animais, incluindo o homem. Para compreender o funcionamento dos métodos tradicionais de visão estéreo apresenta-se o seguinte problema: tem-se duas imagens provenientes de duas câmeras com eixos óticos paralelos em posições horizontais distintas, e cujas posições relativas uma à outra são conhecidas. Utilizando essas imagens deseja-se obter a distância entre um objeto presente nas imagens e as câmeras. A essa distância dá-se o nome de profundidade.

A forma mais tradicional usada para estimar a profundidade de um objeto usando pares de imagens estéreo é a partir do cálculo da disparidade. A disparidade é a diferença na localização horizontal de um mesmo objeto entre as duas imagens (ZBONTAR; LECUN, 2015). Ao se obter a disparidade, a profundidade do objeto pode ser calculada diretamente de forma simples (SCHARSTEIN; SZELISKI, 2002).

Redes neurais convolucionais para calcular a disparidade começaram a ser utilizadas em visão estéreo principalmente no cálculo do custo de correspondência (ZBONTAR; LECUN, 2015), que é uma figura de mérito representando quão boa é a paridade entre dois pixels das imagens direita e esquerda. Ou seja, o custo de correspondência representa a "probabilidade" de que o pixel selecionado na imagem direita represente o mesmo objeto, ou a mesma parte de um objeto, que o pixel selecionado na imagem esquerda.

Diferentes autores propuseram variadas arquiteturas de rede para resolver o problema de determinação da disparidade. Uma breve apresentação de algumas delas é feita a seguir.

No artigo de (ZBONTAR; LECUN, 2015), os pesquisadores propõem a utilização de uma rede neural convolucional siamesa na etapa do cálculo do custo de correspondência de um algoritmo de visão estéreo. A função dessa rede neural é aprender a identificar se uma janela de pixels selecionados em uma das imagens e outra da segunda imagem são ou não similares, ou seja, se elas representam ou não uma mesma região da cena retratada. Para esse fim, a rede neural foi treinada usando um sistema de classificação binária com exemplos de pares de grupos de pixels similares e não similares.

A rede proposta pelos autores é siamesa e simétrica, ou seja, ela é composta por duas sub-redes idênticas que começam separadas e têm suas saídas unidas no meio da rede. A entrada das sub-redes é um par de pequenas janelas, cada uma de uma das imagens estéreo. Ambas as sub-redes são compostas por camadas convolucionais seguidas por *ReLus* em todas as camadas, com exceção da última. A saída é um vetor que representa as propriedades de cada janela analisada. Esses vetores são comparados na parte final unificada da rede neural para definir a medida de similaridade entre as janelas.

A saída da rede neural é usada para calcular a disparidade que é então usada para calcular a profundidade. Os próprios autores argumentam que o sucesso do seu método, que obteve ótimos resultados nos bancos de dados KITTI(GEIGER et al., 2013) e *Middlebury* (SCHARSTEIN et al., 2014), se deve ao uso de uma rede neural convolucional siamesa no cálculo do custo de correspondência e não às outras etapas do seu método, uma vez que as outras não são inovadoras (ZBONTAR; LECUN, 2015).

Outro trabalho interessante foi o de (LUO; SCHWING; URTASUN, 2016), no qual os autores propõe a criação de uma rede neural para o cálculo do custo de correspondência, com foco em garantir um baixo tempo de processamento, de forma a permitir que aplicações em tempo real sejam capazes de fazer uso do seu algoritmo. A arquitetura de rede proposta é muito semelhante à da rede usada por (ZBONTAR; LECUN, 2015), ou seja, uma rede neural convolucional siamesa simétrica na qual cada sub-rede apresenta várias camadas de convolução seguidas por funções de ativação *ReLu*, com exceção da última camada. A entrada da rede, porém, é diferente, sendo que cada sub-rede recebe como entrada uma das duas imagens inteiras e não apenas pequenas janelas das imagens.

Para treinar sua rede neural, os pesquisadores (LUO; SCHWING; URTASUN, 2016) definiram uma função de erro a ser minimizada de forma a tratar o problema como um problema de classificação múltipla, onde as classes são todas as possíveis disparidades para o pixel em questão. Esse tratamento é diferente da classificação binária utilizada por (ZBONTAR; LECUN, 2015), onde a rede tenta determinar se o par de janelas em análise é similar (corresponde a mesma região) ou não. De acordo com (LUO; SCHWING; URTASUN, 2016) essa mudança resulta numa performance superior ao parear os pixels de uma imagem com os seus equivalentes na outra.

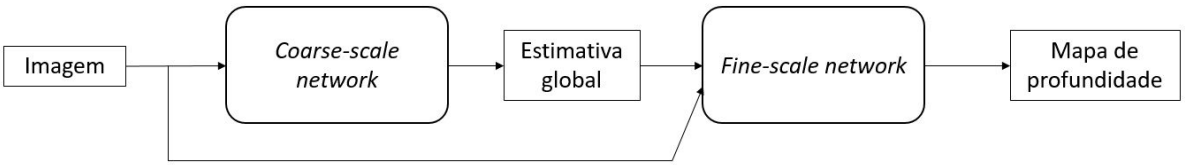
Tanto nos trabalhos de (ZBONTAR; LECUN, 2015) como no de (LUO; SCHWING; URTASUN, 2016), os autores fizeram uso de métodos de geração de dados de treinamento (*Data Augmentation*) para melhorar os resultados de suas redes neurais. Além disso, (LUO; SCHWING; URTASUN, 2016) também fizeram uso de métodos de regularização como Normalização de *Batch*.

3.2.2 Visão monocular

Além do campo da visão estéreo, existe também muito interesse na área de extração de informações de profundidade a partir de uma única imagem, a chamada visão monocular. Nesse caso não é possível utilizar informações de disparidade, sendo necessários outros métodos para se obter a profundidade diretamente da imagem. Além disso, existe uma grande ambiguidade quanto à escala quando trabalha-se com apenas uma imagem, uma vez que uma mesma imagem pode representar inúmeras combinações de cena, ou de escalas diferentes numa mesma cena. Para combater essa ambiguidade, alguns pesquisadores como (EIGEN; PUHRSCH; FERGUS, 2014) utilizam métricas de cálculo de erro que independem da escala. Essas métricas consideram as relações espaciais dentro da cena ao invés de considerar apenas a sua escala absoluta.

Assim como na área de visão estéreo, redes neurais convolucionais são muito utilizadas para a estimativa de profundidade em visão monocular. (EIGEN; PUHRSCH; FERGUS, 2014) propõem uma arquitetura de rede neural para estimativa de profundidade a partir de uma única imagem. Nesse trabalho, os pesquisadores estimam um mapa de profundidade de uma imagem utilizando uma rede neural com dois componentes. O primeiro gera uma estimativa global do mapa de profundidade da imagem enquanto o segundo a refina utilizando informações locais. Para o treinamento dessa rede neural, os autores definiram uma função de erro que explicitamente utiliza relações de profundidade entre os pixels além do erro pontual absoluto. Um esquema ilustrando a ideia de (EIGEN; PUHRSCH; FERGUS, 2014) é mostrado na Figura 12.

Figura 12 – A ideia de (EIGEN; PUHRSCH; FERGUS, 2014)



As primeiras cinco camadas da primeira componente, chamada de *coarse-scale network* pelos autores, fazem uso de operações de convolução seguidas de operações de *max pooling* para combinar informações de diferentes partes da imagem enquanto diminuem a dimensão dos dados que serão as entradas das camadas seguintes. Já as últimas duas camadas são totalmente conectadas, garantindo que cada neurônio dessas camadas tem como entrada informações provenientes de toda a imagem. Todas as camadas ocultas fazem uso da função *ReLu* como função de ativação, com exceção da última camada da *coarse-scale network* que é linear.

Já a rede do segundo componente, chamada de *fine-scale network* pelos autores, apresenta apenas uma camada de *pooling*, a primeira, e todas as suas outras camadas são

convolucionais. Dessa forma, cada neurônio recebe informações apenas de um conjunto restrito de pixels, limitado pelo tamanho do filtro da convolução. Essa rede tem como entrada a imagem analisada e, na sua segunda camada, recebe também a saída da *coarse-scale network*, fazendo uso assim da estimativa global de profundidade calculada anteriormente. Dessa forma a *fine-scale network* pode melhorar a previsão inicial feita pela *coarse-scale network* utilizando detalhes locais da imagem (EIGEN; PUHRSCH; FERGUS, 2014).

Outra ideia sobre visão monocular é a de (KUMAR; BHANDARKAR; PRASAD, 2018). Eles propõem a criação de uma rede neural convolucional e recorrente capaz de estimar profundidade de quadros de vídeos monoculares. A grande inovação da sua arquitetura foi a criação de uma rede capaz de estimar de forma implícita a profundidade de objetos em quadros de um vídeo usando não apenas informações presentes no quadro atual, mas também informações presentes em todos os quadros anteriores.

Sua arquitetura pode ser dividida em duas partes. A primeira é a de codificação e a segunda a de decodificação. A parte de codificação da rede consiste em 8 camadas LSTM convolucionais, a primeira com filtros 7x7 e 32 canais, seguidas por três pares dessas camadas com respectivamente filtros 5x5, 3x3 e 3x3 e 64, 128 e 256 canais. Por fim, a última camada apresenta filtro 3x3 e 512 canais. Seguindo a parte de codificação da rede está a parte de decodificação. Ela é composta por cinco camadas de convolução transposta, pareadas com com cinco camadas convolucionais. Esses cinco pares apresentam filtros 3x3 e, respectivamente, 512, 256, 128, 64 e 32 canais.

Camadas de convolução transposta realizam uma operação de convolução com *padding* especial de forma a gerar uma imagem de saída com resolução espacial inversa da esperada numa convolução normal. Por exemplo, em uma convolução normal sobre uma imagem 5x5, usando um filtro 3x3, a saída será uma imagem 2x2, enquanto a convolução transposta de uma imagem 2x2, usando filtro 3x3, produz uma imagem 5x5. Por essa razão, a convolução transposta também é chamada de deconvolução. É importante notar que, apesar das dimensões da saída da convolução transposta serem iguais às dimensões da entrada de uma convolução que gerou a entrada da convolução transposta, os valores são diferentes. A convolução transposta não reverte uma operação de convolução ela apenas reverte a mudança espacial de dimensão de uma convolução.

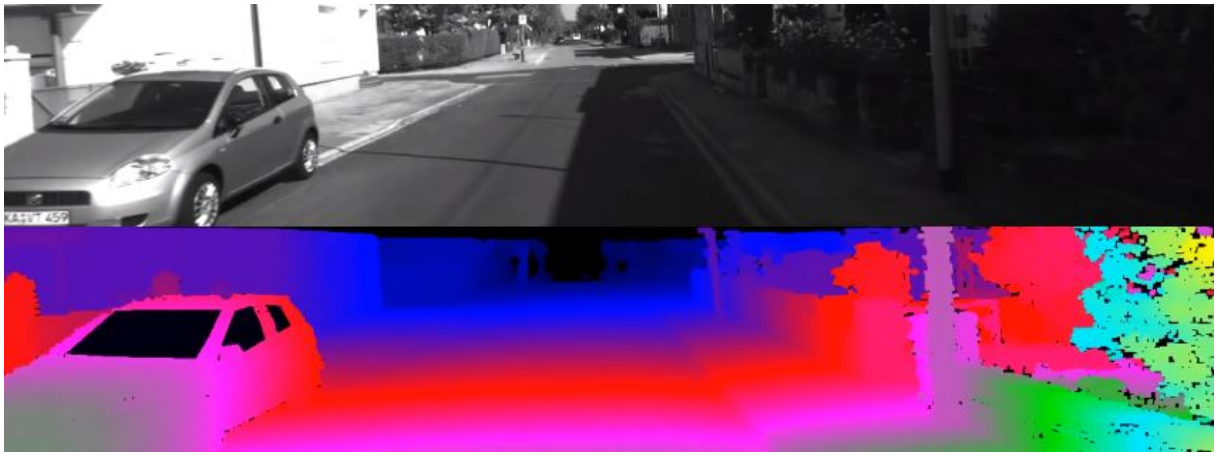
Devido ao uso de camadas LSTM convolucionais, a rede é capaz de aprender relações espaço-temporais entre os quadros e suas profundidades, o que permite que a mesma utilize mais informações ao estimar a profundidade do que redes monoculares tradicionais.

3.3 A base de dados KITTI

Para a execução das ideias mostradas na grande maioria dos trabalhos analisados neste capítulo, utilizou-se a base de dados KITTI (GEIGER et al., 2013). Ela é produto do KITTI *Vision Benchmark Suite*, um projeto conjunto do *Karlsruhe Institute of Technology* na Alemanha e o *Toyota Technological Institute*, nos Estados Unidos. Essa base de dados foi criada para ser usada em aplicações de visão computacional, principalmente visão estéreo, fluxo ótico, odometria visual e detecção 3D de objetos. Para a criação dessa base de dados foi utilizada a plataforma *Annieway*, desenvolvida para pesquisas com carros autônomos. Os sensores dessa plataforma móvel foram utilizados para gerar a base de dados gravando a visão frontal do veículo e sincronizando essa gravação com informações de profundidade coletadas em tempo real, enquanto o carro trafegava por ruas e estradas na cidade de Karlsruhe na Alemanha.

Essa base de dados disponibiliza, em seu site, cerca de 150 vídeos com resolução de 1242 x 375 pixels, 10 quadros por segundo e duração variando de poucos segundo até 8 minutos. De forma a facilitar o uso dos quadros dos vídeos, é possível baixá-los na forma de uma sequência ordenada de imagens no formato png, sendo todas elas associadas com informações de profundidade. No total essa base de dados apresenta ao redor de 43 mil pares de imagens vinculadas com informações precisas de profundidade. Na Figura 13 é ilustrado um exemplo de imagem e de mapa de profundidade disponível na biblioteca KITTI (GEIGER et al., 2013).

Figura 13 – Exemplo de imagem do banco de imagens KITTI



Fonte: (GEIGER et al., 2013)

A base de dados KITTI (GEIGER et al., 2013) foi escolhida para o treinamento e validação das redes neurais desenvolvidas neste trabalho devido à sua qualidade, o seu tamanho e sua excelente documentação, além da sua predominância em várias publicações recentes na área de estimativa de profundidade de imagens. Outra vantagem que essa base de dados apresenta, é a sua relação muito próxima ao problema de visão computacional

em carros autônomos, uma vez que suas imagens são exatamente do tipo com o qual um sistema de estimação de profundidade de um carro autônomo teria que trabalhar. Uma rede treinada usando essa base de dados estaria naturalmente adaptada a esse problema.

