PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA DE CIÊNCIAS EXATAS E DA COMPUTAÇÃO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANÁLISE COMPARATIVA ENTRE OS MODELOS CNN E LSTM PARA PREDIÇÃO DE FLUXO DO TRÁFEGO URBANO NA CIDADE DE RECIFE

RAFAEL GOMES DA SILVA

GOIÂNIA
2020

RAFAEL GOMES DA SILVA

ANÁLISE COMPARATIVA ENTRE OS MODELOS CNN E LSTM PARA PREDIÇÃO DE FLUXO DO TRÁFEGO URBANO NA CIDADE DE RECIFE

Trabalho de Conclusão de Curso apresentado à Escola de Ciências Exatas e da Computação, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

Orientador(a):
Prof. Me. Anibal Santos Jukemura

GOIÂNIA
2020

RAFAEL GOMES DA SILVA

ANÁLISE COMPARATIVA ENTRE OS MODELOS CNN E LSTM PARA PREDIÇÃO DE FLUXO DO TRÁFEGO URBANO NA CIDADE DE RECIFE

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharel em Ciência da Computação, e aprovado em sua forma final pela Escola de Ciências Exatas e da Computação, da Pontifícia Universidade Católica de Goiás, em ____/____/_____.

Profa. Ma. Ludmilla Reis Pinheiro dos Santos
Coordenador(a) de Trabalho de Conclusão de Curso

Banca Examinadora:

Orientador(a): Prof. Me. Anibal Santos Jukemura

Prof. Dr. Gustavo Siqueira Vinhal

Prof. Dr. Sibelius Lellis Vieira

GOIÂNIA
2020

Aos meus pais e à minha irmã pelo apoio.

AGRADECIMENTOS

Primeiramente, agradeço à ciência, por ser a fonte de todo o conhecimento humano, e por proporcionar as ferramentas necessárias para a solução dos diversos problemas inerentes à vida.

À minha família, ao meu pai, Edson Gomes Alves, à minha mãe, Evanice da Silva Alves, e à minha irmã, Rafaela Cristina Gomes da Silva, pelo apoio, o incentivo, e a compreensão.

Ao meu orientador, Anibal Santos Jukemura, pelas orientações, os ensinamentos, a paciência, e a confiança depositada.

Aos professores que contribuíram para a minha formação acadêmica. Especialmente, ao professor Gustavo Siqueira Vinhal e ao professor Sibelius Lellis Vieira, que fizeram parte da banca examinadora deste trabalho.

E por fim, a todas as pessoas que direta ou indiretamente contribuíram tanto para a minha formação acadêmica como para a realização deste trabalho.

“ *Predicting the future isn’t magic,*
it’s artificial intelligence. ”
Dave Waters

RESUMO

Diante do aumento na quantidade de dados disponíveis para o tráfego urbano, e das consequências causadas por congestionamentos, como o aumento no tempo de espera dos motoristas e o aumento na emissão de CO_2 na atmosfera, este trabalho se propôs a realizar uma análise comparativa entre os modelos de *Deep Learning* CNN e LSTM para demonstrar qual modelo melhor se adapta ao problema de predição de fluxo do tráfego urbano na cidade de Recife. Utilizando técnicas de pré-processamento dos dados para otimizar os resultados, e adotando estruturas para os modelos com base nos trabalhos relacionados, os dados foram divididos em subconjuntos para a análise de cenários diferentes. Apesar do modelo CNN ter apresentado um limite no valor máximo de predição, e do modelo LSTM não ter conseguido realizar predições para um dos subconjuntos, os resultados, em geral, indicaram que o modelo LSTM possui uma performance melhor do que o modelo CNN.

Palavras-chave: Análise preditiva. Fluxo de tráfego urbano. *Deep Learning*. CNN. LSTM.

ABSTRACT

In view of the increase in the amount of data available for urban traffic, and of the consequences caused by congestion, such as the increase in waiting times for drivers and the increase in the emission of CO_2 in the atmosphere, this work proposed to carry out a comparative analysis between the Deep Learning models CNN and LSTM to demonstrate which model best fits the urban traffic flow prediction problem in the city of Recife. Using data pre-processing techniques to optimize the results, and adopting structures for the models based on the related works, the data were divided into subsets for the analysis of different scenarios. Although the CNN model presented a limit on the maximum prediction value, and the LSTM model was unable to make predictions for one of the subsets, the results, in general, indicated that the LSTM model has a better performance than the CNN model.

Keywords: Predictive analysis. Urban traffic flow. Deep Learning. CNN. LSTM.

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplos de *underfitting* e *overfitting*, em comparação com o ajuste (ou *fit*) ideal. 19
Figura 2 – Exemplo de um ponto de parada ideal utilizando *early stopping*, com relação às taxas de erro de treinamento e validação ao longo de um treinamento. 20
Figura 3 – Diagrama de Euler mostrando a relação entre *Deep Learning* e IA. .. 21
Figura 4 – Modelo matemático de um neurônio artificial. 22
Figura 5 – Diferença entre uma (a) SLP e uma (b) MLP. 23
Figura 6 – Diferença entre uma (a) FFNN e uma (b) RNN. 24
Figura 7 – Exemplo do processo do SGD. 25
Figura 8 – Gráficos das funções de ativação ReLU e TanH. 25
Figura 9 – Exemplo de *dropout*. 26
Figura 10 – Estrutura típica de uma CNN. 27
Figura 11 – Exemplo de operação de convolução com um *kernel* de tamanho 2×2 e *stride* $= 1$. 27
Figura 12 – Exemplo de aplicação de *padding* para manter o tamanho da matriz de entrada. 28
Figura 13 – Exemplo de operação de *pooling* de tamanho 2×2, do tipo *max pooling*. 29
Figura 14 – Estrutura típica de uma LSTM, dividida entre a (a) visão geral e a (b) visão da célula LSTM. 30
Figura 15 – Diferença entre um (a) pseudocódigo e (b) Python para realizar a soma de todos os números pares em um intervalo de 1 a 10. 36
Figura 16 – Exemplo de multiplicação entre dois vetores (a) sem NumPy e (b) com NumPy. 37
Figura 17 – Exemplos de visualizações criadas com Matplotlib. 38
Figura 18 – Localizações dos pontos de observação no mapa. 43
Figura 19 – Porcentagem de amostras ausentes por mês. 43
Figura 20 – Porcentagem de amostras ausentes por subconjunto. 45
Figura 21 – Exemplo de uma matriz espaço-temporal em forma de imagem para um único dia. 46
Figura 22 – Mapa de correlação espacial antes e depois da reorganização dos pontos de observação. 47
Figura 23 – Quantidade de amostras por subconjunto. 48
Figura 24 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto A, para o modelo CNN. 56