Os mapas de profundidade fornecidos pela base de dados KITTI (GEIGER et al., 2013) consistem em imagens no formato png *uint16* com a mesma resolução que os quadros fornecidos. Nessas imagens png, o valor de 0 em um pixel representa um pixel inválido, ou seja um para o qual não existe um valor de profundidade real conhecido. Os demais pixels da imagem codificam informações de profundidade nos seus valores, que podem ser transformados em metros convertendo-se os valores *uint16* para *float* e depois realizando uma divisão por 256.

Conforme explicado no capítulo 2 Análise de Requisitos, a base de dados KITTI compara o desempenho dos métodos de estimativa de profundidade usando quatro métricas de erro distintas, o erro relativo quadrado, o erro relativo absoluto, a *iRSME* (*Root mean squared error of the inverse depth*) e o erro logarítmico de escala invariante. Essas quatro métricas de erro distintas são explicadas a seguir.

O erro relativo quadrado $E_{rel\ sq}$ calcula a média das razões dos quadrados das diferenças entre as previsões do sistema e os valores reais com os valores reais em cada ponto. A sua fórmula é mostrada a seguir.

$$E_{relsq} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{y_i}\right)^2 \tag{3.21}$$

onde para cada um dos N pixels de cada quadro $\hat{y}_\mathrm{i}$ são as previsões de profundidade calculadas e y_i são os valores reais de profundidade dos pixels da imagem.

O erro relativo absoluto $E_{rel\ abs}$ calcula a média das razões dos módulos das diferenças entre as previsões do sistema e os valores reais com os valores reais em cada ponto. A sua fórmula é mostrada a seguir.

$$E_{relabs} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{3.22}$$

onde n, y_i e $\hat{y}_\mathrm{i}$ representam os mesmos valores que na fórmula do erro relativo quadrado.

A raiz do erro médio quadrado do inverso das profundidades *iRSME* é calculada fazendo-se a raiz quadrada da somatória dos quadrados da diferença entre o inverso da profundidade estimada e o inverso da profundidade real de cada pixel da imagem dividida

pelo número de pixels da imagem. A fórmula para o seu cálculo é apresentada a seguir.

$$iRSME = \sqrt{\frac{\sum_{i=1}^{N}(\frac{1}{y_i} - \frac{1}{\hat{y}_i})^2}{N}} \tag{3.23}$$

onde N, y_i e $\hat{y}_\text{i}$ representam os mesmos valores que na fórmula do erro relativo quadrado.

O erro logarítmico de escala invariante E_{lei} foi proposto inicialmente por (EIGEN; PUHRSCH; FERGUS, 2014) e posteriormente adotado como um dos critérios de avaliação da base de dados KITTI(GEIGER et al., 2013). Essa métrica visa medir a variação de profundidade relativa entre dois pontos na cena sem considerar a escala de profundidade absoluta da mesma.

Para um mapa de profundidade previsto $\hat{\boldsymbol{y}}$ e um mapa de profundidade real $\boldsymbol{y}$, cada um com N pixels podemos definir o erro logarítmico de escala invariante como:

$$E_{lei} = \frac{1}{2N}\sum_{i=1}^{N}(\log \hat{y}_i - \log y_i + \alpha(y_i, \hat{y}_i))^2 \tag{3.24}$$

onde

$$\alpha(y_i, \hat{y}_i) = \frac{1}{N}\sum_{i=1}^{N}(\log y_i - \log \hat{y}_i) \tag{3.25}$$

Para cada previsão y temos que e^{α} é a escala que melhor alinha a previsão às profundidades reais da cena. Qualquer múltiplo escalar de y, ou seja qualquer mudança na escala de y, apresenta o mesmo erro nessa métrica, por essa razão ela é chamada de erro logarítmico de escala invariante.

Essa métrica também pode ser definida da seguinte forma análoga:

$$E_{lei} = \frac{1}{N}\sum_{i=1}^{N}(\log y_i - \log \hat{y}_i)^2 - \frac{1}{N^2}(\sum_{i=1}^{N}(\log y_i - \log \hat{y}_i))^2 \tag{3.26}$$

Nessa forma, a métrica se torna uma modificação do erro logarítmico tradicional, primeiro termo, com um termo adicional, o segundo termo. Esse termo faz com que previsões imperfeitas apresentem um erro menor caso seus resultados incorretos sejam consistentes uns com os outros, ou seja, quando a causa dos seus erros é um problema na estimação da escala global da cena e não da profundidade relativa entre objetos.

Como explicado anteriormente, na base de dados KITTI (GEIGER et al., 2013) nem todos os pixels das imagens estão associados a um valor medido de profundidade real.

Desse modo, todos os erros são calculados usando apenas os pixels válidos, ou seja, aqueles com valores reais de profundidade conhecidos.

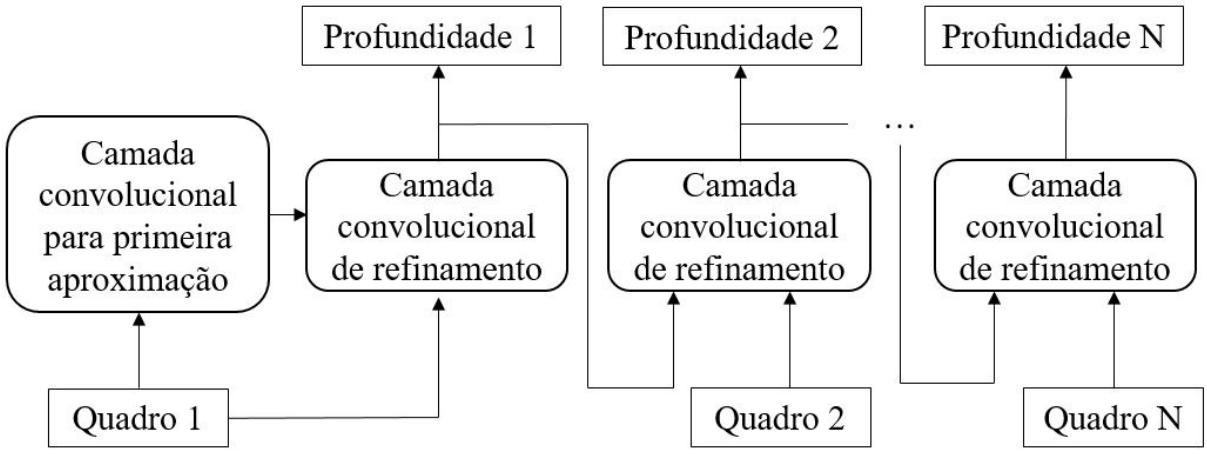
4 CONCEPÇÃO DA REDE NEURAL

O trabalho de (EIGEN; PUHRSCH; FERGUS, 2014) teve uma influência significativa na ideia desenvolvida neste projeto. As redes neurais idealizadas para a estimação da profundidade de objetos em vídeos baseiam-se fortemente na pesquisa desses autores.

A suposição básica deste projeto consiste no fato de que as profundidades dos pixels de um quadro de um vídeo são uma primeira aproximação bastante razoável dessa mesma medida para o quadro seguinte. Para adicionar essa capacidade de memória na rede neural foram testadas camadas recursivas tradicionais, camadas do tipo LSTM e LSTM convolucionais.

O ponto forte dessa ideia é o de que as informações de profundidade obtidas em um tempo anterior são reutilizadas para cálculos futuros. A expectativa é, portanto, que as estimativas se tornem cada vez melhores e mais precisas até que se estabilizem. Como dito anteriormente, um dos objetivos deste trabalho é demonstrar a efetividade desse princípio, na Figura 14 esse princípio é ilustrado.

Figura 14 – Idealização inicial da rede desenvolvida

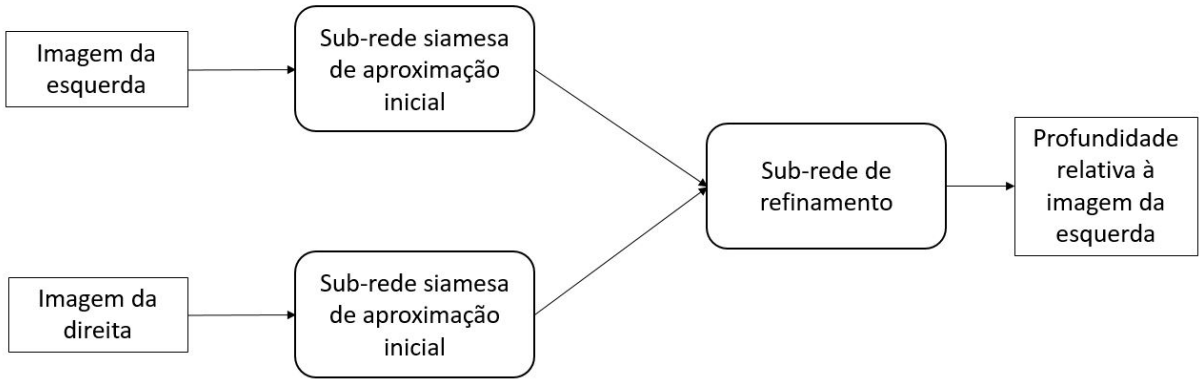


Como explicado anteriormente, (EIGEN; PUHRSCH; FERGUS, 2014) propuseram uma rede convolucional para a estimação de profundidade usando apenas uma imagem. Para isso, os autores dividiram a estrutura da rede em duas partes, sendo que a primeira, a sub-rede de aproximação inicial (*coarse-scale network*), realiza uma primeira aproximação e a segunda, a sub-rede de refinamento (*fine-scale network*), usando a imagem original, refina os resultados obtidos na parte anterior e melhora as estimativas.

A ideia básica proposta por (EIGEN; PUHRSCH; FERGUS, 2014) foi adaptada para o caso de visão estéreo, no qual são utilizadas duas sub-redes siamesas simétricas de

aproximação inicial (uma para a imagem provinda da câmera da esquerda e outra para a da direita). Além disso, ao agregar-se também a ideia de dependência temporal, obtém-se uma rede siamesa, recursiva, convolucional e de estimação direta da profundidade. Desse modo, foi definida a arquitetura básica deste trabalho mostrada na Figura 15.

Figura 15 – Arquitetura de rede siamesa, recursiva e convolucional proposta nesse trabalho



A proposta de arquitetura ilustrada na Figura 15 possui diversas vantagens, sendo que a principal é a modularidade. Ou seja, é possível que diferentes sub-redes de aproximação ou de refinamento sejam concebidas sem que o conjunto seja afetado. Neste capítulo isso é demonstrado, sendo que três variações de redes convolucionais são exploradas para as redes de aproximação e outras três variações de redes recursivas são exploradas no refinamento.

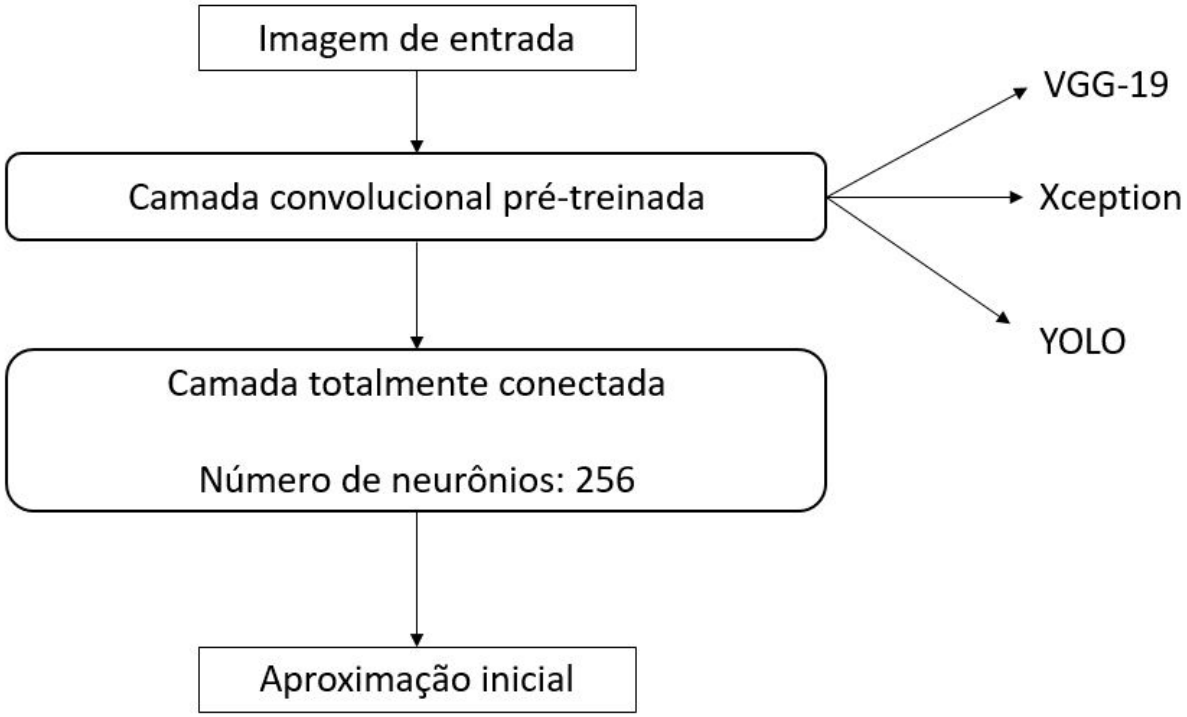
4.1 A sub-rede de aproximação inicial

Como a base de dados KITTI (GEIGER et al., 2013) fornece imagens estéreo, decidiu-se fazer uso pleno dessa camada de informação adicional. A sub-rede de aproximação inicial é, portanto, uma rede convolucional siamesa simétrica, onde cada imagem, esquerda e direita, passa por um dos lados da mesma. Essa aplicação de redes siamesas para processar as imagens estéreo foi inspirada por (ZBONTAR; LECUN, 2015), (LUO; SCHWING; URTASUN, 2016) e (CHEN et al., 2015), mesmo que, no caso deste trabalho, nenhum cálculo de disparidade seja feito.

Como dito anteriormente, cada lado da sub-rede de aproximação inicial é principalmente convolutivo, sendo que preferiu-se utilizar redes consagradas e pré-treinadas, tais como as redes YOLO (REDMON, 2017), VGG19 (SIMONYAN; ZISSERMAN, 2014) e *Xception* (CHOLLET, 2016), no lugar de uma arquitetura completamente nova. Essa decisão foi tomada uma vez que o objetivo desse trabalho é testar a eficácia do uso de relações temporais na estimação de profundidade com redes neurais e não o desenvolvimento de novas arquiteturas convolucionais.

Na Figura 16 é ilustrada a arquitetura básica da sub-rede de aproximação inicial e são mostradas as suas variações de redes convolucionais.

Figura 16 – Arquitetura da sub-rede de aproximação inicial



Como pode ser observado, a parte convolucional de cada um dos lados da rede siamesa é formada pelas redes convolucionais citadas anteriormente. Posteriormente, as saídas da rede convolucional passam por uma camada totalmente conectada com 256 neurônios antes de serem enviadas para a sub-rede de refinamento. É importante mencionar que essa última camada consiste em um gargalo da rede, dado que ela diminui muito o tamanho de cada imagem (para 16x16x1). Durante a concepção da sub-rede de aproximação inicial, percebeu-se a necessidade de usar uma camada totalmente conectada no final, visto que ela melhorava as estimativas consideravelmente. Porém, o tamanho teve que ser propositalmente diminuído, dada a limitação de capacidade computacional deste projeto. Os capítulos 5 e 6 fornecem maiores informações sobre o mencionado empecilho.

A seguir são explicados os motivos de escolhas das três redes convolucionais testadas neste projeto, assim como uma explicação sobre as mesmas.

4.1.1 Redes pré-treinadas

Uma das grandes facilidades de se utilizar redes convolucionais pré-treinadas é que elas apresentam pesos já definidos, ou seja, os pesos são inicializados com valores obtidos por um treinamento já concluído. Esse processo visa reduzir o tempo de treinamento necessário

para redes que utilizam essas camadas, uma vez que, para aplicações semelhantes à original, os pesos pré-treinados consistem de uma boa aproximação para os pesos na nova rede e os tempos de convergência usando esses valores iniciais devem ser mais curtos.

O pré-treinamento dessas redes foi realizado usando a base de dados *Imagenet* (DENG et al., 2009), que é muito utilizada na área de visão computacional, principalmente na área de identificação de objetos. A *Imagenet* contém cerca de 14 milhões de imagens relacionadas a diversos conceitos chamados de *synsets* (*synonym set*). Cada conceito apresenta em média 500 imagens relacionadas e cada imagem pode estar relacionada a mais de um conceito. A imagem de um crocodilo em um rio, por exemplo, pode estar relacionada com os conceitos animal, crocodilo, rio e água, esses conceitos são classificados de acordo com o seu conteúdo em categorias variadas como, por exemplo: ferramentas, animais, formações geológicas e pessoas. Cada uma dessas categorias amplas apresenta várias camadas de sub-categorias. O *synset* robô, por exemplo, se encontra no seguinte ramo da árvore de conceitos: artefato > instrumentação > dispositivo > mecanismo > robô.

Apesar das redes treinadas usando a base de dados *Imagenet* serem treinadas principalmente para tarefas de detecção e classificação de objetos em imagens, existe grande similaridade com relação a tarefa de estimação de profundidade abordada neste trabalho, visto que saber quais pixels compõem um dado objeto auxilia na determinação de profundidade. Por essa razão, o uso desses pesos acelera o processo de treinamento posterior das redes desenvolvidas nesse trabalho.

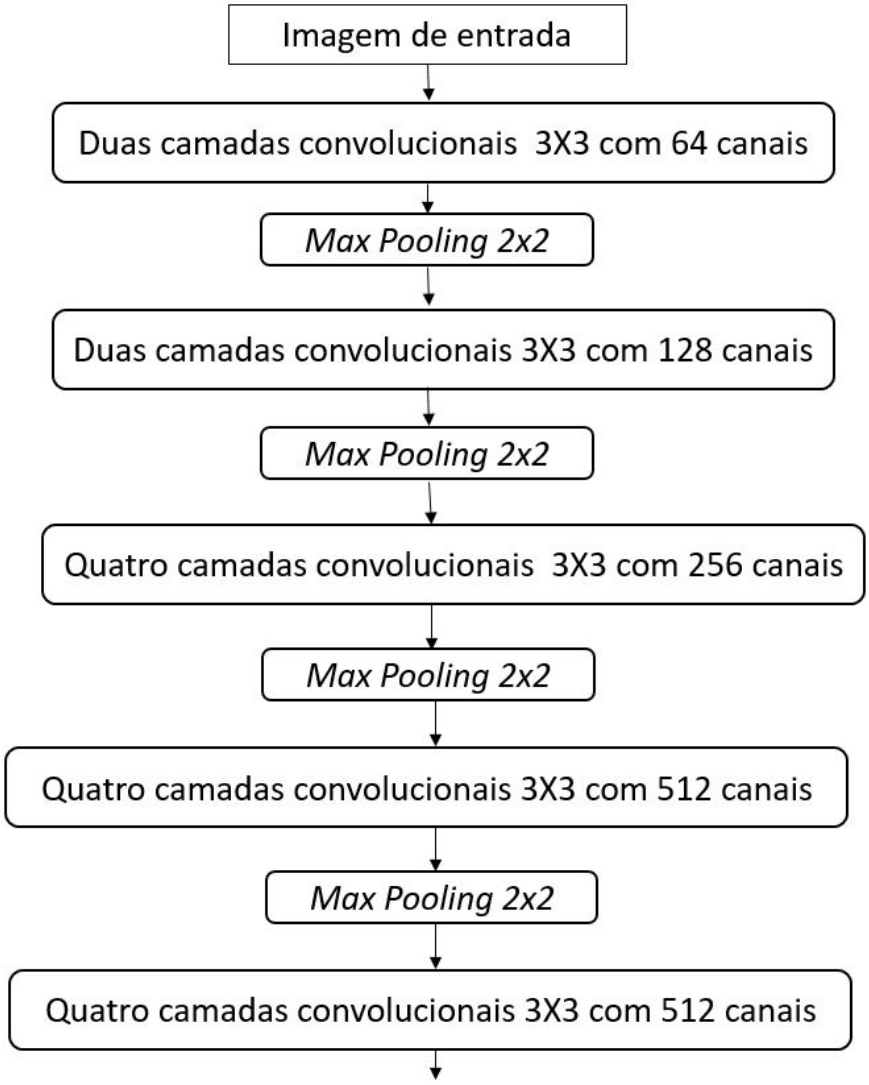
Nas redes desenvolvidas nesse trabalho, foram importadas apenas as camadas convolucionais pré-treinadas na base de dados *Imagenet* e as camadas *max pooling*, que não apresentam parâmetros de aprendizado.

4.1.2 Rede VGG19

A primeira rede convolucional pré-treinada testada é conhecida como VGG19. Essa rede é muito profunda, contendo 19 camadas com parâmetros de aprendizado. Delas 16 camadas são convolucionais e 3 totalmente conectadas. Além disso, são ainda utilizadas 5 camadas de max pooling e uma camada *softmax*. Dessas camadas, apenas as camadas convolucionais e max pooling são importadas e utilizadas nas redes propostas neste trabalho. Na Figura 17 é ilustrada a arquitetura da rede VGG19.

Como mostrado na Figura 17, a imagem de entrada passa por 5 conjuntos de camadas convolucionais, todas com filtro 3x3 e passo unitário. Os primeiros dois conjuntos contém 2 camadas convolucionais cada um e os demais apresentam quatro camadas convolucionais cada. O primeiro conjunto apresenta 64 canais e, após cada conjunto, o número de canais é dobrado, com exceção do último conjunto no qual o número de canais

Figura 17 – Arquitetura da rede VGG19 utilizada



não muda . Dessa forma, o número de canais em cada conjunto é de 64, 128, 256, 512, 512. Depois de cada conjunto de camadas convolucionais, se encontra uma camada max pooling com filtro 2x2 e passo dois e, ao passar por essas camadas, o tamanho da imagem é reduzido pela metade devido ao passo. No seu formato original, a rede faz ainda com que a saída da última camada convolucional passe por três camadas totalmente conectadas e por uma camada *softmax*. Essas últimas etapas foram retiradas para a sub-rede de aproximação inicial.

Conforme pode ser percebido, essa rede é muito grande, possuindo cerca de 20 milhões de parâmetros apenas nas camadas convolucionais.

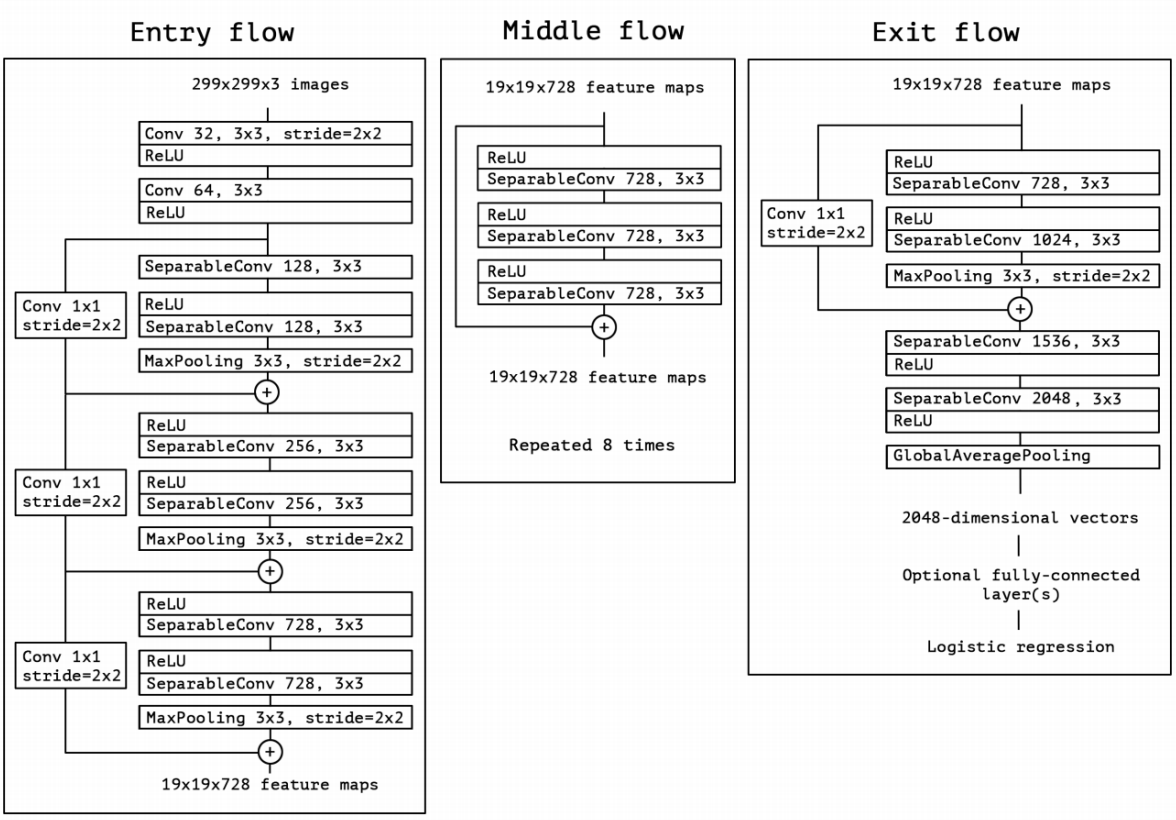
4.1.3 Rede Xception

Outra rede convolucional utilizada neste trabalho é a *Xception*. Ela é uma versão mais "extrema" da rede *Inception*, sendo essa a origem do seu nome. Por essa razão, é necessário

entender o funcionamento da primeira rede para compreender a segunda.

A rede *Inception* foi desenvolvida de forma a permitir que múltiplos filtros convolutivos de tamanho diferente possam operar numa mesma profundidade, como se pertencessem a uma única camada. O módulo *Inception* tradicional realiza três operações de convolução diferentes, com filtros 1x1, 3x3 e 5x5, e uma operação *max pooling* numa mesma entrada da rede. As saídas dessas operações são concatenadas e enviadas para o próximo módulo *Inception*. Para reduzir o trabalho computacional realizado nas convoluções 3x3 e 5x5, é comum ser colocada uma convolução 1x1 antes de cada uma dessas convoluções, de forma a reduzir significamente o número de operações realizadas nessas convoluções sem alterar expressivamente seu resultado.

Figura 18 – Arquitetura da rede *Xception* utilizada



Fonte: (CHOLLET, 2016)

Na rede *Xception*, o bloco básico da rede *Inception* é modificado, de forma a separar completamentamente o mapeamento de informações entre os canais da entrada, feito por operações de convolução 1x1, da posterior correlação espacial, realizada por operações de convolução separadas em cada um dos canais. A rede é composta por camadas de convolução separadas em profundidade. Essa operação de convolução separada consiste em dividir uma convolução tradicional em duas operações distintas. A primeira aplica um filtro para cada canal da imagem e a segunda operação aplica uma outro filtro 1x1 entre todos os canais da imagem de saída da convolução anterior. Na Figura 18, retirada de

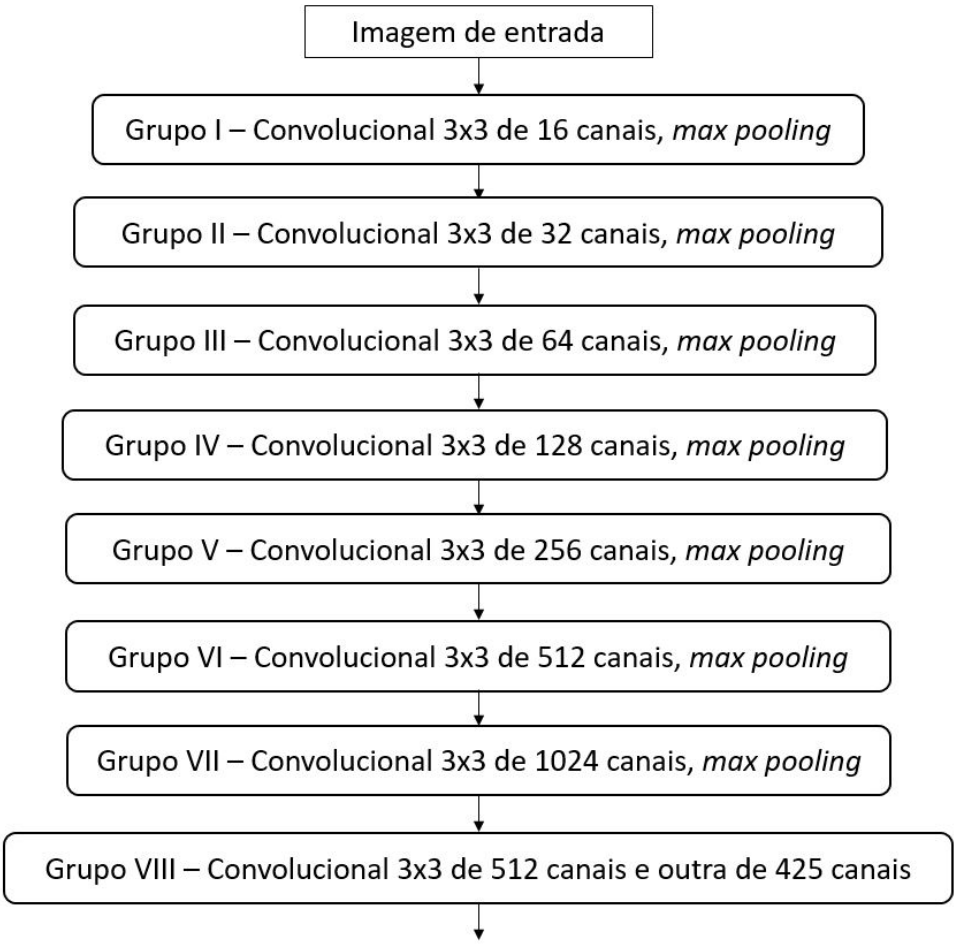
(CHOLLET, 2016), pode ser observada a arquitetura dessa rede.

O uso de convoluções separadas reduz o número de parâmetros de aprendizado da rede sem reduzir a sua capacidade. Por essa razão, a rede *Xception* mesmo contendo um número semelhante de parâmetros que outras redes (cerca de 20 milhões), como a variação V3 *Inception*, apresenta um resultado melhor.

4.1.4 Rede YOLO

A terceira e última rede testada para a sub-rede de aproximação inicial foi uma das versões da rede conhecida como YOLO (*You Only Look Once*). A arquitetura desta popular rede é mostrada na Figura 19.

Figura 19 – Arquitetura da rede YOLO utilizada



Como ilustrado, a versão de YOLO utilizada possui 9 camadas convolucionais com filtros 3x3 divididas em 8 grupos. Cada grupo, com exceção do último, é separado por camadas *max pooling*, seis no total, com passo 2 e filtro 2x2. Os 7 primeiros grupos convolutivos apresentam respectivamente 16, 32, 64, 128, 256, 512 e 1024 canais. Por fim, o oitavo grupo possui duas convoluções com 512 e 425 canais respectivamente. Todas as

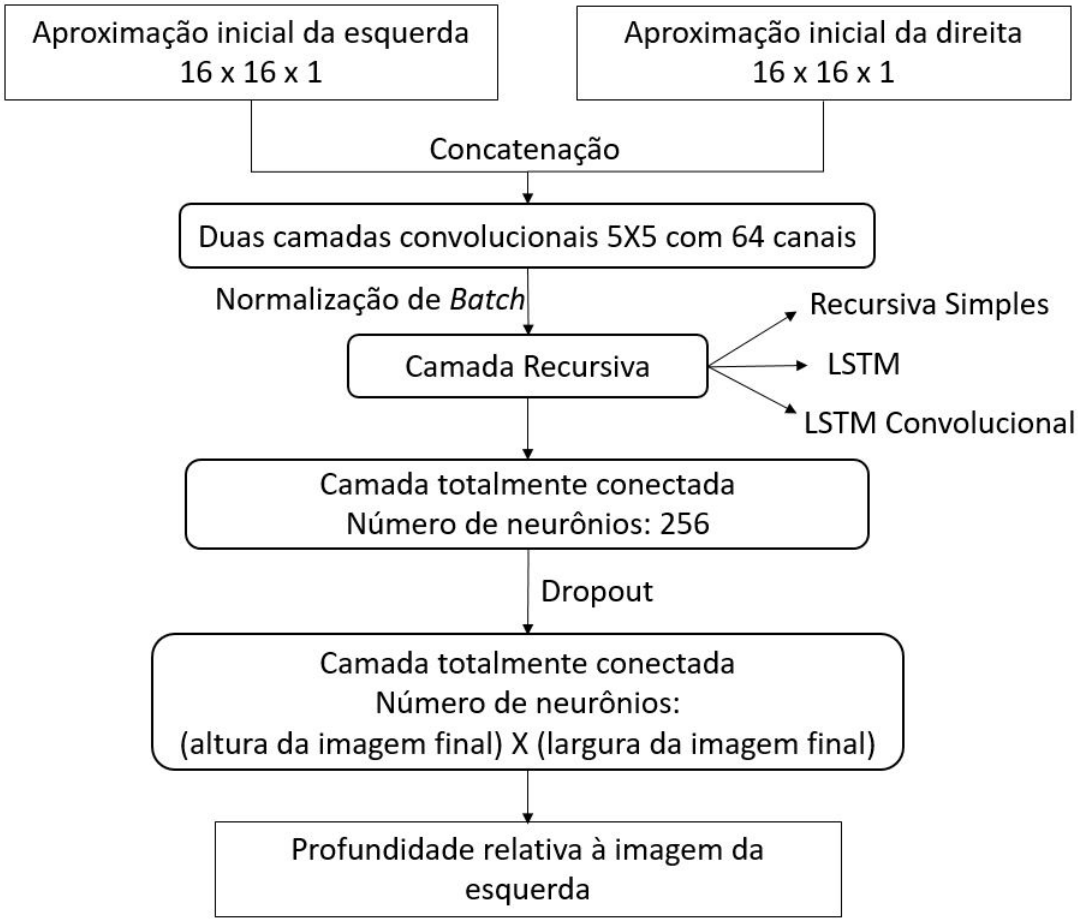
camadas convolucionais, com exceção da última, são seguidas por Normalização de *Batch* e as funções de ativação de todas são do tipo *LeakyRelu*.

Essa rede foi desenvolvida especificamente para a detecção de objetos e definição de delimitação de fronteira para eles. Assim como as outras redes utilizadas, as camadas convolucionais da YOLO foram pré-treinadas na base de dados *Imagenet* e ela contém cerca de 11 milhões de parâmetros.

4.2 A sub-rede de Refinamento

A sub-rede de refinamento contém a parte recursiva da rede, responsável pelas relações temporais da mesma. Ela também é composta por camadas convolucionais e totalmente conectadas. Na Figura 20 é ilustrada a arquitetura básica dessa parte da rede.

Figura 20 – Sub-rede de refinamento



Como pode ser visto na Figura, a primeira camada da sub-rede de refinamento realiza a concatenação das duas saídas paralelas da sub-rede de aproximação inicial produzidas pela parte siamesa da rede. Após a concatenação, seguem duas camadas convolucionais com 64 canais e filtros 5x5. Elas são sucedidas pela camada recursiva que poderá ser de três tipos: Recursiva simples, LSTM ou LSTM convolucional. Por fim, a

imagem passa por duas camadas totalmente conectadas, a primeira com 256 neurônios e a segunda com número de neurônios dependente do tamanho da imagem de saída desejada para a rede.

Tal qual mencionado no parágrafo anterior, foram propostas três diferentes camadas recursivas para a sub-rede de refinamento, sendo que uma delas utiliza uma camada recursiva tradicional, a outra faz uso de uma camada LSTM e a última uma LSTM convolucional.

As arquiteturas com as camadas recursiva simples e LSTM apresentam apenas uma camada recursiva com número de neurônios igual ao número de pixels da imagem final, ou seja, a altura da imagem final multiplicada pela largura da mesma. Já a arquitetura LSTM convolucional apresenta seis camadas com 64 canais e filtros 7x7, 5x5, 3x3, 5x5, 5x5 e 5x5 respectivamente. A escolha dos filtros e do número de canais foi inspirada no trabalho de (KUMAR; BHANDARKAR; PRASAD, 2018).

5 METODOLOGIA

De modo a viabilizar a análise da proposta de arquitetura de rede deste trabalho e de contornar as limitações de poder computacional existentes na execução deste projeto, fez-se necessário o estabelecimento de uma metodologia composta de três principais passos: a concepção das arquiteturas de rede a serem analisadas, o treinamento dessas arquiteturas e a comparação das mesmas de forma a encontrar a mais promissora.

A partir da ideia básica do uso de informações temporais para a melhoria da estimação de profundidade, três variações de arquitetura convolucional e três de arquitetura recursiva foram propostas. Todas elas foram implementadas na linguagem *Python* a partir do uso das bibliotecas *Tensorflow* (ABADI et al., 2015) e *Keras* (CHOLLET, 2015), que facilitam muito a criação dos métodos discutidos no capítulo 3, devido à sua documentação e clareza. As arquiteturas aqui mencionadas são explicadas em detalhe no capítulo 4.

Para o treinamento de todas as redes desenvolvidas neste trabalho foi utilizado o *cluster* lince do HPC da USP. Esse *cluster* apresenta 32 servidores, cada um com 16 núcleos físicos *Intel(R) Xeon(R) E5-2680 rv0 @ 2.70GHz*, com duas placas *Tesla K20m* e 128 GB de memória *RAM*.

5.1 Definição do erro de treinamento

Como primeiro passo importante durante a etapa de definição de parâmetros de treinamento, decide-se qual a função de erro a ser utilizada. (EIGEN; PUHRSCH; FERGUS, 2014) propuseram em seu trabalho que sua métrica de erro logarítmico de escala invariante fosse usada no cálculo da função de erro minimizada durante o treinamento da rede neural. Nessa função, porém, eles propõem a introdução do coeficiente λ que regula a influência do erro relativo no aprendizado.

$$E_{lei} = \frac{1}{N}\sum_{i=1}^{N}(\log y_i - \log \hat{y}_i)^2 - \frac{\lambda}{N^2}(\sum_{i=1}^{N}(\log y_i - \log \hat{y}_i))^2 \tag{5.1}$$

Para λ igual a um, o erro de aprendizado é exatamente o erro logarítmico de escala invariante e para λ igual a zero é o erro logarítmico normal. Os autores concluíram que o valor de λ igual a 0,5 produz boas aproximações em escala absoluta, enquanto melhora um pouco a saída qualitativa. Por essa razão, a dupla adotou essa abordagem para definir o erro de treinamento das redes.

5.2 Treinamento e seleção da sub-rede convolucional

Para a primeira etapa de treinamento da rede, foi decidido que uma seleção entre três alternativas de sub-rede de aproximação inicial seria realizada, mantendo-se fixa a sub-rede de refinamento mais simples, ou seja, aquela que faz uso de uma camada recursiva simples. Dessa forma, foram treinadas as três variações de sub-rede de aproximação inicial, cada uma usando uma das redes convolucionais pré-treinadas VGG19, *Xception* e YOLO.

Para avaliar o desempenho das diferentes arquiteturas convolucionais e escolher a mais promissora, essas redes foram submetidas a um treinamento com as imagens de entrada (ordenadas temporalmente) de tamanho 80x80 e de saída (que contém a informação de profundidade) de 40x40. O tamanho de cada vídeo de treinamento variava de acordo com aquele disponível pela biblioteca KITTI (GEIGER et al., 2013) e o máximo possível de se colocar na memória do HPC. As três arquiteturas da sub-rede de aproximação inicial, cada uma fazendo uso de uma das três arquiteturas convolucionais pré-treinadas, foram treinadas em paralelo em *clusters* distintos do computador disponível. O treinamento das redes em paralelo permite um uso melhor da capacidade computacional disponível, além de reduzir o tempo consumido para treinar as três redes, já que os três treinamentos são realizados simultaneamente.

Inicialmente, é utilizada uma taxa de aprendizado de 0,001. A cada duas épocas de treinamento, nas quais o erro de validação varia menos que 0,001, a mesma é automaticamente dividida por 10. Após dez épocas com o erro de validação variando menos que 0,001, o treinamento é encerrado. O conhecido algoritmo de otimização Adam foi utilizado.

5.3 Treinamento e seleção da sub-rede recursiva

Após a seleção da melhor alternativa de sub-rede de aproximação inicial, as três opções de sub-rede de refinamento (recursiva simples, LSTM totalmente conectada e LSTM convolucional) são treinadas usando a arquitetura de aproximação inicial de melhor desempenho.

Novamente os treinamentos são realizados em paralelo, a mesma taxa de aprendizado é utilizada, assim como o mesmo critério de parada e o mesmo algoritmo de otimização. Porém, para essa segunda etapa de treinamento, considera-se também a duração de cada vídeo, de forma a dar mais importância para os mais longos. A taxa de aprendizado de cada etapa de treinamento é multiplicada por um peso que leva em conta o número de quadros do vídeo. A definição desse novo parâmetro pode ser vista na fórmula a seguir:

$$P = 1 + \frac{N_i}{N_{max}} \tag{5.2}$$

onde P representa o peso de multiplicação, N_i o número de quadros do *batch* de treinamento i e N_{max} o número máximo de quadros dos *batches* de treinamento.

5.4 Observações sobre a capacidade computacional

Uma das grandes limitações deste projeto se deve às restrições de capacidade computacional disponível. Para treinar uma rede de forma rápida, o computador deve ser capaz de carregar simultaneamente na sua memória RAM a rede neural e um *batch* de treinamento inteiros. Para redes com muitos parâmetros ou *batches* de treinamento muito grandes, isso pode exigir muita memória do computador. Devido ao tamanho das redes idealizadas e da base de treinamento utilizada, medidas tiveram que ser tomadas para garantir que o computador disponível fosse capaz de treinar as redes em tempo razoável.

Apesar de o treinamento de redes neurais ser mais rápido quando realizado em GPUs, o treinamento das redes propostas foi realizado em CPUs, uma vez que as GPUs disponíveis não tinham memória o suficiente para carregar as redes. Uma alternativa considerada foi realizar o treinamento da rede usando as várias GPUs para realizar as operações e cálculos, enquanto o modelo da rede e os *batches* de treinamento permaneciam na memória da CPU. Infelizmente essa solução exigiria que as informações a serem processadas fossem constantemente transferidas da CPU para as GPUs e, como essa conexão é mais lenta, ela tornava esse processo muito lento. Além disso, mesmo a memória das CPUs disponíveis era incapaz de carregar uma versão da rede maior do que a utilizada, devido a limitações de memória.

Para o treinamento das arquiteturas de aproximação inicial e de refinamento, foram utilizados, respectivamente, cortes dos quadros da base de dados KITTI (GEIGER et al., 2013) com dimensões de 80x80 e de 240x80 para as imagens de entrada e 40x40 e 120x40 para as de profundidade. Essa redução drástica no tamanho original das imagens de entrada (1242x375) se deve ao problema de limitação de capacidade computacional disponível para o treinamento da rede, visto que o número de parâmetros de aprendizado das camadas totalmente conectadas e recursivas cresce muito rápido com o aumento do tamanho das imagens de saída. Por essa razão, o treinamento de redes que operam com imagens maiores do que os tamanhos utilizados exigem mais memória do que a do computador disponível, que é de 128 GB.

Devido a esse problema de memória, a base de dados foi dividida em *batches* com tamanho variável e baseado no número de quadros dos vídeos disponíveis. Cada vídeo foi convertido em um *batch* de treinamento. O tamanho máximo deles, em quadros, foi de 1200 para o treinamento da sub-rede de aproximação inicial e de 300 para a sub-rede de refinamento. Esses números máximos de quadros por *batch* são o maior número de quadros por *batch* que o computador usado no treinamento era capaz de carregar na sua memória.

Quando possível os tamanhos máximos de *batch* foram usados pois *batches* muito pequenos reduzem a capacidade de memória que as camadas recursivas são capazes de aprender. Para chegar nos valores máximos de *batches* mencionados foi utilizado o método da busca binária.

Os números são diferentes para os dois treinamentos, pois o tamanho das imagens de entrada nos dois treinamentos é diferente. Vídeos com mais quadros do que o limite estipulado são divididos em múltiplos *batches* menores. Como os vídeos utilizados apresentam 10 quadros por segundo, os *batches* de tamanhos máximos de 300 e de 1200 quadros e tamanho mínimo de 12 quadros (menor vídeo disponível), representam vídeos com duração variando entre 1,2 e 120 segundos para o treinamento da sub-rede de aproximação inicial, e entre 1,2 a 30 segundos para o treinamento da sub-rede de refinamento.

5.5 Comparação dos resultados com os da base de dados KITTI e discussão dos mesmos

A combinação da melhor rede convolucional com a melhor recursiva será avaliada no caítulo 6 com relação aos requisitos estabelecidos. Os seus resultados também serão comparados a trabalhos de outros pesquisadores de acordo com o desempenho na tarefa de estimação de profundidade usando a base de dados KITTI (GEIGER et al., 2013).

É importante mencionar que os resultados de outros pesquisadores que serão analisados foram obtidos por redes que fazem uso de apenas uma imagem para gerar as suas estimativas, o que é diferente do proposto neste trabalho. Porém, para o caso de visão estéreo, o banco de imagens não possui uma comparação de estimação direta, mostrando apenas a assertividade no cálculo da disparidade. Desse modo, as estimativas geradas por este projeto se enquadram melhor naqueles resultados apresentados na Tabela 1 e, por isso, os últimos serviram como comparação de desempenho.

6 RESULTADOS

Neste capítulo são apresentados e analisados os resultados de treinamento das arquiteturas de rede propostas, assim como a escolha daquela com melhor desempenho. Os resultados da última são então comparados com os requisitos definidos no capítulo 2.

Primeiramente são apresentados os resultados do treinamento das três arquiteturas convolucionais (VGG19, *Xception* e YOLO) e a mais promissora é então escolhida. Posteriormente são apresentados os resultados referentes às três arquiteturas recursivas e a de melhor desempenho é selecionada. Finalmente são analisados em mais detalhes os resultados da arquitetura final em comparação com os requisitos definidos para esse projeto.

6.1 Análise das sub-redes de aproximação inicial e as opções convolucionais

Conforme explicado anteriormente, as três arquiteturas convolucionais, usando as redes importadas VGG19, *Xception* e YOLO, da sub-rede de aproximação inicial foram treinadas com a arquitetura recursiva simples da sub-rede de refinamento. Os parâmetros e detalhes do treinamento foram explicados de forma detalhada no capítulo 5. A seguir é feita uma análise do desempenho de cada uma das variações de arquitetura, seguida pela escolha daquela mais promissora. Todos os erros apresentados aqui são calculados usando a função de erro minimizada no treinamento, ou seja, o erro logarítmico de escala invariante usando coeficiente λ igual a 0,5. Os tamanhos de imagens utilizados foram de 80x80 para as entradas e 40x40 para o mapa de profundidades.

6.1.1 O treinamento da sub-rede de aproximação inicial usando a rede convolucional VGG19

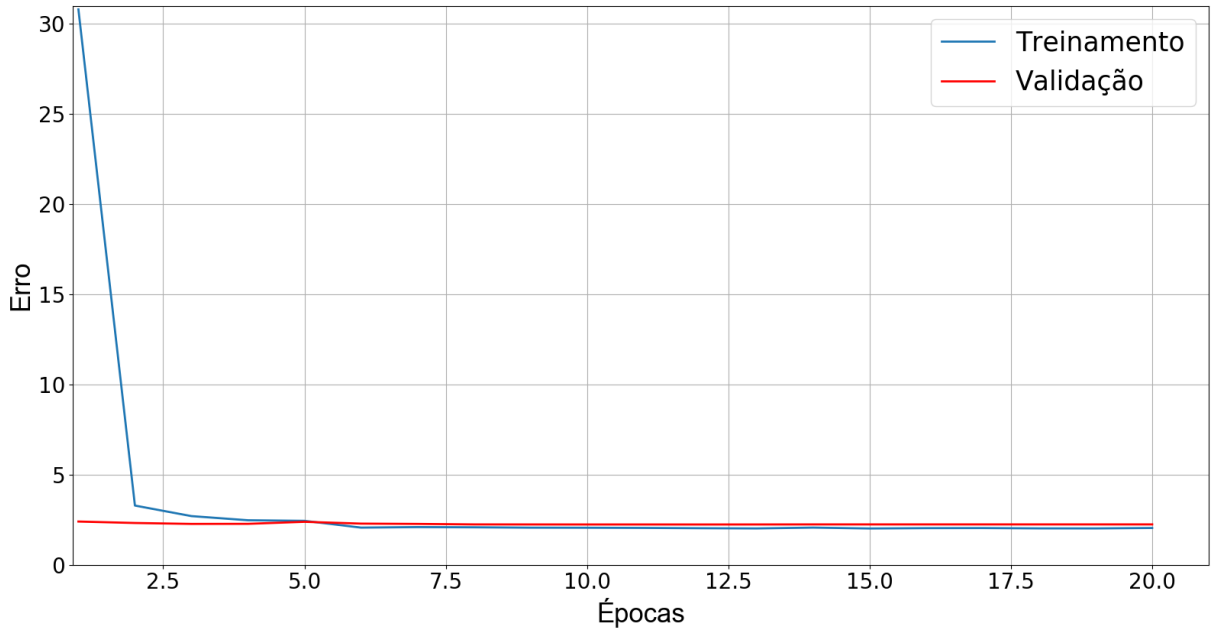
A arquitetura usando a rede VGG19 apresenta em torno de 50 milhões de parâmetros treináveis. Ela treinou por 20 épocas antes de convergir, o que aconteceu após aproximadamente 20 horas de treinamento. O erro médio de validação dessa arquitetura no treinamento foi de 2,236, enquanto que no corpo de testes ela obteve erro de 0,9033. Na Figura 21 está ilustrada a evolução dos erros de treinamento e de validação ao longo do treinamento.

Pode-se perceber que o erro de treinamento começa muito alto e diminui bruscamente nas primeiras épocas antes de se estabilizar, enquanto o erro de validação começa

menor e diminui devagar ao longo do treinamento. Essa diferença de valor inicial entre os dois erros ocorre, pois cada valor do erro de treinamento mostrado é o erro médio de treinamento ao longo de uma época inteira. Já o erro de validação é calculado apenas no final de cada época. Durante a primeira época de treinamento o erro de treinamento começa muito alto e diminui rapidamente. Dessa forma, o erro médio de treinamento da primeira época é muito maior do que o erro de validação da mesma, mesmo que o erro de treinamento ao final da época seja próximo do erro de validação.

As últimas épocas do treinamento tiveram muito pouca variação no erro de validação devido ao critério de parada do treinamento. Ele consiste na ocorrência de dez épocas consecutivas com uma variação no erro de validação menor que 0,001. Esse comportamento pode ser observado em todos os treinamentos realizados.

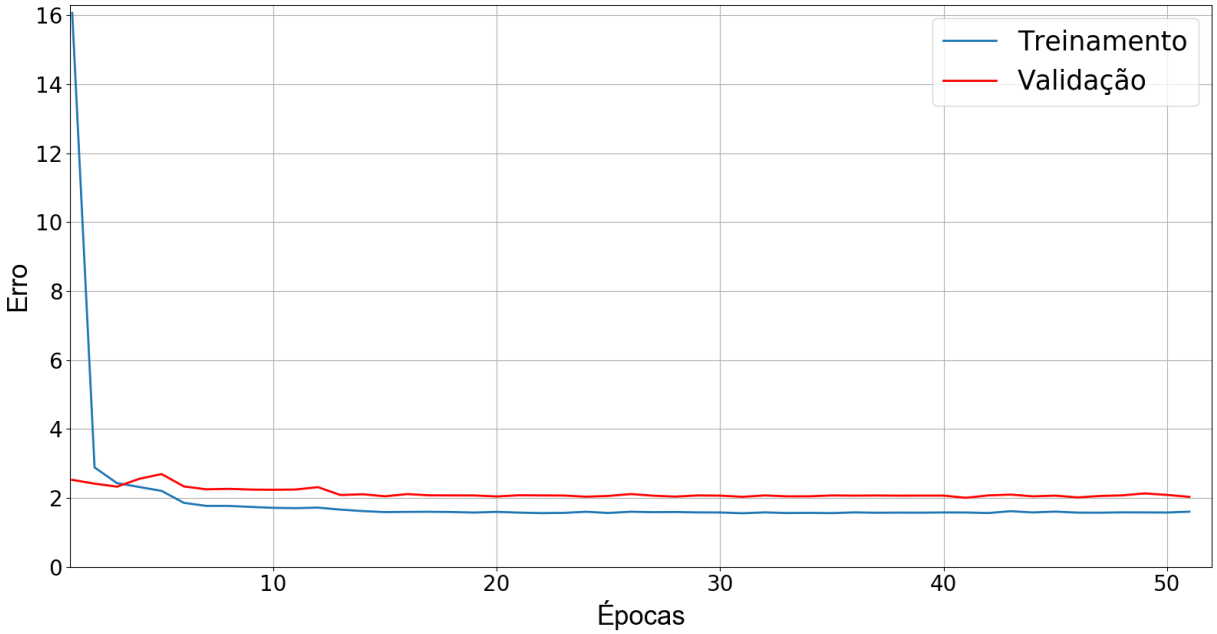
Figura 21 – Variação do erro por época de treinamento - VGG19



6.1.2 O treinamento da sub-rede de aproximação inicial usando a rede convolucional Xception

Como a rede *Xception* é a maior das redes convolucionais importadas, a arquitetura que a utiliza é a que apresenta mais parâmetros de treinamento, sendo eles cerca de 60 milhões. Essa arquitetura foi também a que levou mais tempo para convergir, treinando por 51 épocas ao longo de 44 horas. Pode-se observar que o tempo de treinamento médio de cada época nessa arquitetura foi de 52 minutos, um pouco menor do que os 60 minutos obtidos na arquitetura VGG19. Ao final do treinamento essa arquitetura obteve erro de 0,8125 no corpo de testes e erro de validação médio de 2,029. Na Figura 22 pode-se observar o comportamento dos erros de treinamento e validação ao longo do treinamento.

Figura 22 – Variação do erro por época de treinamento - *Xception*



6.1.3 O treinamento da sub-rede de aproximação inicial usando a rede convolucional YOLO

A rede YOLO é a menor das redes convolucionais importadas e a arquitetura que a utiliza apresenta apenas 43 milhões de parâmetros de aprendizado. Ela também foi a que convergiu mais rápido, precisando de cerca de 17 horas, ao longo das quais a rede treinou por 27 épocas. Pode-se observar que apesar de o tempo de treinamento dessa rede ter sido o mais curto, ela ainda treinou por sete épocas a mais do que a rede VGG19, tendo um tempo médio de treinamento de 38 minutos por época. Após a convergência, a rede apresentou erro no corpo de testes de 1,412 e erro de validação médio de 2,681. Na Figura 23 pode-se conferir a evolução do erro de treinamento e de validação ao longo do treinamento.

6.1.4 Escolha da sub-rede de aproximação inicial mais promissora

Comparando os resultados de treinamento expostos anteriormente, percebe-se que a arquitetura com a rede *Xception* apresentou os menores erros no corpo de testes. Além disso, essa arquitetura também obteve erros de validação e de treinamento inferiores às demais.

Na Tabela 2 é comparado o desempenho das três arquiteturas. Os parâmetros analisados são os erros de treinamento, validação e teste, além do tempo médio que a rede leva para processar um único quadro de um vídeo.

Nos três parâmetros de erro, as redes apresentaram um desempenho consistente.

Figura 23 – Variação do erro por época de treinamento - YOLO

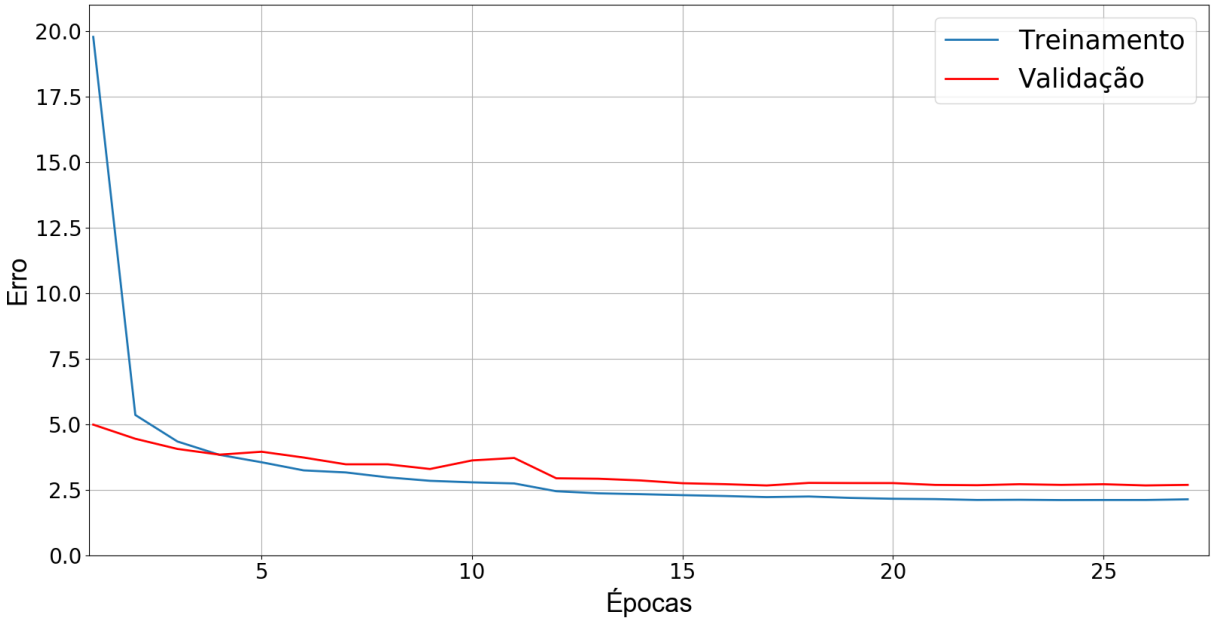


Tabela 2 – Desempenho das redes convolucionais

	Erro logarítmico de escala invariante			Tempo para processar um quadro
	Treino	Validação	Testes	
Rede VGG19	2,0380	2,2362	0,9033	53,65 ms
Rede Xception	1,5994	2,0291	0,8125	34,32 ms
Rede YOLO	2,1307	2,6811	1,4124	10,17 ms

Aquelas que obtiveram menores erros de treinamento também obtiveram menores erros de validação e de testes. As três arquiteturas apresentaram erros de treinamento menores do que os erros de validação e erros de testes menores do que ambos. As arquiteturas *Xception* e VGG19 tiveram erros de teste muito próximos enquanto aqueles que a rede YOLO obteve foram significativamente maiores.

O único parâmetro analisado no qual a rede usando *Xception* não se mostrou a mais promissora foi no tempo médio de processamento, no qual a arquitetura YOLO provou-se a mais rápida, sendo três vezes mais rápida que a *Xception* e cinco vezes mais rápida que a VGG19. Porém, como mostrado no capítulo 2, a base de dados KITTI (GEIGER et al., 2013) usa o tempo como último critério de comparação em termos de relevância.

A arquitetura *Xception* foi, portanto, escolhida como a arquitetura convolucional mais promissora, devido ao seu desempenho superior no erro de testes, que foi definido inicialmente como o principal critério de seleção. Além disso, ela obteve desempenho bastante superior no tempo de processamento com relação a VGG19, que obteve resultados próximos no erro de teste.

6.2 Análise das sub-redes de refinamento e as opções recursivas

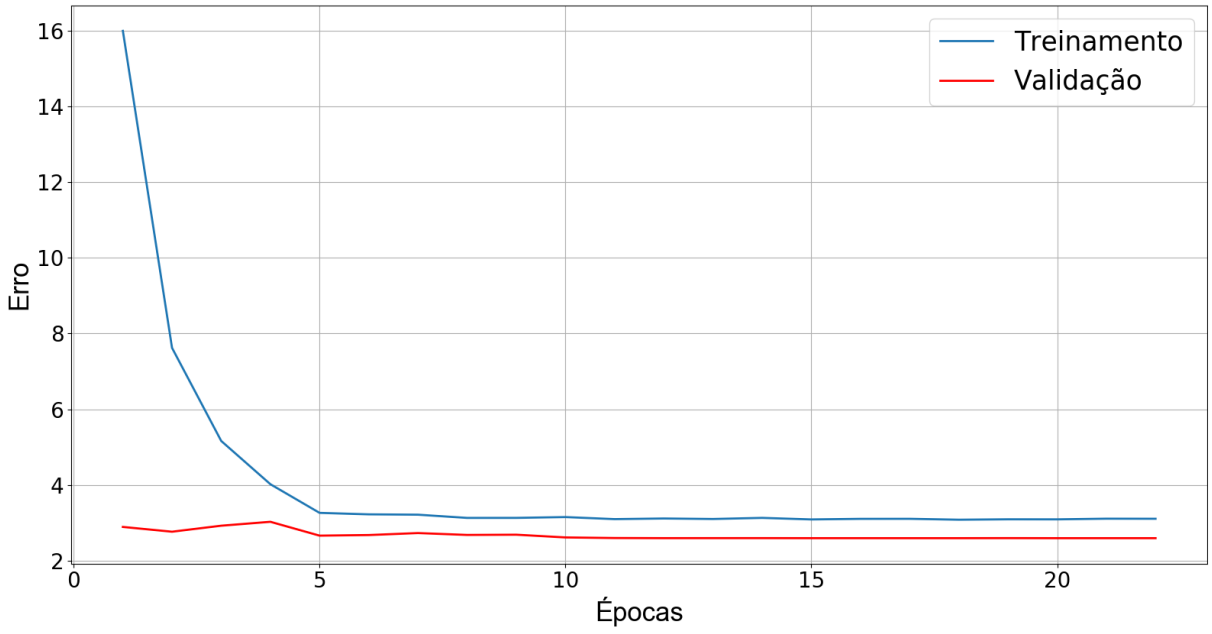
O treinamento das três alternativas de rede de refinamento que fazem uso das redes recursivas simples, LSTM e LSTM convolucional foi realizado usando a rede convolucional *Xception*, selecionada na seção anterior, na sub-rede de aproximação inicial. A seguir são analisados os resultados de cada uma das três alternativas de arquitetura recursiva e a mais promissora é escolhida. Novamente todos os erros apresentados aqui são calculados usando o erro logarítmico de escala invariante com coeficiente λ igual a 0,5 minimizado no treinamento.

Conforme mencionado anteriormente no capítulo 5, o tamanho das imagens utilizadas nessa etapa de treinamento (240x80 nas entradas e 120x40 para os mapas de profundidades) foi maior do que na etapa anterior, o que aumentou o tamanho das redes, devido a sua influência nas camadas totalmente conectadas.

6.2.1 O treinamento da sub-rede de refinamento usando a rede recursiva simples

A arquitetura que faz uso de camadas recursivas simples apresenta cerca de 150 milhões de parâmetros de aprendizado. Ela treinou por 22 épocas ao longo de 91 horas, demorando em torno de 4 horas e 8 minutos por época. Ao final do treinamento essa arquitetura obteve erro de validação de 2,5898 e erro de 2,152 no corpo de testes. Na Figura 24 é mostrado o comportamento dos erros de treinamento e de validação dessa arquitetura ao longo do treinamento.

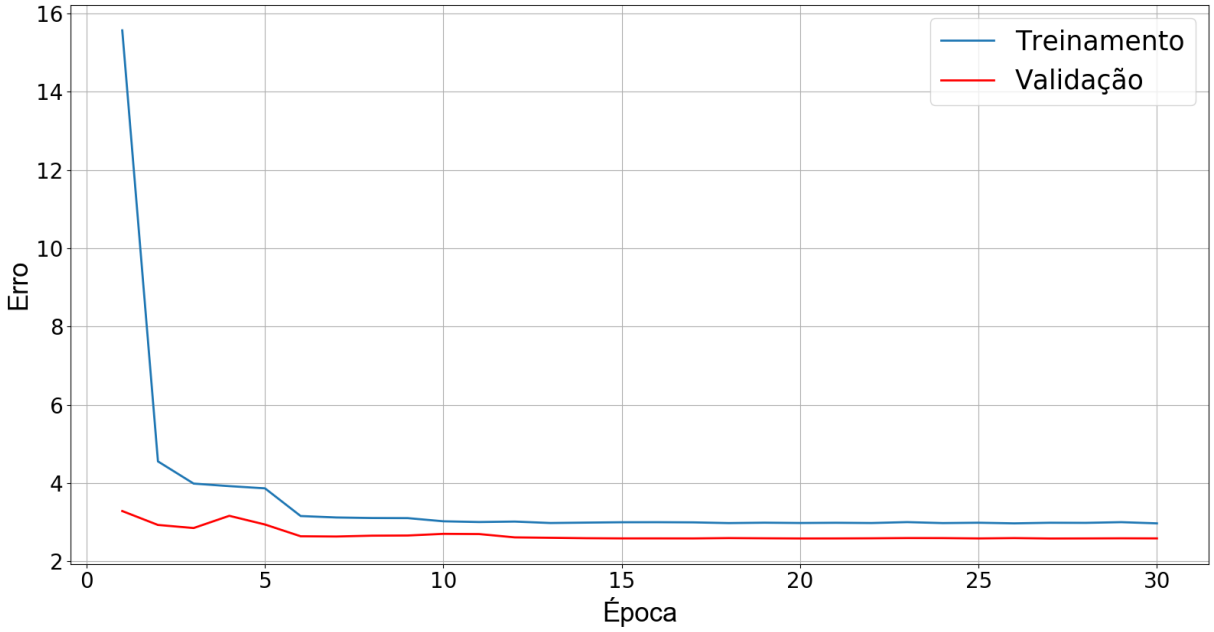
Figura 24 – Variação do erro por época de treinamento - Recursiva simples



6.2.2 O treinamento da sub-rede de refinamento usando a rede LSTM

A arquitetura da rede de refinamento baseada em redes LSTM é a que apresenta mais parâmetros de treinamento, com cerca de 455 milhões deles sendo de aprendizado. São três vezes mais parâmetros do que a arquitetura recursiva simples e quase nove vezes mais do que a arquitetura LSTM convolucional. Essa arquitetura foi a que mais demorou para convergir no treinamento, levando aproximadamente 236 horas e 30 épocas, com tempo médio de 7 horas e 52 minutos por época. Apesar de ser a rede mais complexa e que treinou por mais tempo, a LSTM não obteve o menor erro no corpo de testes, ficando com 2,263. Pode-se observar na Figura 25 o comportamento da arquitetura LSTM ao longo do seu treinamento.

Figura 25 – Variação do erro por época de treinamento - LSTM



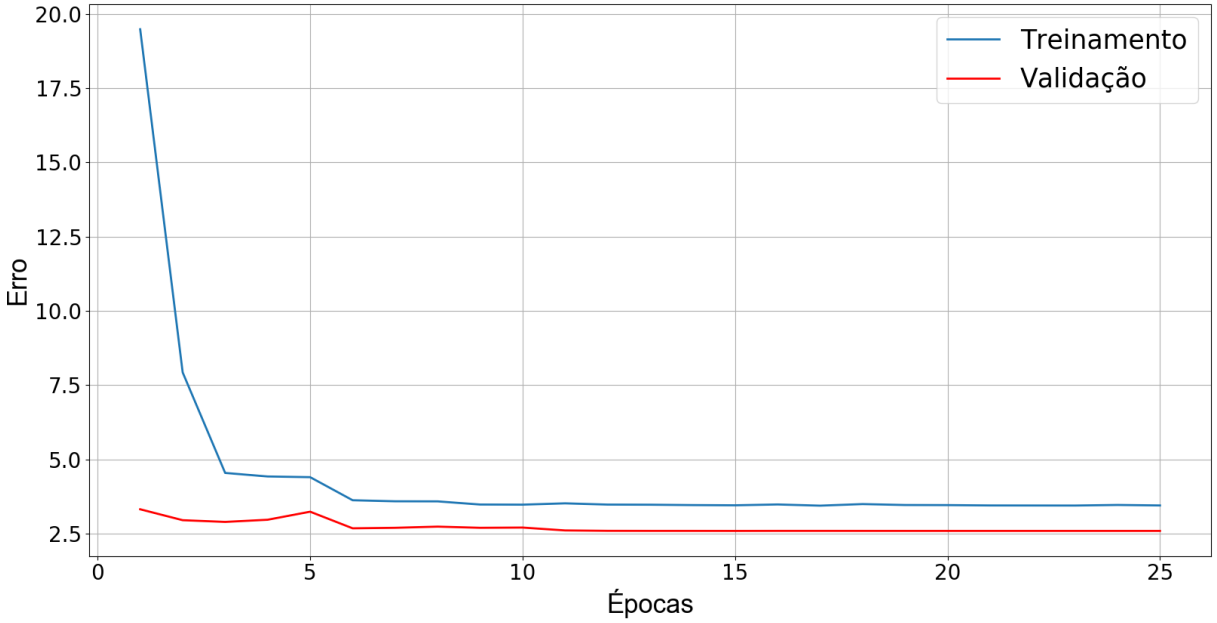
Apesar do resultado nos testes parecer contra intuitivo, uma vez que a camada LSTM foi desenvolvida de forma a melhorar a capacidade de memória das camadas recursivas simples, a dupla inferiu que esse comportamento é uma consequência da relevância maior de memória de curto prazo sobre aquela de longo prazo no caso de aplicação deste projeto. Isso pôde ser observado, por exemplo, quando o vídeo analisado retrata o automóvel fazendo curvas. Neste caso, a memória de longo prazo tende a atrapalhar mais a estimativa seguinte do que a de curto, dado que as mudanças de quadro a quadro tendem a ser mais significativas.

Além disso, como o número de exemplos de treinamento utilizados foi restrito e o tamanho das imagens limitado, a rede LSTM pode ter sofrido com algum efeito de *overfitting* acentuado, se comparado ao desempenho das outras redes recursivas.

6.2.3 O treinamento da sub-rede de refinamento usando a rede LSTM convolucional

O treinamento da arquitetura de rede de refinamento baseado em camadas LSTM convolucionais demorou aproximadamente 91 horas, levando em torno de dez minutos a menos que a arquitetura recursiva simples, durante as quais a rede treinou por 25 épocas. Essa arquitetura é a mais leve, com cerca de 56 milhões de parâmetros de aprendizado. Ela obteve resultado ligeiramente melhor que a arquitetura LSTM totalmente conectada nos corpos de testes com erro de 2,255. No erro de validação ela obteve o pior resultado, embora ele tenha sido muito próximo daqueles das outras arquiteturas com 2,597. Com relação ao erro no treinamento, o seu desempenho foi bastante inferior àqueles das outras redes. Na Figura 26 é ilustrada a evolução dos erros de treinamento e validação durante o treinamento da arquitetura LSTM convolucional.

Figura 26 – Variação do erro por época de treinamento - LSTM convolucional



Esperava-se que essa rede obtivesse resultados superiores àqueles de fato obtidos, devido às capacidades de relação espaço-temporal das camadas LSTM convolucionais. Percebeu-se, porém, que a arquitetura montada não é capaz de fazer uso pleno dessa característica desse tipo de camada recursiva, sendo que, provavelmente uma arquitetura mais próxima daquela proposta por (KUMAR; BHANDARKAR; PRASAD, 2018), na qual a parte convolucional inicial da rede é totalmente substituída por camadas LSTM convolucionais, seria mais bem-sucedida. Essa arquitetura, porém, diferiria muito daquelas propostas neste trabalho e não poderia ser treinada seguindo o processo de treinamento modular deste projeto, ou seja, treinando e escolhendo inicialmente a parte convolucional da rede (sub-rede de aproximação inicial) e posteriormente a parte recursiva (sub-rede de

refinamento).

6.2.4 Escolha da sub-rede de refinamento mais promissora

Ao se analisar o desempenho das arquiteturas da sub-rede de refinamento, mostrado na Tabela 3, percebe-se que a arquitetura recursiva simples obteve os melhores resultados no corpo de testes, sendo consideravelmente melhor do que aqueles das arquiteturas LSTM e LSTM convolucional. Porém, diferentemente dos resultados da etapa de treino da sub-rede de aproximação inicial, os erros de validação e de treinamento não acompanharam o comportamento do erro de testes. As arquiteturas LSTM e recursiva simples apresentaram erros de validação muito próximos, sendo aqueles obtidos pela primeira marginalmente menores do que os pela segunda e ambos menores que o erro da LSTM convolucional. Com relação aos erros de treinamento, a LSTM apresentou o menor valor, com uma distância considerável da recursiva simples e uma ainda maior da LSTM convolucional.

Tabela 3 – Desempenho das redes recursivas

	Erro logarítmico de escala invariante			Tempo para processar um quadro
	Treino	Validação	Testes	
Recursiva simples	3,1036	2,5898	2,1519	100,35 ms
LSTM	2,9735	2,5889	2,2626	127,17 ms
LSTM convolucional	3,4544	2,5973	2,2547	109,20 ms

As três arquiteturas apresentaram tempos de processamento de cada quadro muito próximos, sendo a arquitetura recursiva simples a mais rápida com 100,35 ms, seguida pela LSTM convolucional com 109,2 ms e, por fim, a LSTM totalmente conectada com 127,17 ms.

Sendo assim, devido ao seu melhor desempenho no corpo de testes, o que mostra uma robustez maior, e ao seu menor tempo médio de processamento, a rede recursiva simples foi aquela escolhida neste trabalho para guardar relações temporais entre os quadros consecutivos de vídeos e assim compor a versão final da sub-rede de refinamento.

6.3 Análise da rede final

Conforme definido anteriormente, a arquitetura de rede mais promissora é a que faz uso da rede *Xception* na sub-rede de aproximação inicial e de camadas recursivas simples na sub-rede de refinamento. Os itens seguintes analisam o desempenho dessa configuração de rede em comparação com os requisitos estabelecidos no capítulo 2.

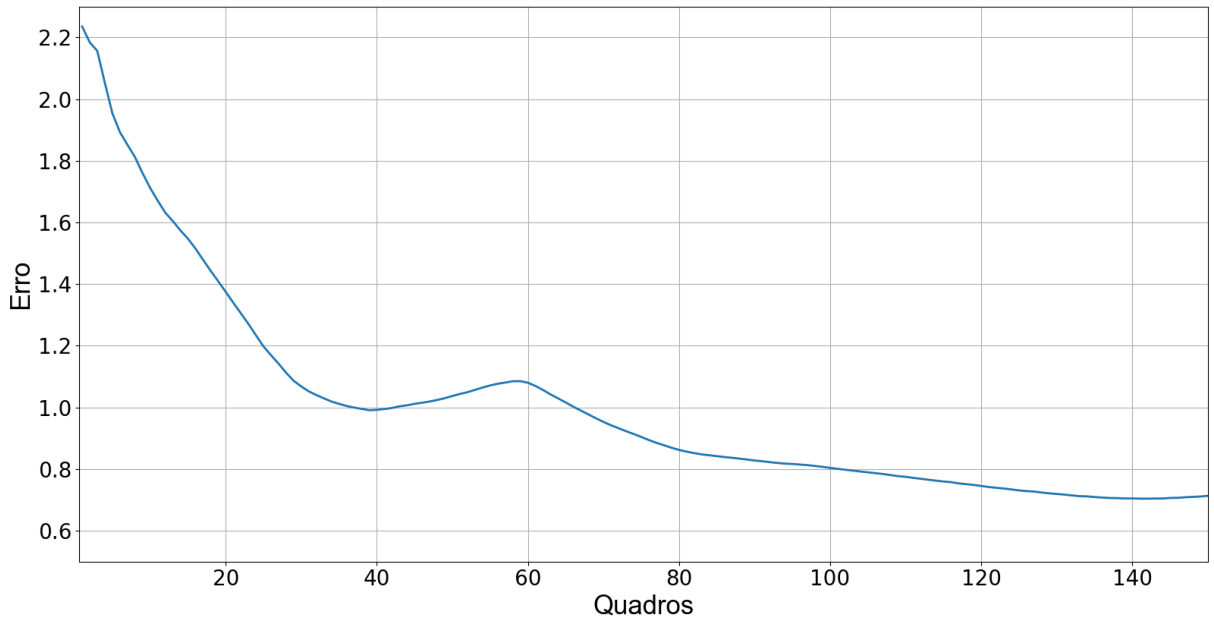
6.3.1 Influência do uso de uma rede recursiva

O principal objetivo deste projeto é o de avaliar a influência do uso de informações temporais na estimativa de profundidade. A expectativa é de que conforme o tempo fosse passando, a parte recursiva da rede começaria a ter cada vez mais informações para utilizar e, dessa forma, caso não ocorressem mudanças muito súbitas nos padrões das imagens entre um instante e outro, o erro médio tenderia a diminuir.

Para comprovar o comportamento esperado, foram criadas n sequências de imagens com um número crescente de quadros de um mesmo vídeo. Ou seja, a primeira apresenta apenas o primeiro quadro, e cada sequência adicional contém um quadro consecutivo a mais que a anterior, até atingir o tamanho n total de quadros. Cada uma dessas sequências foi alimentada à rede já treinada e os erros médios obtidos para elas foram calculados. Desse modo, é possível avaliar como a adição de um quadro no vídeo afeta o erro total da estimativa.

Na Figura 27 é mostrado o gráfico gerado com esses erros para um vídeo com 145 quadros. Conforme era esperado, o erro médio de cada sequência tendeu a diminuir com o aumento do número de quadros, mostrando a influência positiva que as informações de memória têm na rede. As oscilações no erro são atribuídas a mudança bruscas no vídeo que atrapalham o uso da memória. Nesse vídeo especificamente, as câmeras filmaram o movimento do carro numa rodovia em linha reta até em torno do quadro 40, quando um outro carro cruza a visão da câmera rapidamente e causa um pequeno aumento no erro das estimativas.

Figura 27 – Variação do erro por quadro: Vídeo I

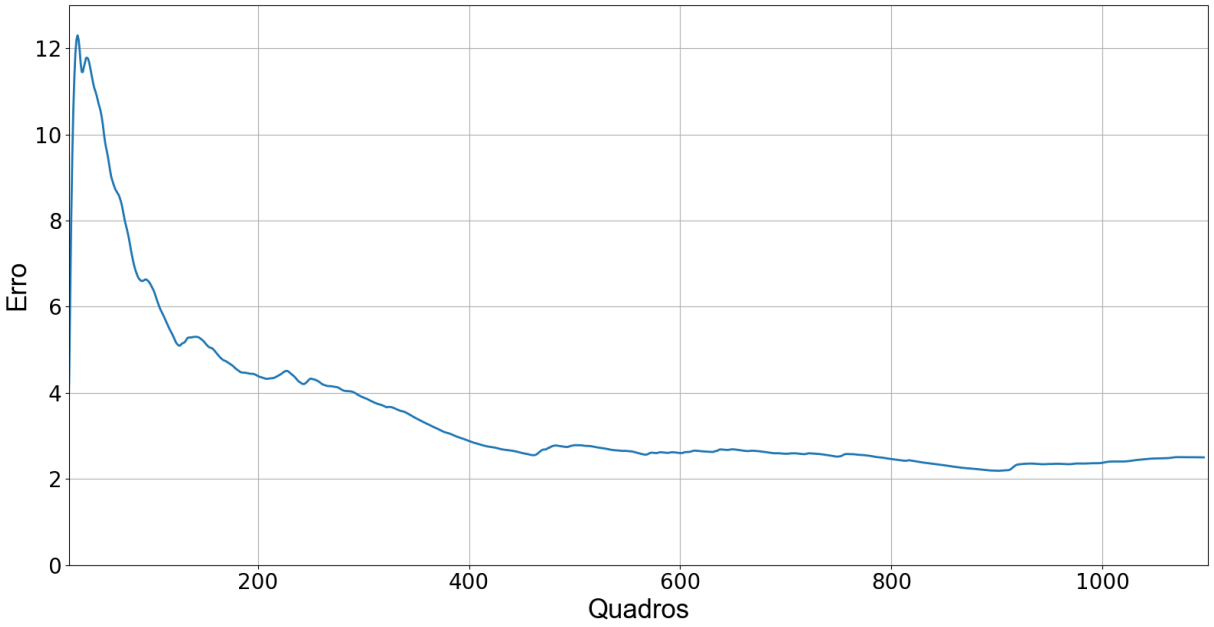


Esse tipo de mudança brusca nas imagens pode fazer com que a influência da

memória na rede seja inicialmente negativa, porém pode-se perceber que, mesmo com essa interferência, o erro médio voltou a diminuir nas sequências posteriores. Isso acontece, pois, mesmo que a memória atrapalhe a operação da rede durante algumas mudanças súbitas ou não usuais nas imagens, a maior parte dos quadros das sequências apresentam padrões de mudanças mais previsíveis, com os quais a rede aprendeu a lidar durante o treinamento.

Na Figura 28 é ilustrado um caso extremo desse problema, no qual o mesmo experimento é feito em um vídeo que começa com o carro fazendo rapidamente uma curva de noventa graus em um cruzamento. Nessa situação, a janela da imagem alimentada para a rede (240x80) tem o seu interior mudando drasticamente conforme o carro vira, passando de maneira rápida por imagens muito distintas, que mostram desde pedestres até outros carros e fachadas de prédios. A curva termina ainda no início do vídeo e, a partir daí, pode-se observar que o erro médio começa a cair drasticamente com o aumento do tempo. Durante o percurso realizado para este mesmo vídeo, o carro realizou outras curvas e outros eventos, como a travessia de pedestres, também ocorreram e aumentaram localmente o erro. Essas pequenas variações podem ser identificadas pelos outros picos do gráfico.

Figura 28 – Variação do erro por quadro: Vídeo II



6.3.2 Comparação com a banco de imagens KITTI

Para a arquitetura de rede estabelecida foram calculados os quatro erros utilizados pela plataforma KITTI (GEIGER et al., 2013) para avaliação do desempenho de redes. Todos esses cálculos foram realizados para o corpo de testes definido pela dupla, sendo que o mesmo não corresponde àquele utilizado na tabela de comparação KITTI (UHRIG et al.,

2017), visto que o último não é disponibilizado pelos responsáveis pelo banco de imagens à trabalhos de estudantes de graduação. Os resultados obtidos neste projeto são apresentados na Tabela 4.

Tabela 4 – Desempenho da arquitetura proposta nas métricas da biblioteca KITTI

	Erro logarítmico de escala invariante	Erro relativo quadrado	Erro relativo absoluto	iRSME	Tempo de processamento de um quadro
Recursiva Simples	1,95	4,25 %	13,02 %	14,86	100,35 ms

Ao comparar esses erros com a Tabela 1 de previsão de profundidade, pode-se observar que a rede desenvolvida apresenta ótimos resultados nos erros logarítmico de escala invariante e *iRSME* e resultados bons, mas inferiores, nos erros relativo quadrado e relativo absoluto.

No caso dos dois primeiros erros, que avaliam a capacidade da rede em reproduzir a tendência média dos mapas de profundidade, a rede teve um excelente desempenho. Esse desempenho foi melhor do que o obtido pelo primeiro colocado da Tabela 1 para o erro logarítmico de escala invariante e o segundo melhor para o *iRSME*. Já para os erros relativos quadrado e absoluto, que avaliam a capacidade da rede em reproduzir variações locais na profundidade, observou-se um desempenho inferior, mas ainda muito bom. A rede proposta estaria na 14ª colocação em relação ao erro relativo quadrado e na 16ª em relação ao erro relativo absoluto.

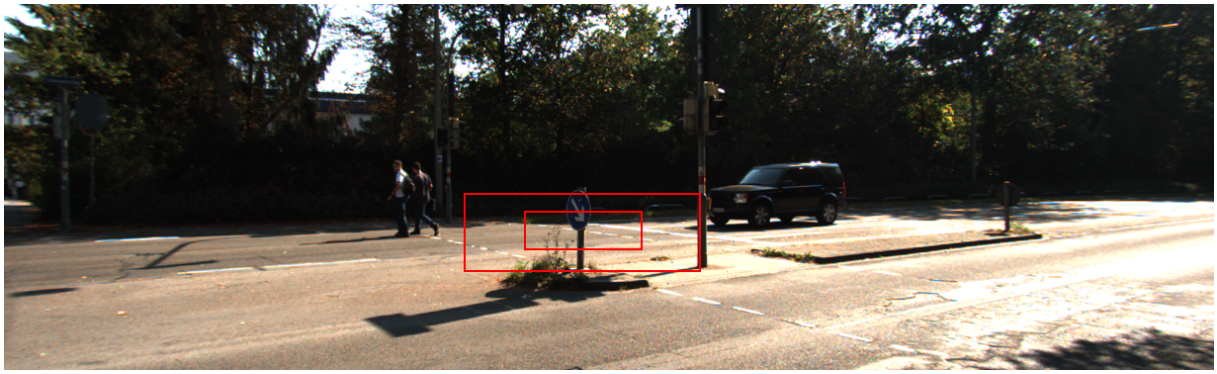
Esse sucesso mostra o grande potencial do uso de informações temporais no cálculo de profundidade de imagens. É importante entretanto mencionar que as entradas na colocação do KITTI (UHRIG et al., 2017) são de redes neurais e algoritmos para o cálculo da profundidade dos pixels de uma imagem utilizando apenas informações da própria imagem, diferentemente da rede proposta neste trabalho. Além disso, outra diferença é que a arquitetura deste projeto recebe como entrada janelas de dimensão 240x80 pixels posicionadas na região central da imagem original e o mapa de profundidades produzido corresponde a apenas um quarto deste tamanho, como mostrado pelos retângulos vermelhos na Figura 29, enquanto que os métodos presentes na tabela 1 utilizam tamanhos maiores de imagem. Dessa forma, a rede proposta não se qualifica para competir nessa categoria, mas a comparação mostra a grande influência do uso de informações temporais dos quadros anteriores e de uma rede siamesa estéreo no desempenho da rede.

Desse modo, os resultados obtidos nessas quatro métricas de erro diferentes mostram que a rede desenvolvida é capaz de produzir uma boa aproximação dos valores de profundidade médios da imagem, conforme pode ser visto pelo bom desempenho nos erros *iRMSE* e logarítmico de escala invariante. A rede, porém, não é capaz de obter uma

precisão tão boa para mudanças locais na profundidade causadas por detalhes menores, conforme o resultado inferior nos erros relativos revela.

Essa dificuldade da rede em lidar com variações locais é, provavelmente, uma consequência do gargalo, já explicado no capítulo 4, presente na sua arquitetura. Para que uma rede seja capaz de produzir saídas precisas global e localmente deve-se garantir que as menores saídas internas da rede tenham ao menos a mesma dimensão que a saída final da rede, 4800 pixels. Como o gargalo produz uma saída de dimensões 16x16x1 (256 pixels), enquanto o mapa de profundidade que se deseja montar apresenta dimensões de 120x40x1 (4800 pixels) é provável que ele possa ter contribuído para os resultados inferiores nos erros relativos.

Figura 29 – Comparação do tamanho original da imagem com os utilizados para este projeto. A janela vermelha externa mostra a região da imagem recebida pela rede, enquanto a janela interior mostra a região para a qual o mapa de profundidade é gerado



Com relação ao tempo médio de processamento de cada quadro, a rede desenvolvida obteve o resultado de 100,35 ms. Observando as Tabelas 4 e 1, poder-se-ia concluir que esse resultado é muito bom, uma vez que entre as 20 melhores entradas, os tempos de processamento variaram de 20 ms até 2 s, com a maioria na faixa dos 200 ms. Entretanto, essa análise não estaria totalmente correta por duas razões. A primeira é que esses tempos de processamento dependem fortemente do *hardware* utilizado, ou seja, a comparação dos tempos de processamento sem levá-lo em conta pode ser enganosa. Os métodos listados na tabela foram todos processados em GPUs com frequências de 1,5 até 3,5 GHz, sendo a grande maioria com frequência de 2,5 GHz. Conforme mencionado anteriormente, todas as redes desenvolvidas nesse trabalho foram processadas em CPUs, notavelmente mais lentas do que GPUs, devido a limitações da memória das GPUs utilizadas.

A outra ressalva que atrapalha a comparação direta dos tempos de processamento é que a rede deste projeto processa imagens de dimensões 240x80, ou seja, menores do que o tamanho original. Desse modo, conforme explicado no capítulo 3, o uso de imagens maiores aumentaria o número de parâmetros da rede e, consequentemente, o seu tempo de

processamento.

Após a realização de testes com imagens de diferentes tamanhos para medir a influência do número de pixels no tempo de processamento médio, concluiu-se que a relação entre as duas grandezas é aproximadamente linear. Isso pode ser observado facilmente comparando-se o tempo de processamento das arquiteturas treinadas para seleção das duas sub-redes (seções 6.1 e 6.2). Em ambos os casos foram treinadas e analisadas duas redes com a mesma arquitetura base, sendo ela com a sub-rede de aproximação inicial baseada na rede *Xception* e a sub-rede de refinamento utilizando camadas de arquitetura recursiva simples. A única diferença entre essas duas se deve ao segundo treinamento ter sido realizado com tamanhos de imagem maiores do que o primeiro, sendo eles 240x80 (19.200 pixels) e 80x80 (6400 pixels, três vezes menor) respectivamente. Para o caso de imagens menores, a rede possui cerca de 60 milhões de parâmetros e tempo de processamento médio de 34,32 ms, enquanto que para imagens maiores ela possui 150 milhões de parâmetros e tempo de processamento médio de 100,35 ms. Pode-se, portanto, perceber que o aumento em três vezes no tamanho da imagem aumentou o tempo de processamento médio de cada quadro em pouco menos de três vezes, o que evidencia a relação aproximadamente linear entre as duas grandezas.

A partir disso, é possível ser feita uma estimativa do tempo médio de processamento da rede proposta caso a mesma operasse com tamanhos de imagens diferentes. Supondo as imagens originais da base de dados KITTI (GEIGER et al., 2013), com dimensões de 1242x375 (465.750 pixels), ou seja, aproximadamente 24 vezes maior do que aquelas utilizadas no segundo treinamento, pode-se estimar um tempo de processamento de aproximadamente 2,4 s. Esse tempo é significativamente superior àquele obtido pela maioria dos trabalhos listados no ranking da base de dados KITTI (UHRIG et al., 2017), porém é importante ressaltar que a arquitetura aqui analisada processa duas imagens e possui camadas recursivas, o que acarreta uma quantidade muito maior de informação sendo levada em conta ao mesmo tempo.

6.3.3 Limitações do projeto

É importante ressaltar novamente que, apesar dos bons resultados obtidos, algumas limitações afetaram a execução deste projeto, impedindo que as estimativas de profundidade fossem ainda melhores. A principal delas foi a grande demanda por capacidade computacional exigida para o treinamento de uma rede neural convolucional e recursiva. Apesar da equipe ter acesso ao supercomputador HPC da USP, sem o qual esse trabalho jamais poderia ter sido concluído, seria necessária uma capacidade computacional maior do que a disponível para treinar a rede idealizada inicialmente.

Com o intuito de elucidar as dificuldades enfrentadas para o treinamento da rede proposta por este projeto, as peculiaridades do treinamento de redes neurais com aplicações

similares são explicadas nos parágrafos seguintes. Posteriormente uma comparação é feita com o processo de treinamento da arquitetura desenvolvida neste trabalho, de modo a permitir um melhor entendimento das dificuldades e limitações que moldaram as escolhas dos hiperparâmetros escolhidos.

Para o problema de visão monocular, as redes utilizadas têm menos parâmetros e, uma vez que essas redes ignoram as relações sequenciais entre os quadros, elas podem utilizar *batches* de treinamento arbitrariamente pequenos para resolver problemas de falta de memória. Por essa razão, elas necessitam de menos capacidade computacional para o seu treinamento. A rede de (EIGEN; PUHRSCH; FERGUS, 2014), por exemplo, foi treinada por 64 horas numa placa de vídeo *NVidia GTX Titan Black*, disponível comercialmente e que pode ser instalada em computadores pessoais.

Para o caso do problema de visão estéreo, as redes tradicionalmente buscam encontrar pares de pixels equivalentes para calcular a disparidade, e não a profundidade diretamente. Por essa razão, as redes podem ser treinadas usando apenas janelas da imagem original, o que diminui drasticamente o tamanho de cada imagem utilizada no treinamento. Além disso, essas redes também não consideram as relações sequenciais entre os quadros, permitindo o uso de *batches* de treinamento ainda menores, se necessário. A rede de (ZBONTAR; LECUN, 2015), por exemplo, foi treinada por menos de dois dias utilizando a placa de vídeo *NVidia GTX Titan*, muito semelhante a utilizada por (EIGEN; PUHRSCH; FERGUS, 2014).

Outro caso é o de redes com um número muito grande de parâmetros, como convolucionais profundas ou recursivas. Pode acontecer do próprio modelo da rede se tornar muito grande, podendo chegar a dezenas de *Gigabytes*. Nesses casos, placas de vídeo comerciais podem não ter memória o suficiente para carregar nem mesmo o modelo da rede. Para contornar esse problema, reduzir o tamanho dos batches de treinamento ou dos objetos de treinamento não é suficiente e um supercomputador precisa ser utilizado. Esse é caso da rede *Xception*, uma rede convolucional com um número muito grande de camadas e de parâmetros, que foi treinada por três dias usando um supercomputador com 60 GPUs *Nvidia K80* (CHOLLET, 2016).

A rede desenvolvida neste trabalho é extremamente grande, uma vez que faz uso da arquitetura *Xception* nos dois ramos siameses da sub-rede de aproximação inicial e da arquitetura recursiva na sub-rede de refinamento. Além disso, a redução excessiva do tamanho dos *batches* de treinamento não foi possível, dado que isso reduziria a capacidade da parte recursiva da rede de aprender.

Dessa forma, a arquitetura proposta se mostrou pesada demais para ser treinada no HPC, mesmo usando a sua CPU. Para que a rede fosse treinada, algumas características da rede idealizada originalmente tiveram que ser alteradas, sendo a primeira medida a redução do tamanho das imagens utilizadas, diminuindo assim drasticamente o número

de parâmetros da arquitetura proposta. A segunda alteração foi a redução do tamanho da camada totalmente conectada presente no final da sub-rede de aproximação inicial. Essa redução introduz um gargalo estreito na rede, porém ele se fez necessário, dado que o uso de camadas maiores aumentava drasticamente o número de parâmetros da rede, impossibilitando o seu treinamento.

Sem dúvida essas alterações reduziram o desempenho e a aplicabilidade da rede desenvolvida, mas infelizmente foram a única forma de compatibilizar a ideia de rede original com a capacidade computacional disponível. Por essa razão, esse trabalho serve para mostrar a viabilidade e o potencial da arquitetura proposta, mesmo que numa versão em escala reduzida.

7 CONCLUSÕES

A rede proposta no capítulo 4 apresentou excelentes resultados de acordo com as métricas de erro utilizadas para a comparação com outros projetos, sendo eles comparáveis e, para alguns casos, melhores do que os resultados obtidos por outros métodos de estimação de profundidade que utilizaram a base de dados KITTI (GEIGER et al., 2013).

Os bons resultados obtidos comprovam, portanto, o potencial da combinação das arquiteturas siamesas convolucionais, usadas tradicionalmente na visão estéreo, com camadas recursivas capazes de guardar informações temporais. Dessa forma, o projeto pode ser considerado bem-sucedido, uma vez que a rede treinada atendeu ao requisito de mostrar a eficiência do uso de redes recursivas no problema abordado e, além disso, obteve ótimos resultados nas métricas definidas pela biblioteca KITTI (GEIGER et al., 2013).

7.1 Sugestões para trabalhos futuros

Apesar dos bons resultados, o projeto passou por limitações durante a sua execução, sendo a principal delas a capacidade computacional. Por esse motivo, reconhece-se que algumas medidas podem ser adotadas por trabalhos futuros para a melhoria dos resultados deste projeto.

A primeira delas seria o uso de um supercomputador mais potente. Isso possibilitaria que imagens de um tamanho significativamente maior fossem utilizadas e, consequentemente, os problemas de mudanças súbitas nos padrões das imagens, explicados no capítulo 6, seriam menores, acarretando melhor aproveitamento da capacidade de memória das redes recursivas. Além disso, com o aumento da capacidade computacional, seria interessante livrar a arquitetura da rede proposta de seus gargalos de informação. A saída das redes siamesas, por exemplo, poderia retornar uma imagem nas dimensões do mapa de profundidades final e, desse modo, menos informações seriam perdidas, melhorando as estimativas finais geradas pela rede.

Outra possível melhoria considerada muito promissora, mas que não foi muito explorada, é o desenvolvimento de uma arquitetura de rede capaz de utilizar ao máximo as capacidades das camadas LSTM convolucionais. Ao longo do andamento do projeto, alternativas de redes diferentes foram contempladas e algumas delas tiveram que ser descartadas, mesmo sendo promissoras, devido à dificuldade de conciliar sua realização com a do resto do projeto.

Sendo assim, com a implementação das mencionadas melhorias, acredita-se que as ideias propostas por este projeto seriam ainda mais promissoras e os seus resultados

cada vez mais assertivos, o que acarretaria em melhorias nas diversas áreas nas quais a estimação de profundidades de objetivos em vídeos pode ser aplicada.

REFERÊNCIAS

ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <http://tensorflow.org/>.

ADAMY, J. *Fuzzy Logik, Neuronale Netze und Evolutionäre Algorithmen*. [S.l.]: Shaker Verlag, 2015. ISSN 4.

CHEN, Z. et al. A deep visual correspondence embedding model for stereo matching costs. *2015 IEEE International Conference on Computer Vision (ICCV)*, p. 972–980, 2015.

CHENG, F.; HE, X.; ZHANG, H. Learning to refine depth for robust stereo estimation. *Pattern Recogn.*, Elsevier Science Inc., New York, NY, USA, v. 74, n. C, p. 122–133, fev. 2018. ISSN 0031-3203. Disponível em: <https://doi.org/10.1016/j.patcog.2017.07.027>.

CHOLLET, F. *keras*. [S.l.]: GitHub, 2015. <https://github.com/fchollet/keras>.

CHOLLET, F. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016. Disponível em: <http://arxiv.org/abs/1610.02357>.

DENG, J. et al. ImageNet: A Large-Scale Hierarchical Image Database. In: *CVPR09*. [S.l.: s.n.], 2009.

EIGEN, D.; PUHRSCH, C.; FERGUS, R. Depth map prediction from a single image using a multi-scale deep network. In: GHAHRAMANI, Z. et al. (Ed.). *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014. p. 2366–2374. Disponível em: <http://papers.nips.cc/paper/5539-depth-map-prediction-from-a-single-image-using-a-multi-scale-deep-network.pdf>.

GEIGER, A. et al. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

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

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural Comput.*, MIT Press, Cambridge, MA, USA, v. 9, n. 8, p. 1735–1780, nov. 1997. ISSN 0899-7667. Disponível em: <http://dx.doi.org/10.1162/neco.1997.9.8.1735>.

KUMAR, A. C. S.; BHANDARKAR, S. M.; PRASAD, M. Depthnet: A recurrent neural network architecture for monocular depth prediction. *Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 396–404, 2018.

LUO, W.; SCHWING, A.; URTASUN, R. Efficient deep learning for stereo matching. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*. [S.l.]: IEEE Computer Society, 2016. v. 2016-January, p. 5695–5703.

NG, A. Deep learning specialization. In: . [s.n.], 2018. Disponível em: <https://www.coursera.org/specializations/deep-learning>.

REDMON, A. F. J. Yolo9000: Better, faster, stronger. *3D Digital Imaging and Modeling, International Conference on*, IEEE Computer Society, Los Alamitos, CA, USA, p. 6517–6525, 2017.

SCHARSTEIN, D. et al. High-resolution stereo datasets with subpixel-accurate ground truth. In: JIANG, X.; HORNEGGER, J.; KOCH, R. (Ed.). *Pattern Recognition*. Cham: Springer International Publishing, 2014. p. 31–42. ISBN 978-3-319-11752-2.

SCHARSTEIN, D.; SZELISKI, R. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, v. 47, p. 7–42, 04 2002.

SHI, X. et al. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214, 2015. Disponível em: <http://arxiv.org/abs/1506.04214>.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. Disponível em: <http://arxiv.org/abs/1409.1556>.

UHRIG, J. et al. Sparsity invariant cnns. In: *International Conference on 3D Vision (3DV)*. [S.l.: s.n.], 2017.

ZBONTAR, J.; LECUN, Y. Stereo matching by training a convolutional neural network to compare image patches. *CoRR*, abs/1510.05970, 2015. Disponível em: <http://arxiv.org/abs/1510.05970>.