Figura 25 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto B, para o modelo CNN. 56
Figura 26 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto C, para o modelo CNN. 57
Figura 27 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto D, para o modelo CNN. 57
Figura 28 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto A, para o modelo LSTM. 59
Figura 29 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto B, para o modelo LSTM. 59
Figura 30 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto C, para o modelo LSTM. 60
Figura 31 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto D, para o modelo LSTM. 60

LISTA DE TABELAS

Tabela 1 – Resultados obtidos por Ma et al. (2017). 32
Tabela 2 – Resultados obtidos por Zhang et al. (2019). 33
Tabela 3 – Resultados obtidos por Zhao et al. (2017). 34
Tabela 4 – Resultados obtidos por Luo et al. (2019). 35
Tabela 5 – Informações disponíveis no conjunto de dados. 41
Tabela 6 – Hiperparâmetros do modelo CNN. 50
Tabela 7 – Hiperparâmetros do modelo LSTM. 53
Tabela 8 – Resultados do modelo CNN. 55
Tabela 9 – Diferença de tempo entre o treinamento e o teste para o modelo CNN, em segundos. 55
Tabela 10 – Resultados do modelo LSTM. 58
Tabela 11 – Diferença de tempo entre o treinamento e o teste para o modelo LSTM, em segundos. 58

LISTA DE ABREVIATURAS E SIGLAS

CO_2	Dióxido de carbono
Adam	*Adaptive Moment Estimation*
API	*Application Programming Interface*
ARIMA	*Autoregressive Integrated Moving Average*
BN	*Bayesian Network*
CNN	*Convolutional Neural Network*
CPU	*Central Processing Unit*
CSV	*Comma-Separated Values*
CTTU	Autarquia de Trânsito e Transporte Urbano de Recife
CUDA	*Compute Unified Device Architecture*
FFNN	*Feed-Forward Neural Network*
GPU	*Graphics Processing Unit*
IA	Inteligência Artificial
IEA	*International Energy Agency*
ITS	*Intelligent Transportation System*
KM	*Kalman Filtering*
KNN	*K-Nearest Neighbors*
LSTM	*Long Short-Term Memory*
MAE	*Mean Absolute Error*
MAPE	*Mean Absolute Percentage Error*
MLP	*Multiple-Layer Perceptron*
MRE	*Mean Relative Error*
MSE	*Mean Squared Error*
ODC	*Origin Destination Correlation*
ReLU	*Rectified Linear Unit*

RMSE	*Root Mean Square Error*
RNA	Rede Neural Artificial
RNN	*Recurrent Neural Network*
SGD	*Stochastic Gradient Descent*
SLP	*Single-Layer Perceptron*
STFSA	*Spatiotemporal Feature Selection Algorithm*
SVM	*Support Vector Machine*
TanH	Tangente Hiperbólica

SUMÁRIO

1 INTRODUÇÃO 15
1.1 Objetivo Principal 16
1.2 Objetivos Secundários 17
1.3 Justificativa 17
1.4 Organização do Texto 17

2 REVISÃO BIBLIOGRÁFICA 18
2.1 Inteligência Artificial 18
2.2 *Machine Learning* 18
2.3 *Deep Learning* 21
2.4 Rede Neural Artificial 22
2.5 *Convolutional Neural Network* 26
2.6 *Long Short-Term Memory* 29
2.7 Trabalhos Relacionados 31
2.8 Ambiente de Desenvolvimento 35
2.8.1 Google Colab 35
2.8.2 Python 36
2.8.2.1 NumPy 37
2.8.2.2 Pandas 37
2.8.2.3 Matplotlib 38
2.8.2.4 PyTorch 39

3 METODOLOGIA 40
3.1 Problema 40
3.2 Conjunto de Dados 40
3.3 Pré-processamento dos Dados 42
3.3.1 Limpeza dos Dados 42
3.3.2 Divisão de Subconjuntos 44
3.3.3 Preenchimento de Amostras Ausentes 44
3.3.4 Construção de Matrizes Espaço-Temporais 45
3.3.5 Reordenação dos Pontos de Observação 46
3.3.6 Treinamento, Validação e Teste 46
3.3.7 Normalização do Fluxo 47
3.4 *Convolutional Neural Network* 48
3.4.1 Estrutura 49
3.4.2 Hiperparâmetros 50
3.4.3 Função de Ativação 50

3.4.4 Função de Perda 51
3.4.5 Algoritmo de Otimização 51
3.5 *Long Short-Term Memory* **51**
3.5.1 Estrutura 51
3.5.2 Hiperparâmetros 53
3.5.3 Função de Ativação 53
3.6 Métricas de Erro **54**

4 RESULTADOS 55
4.1 *Convolutional Neural Network* **55**
4.2 *Long Short-Term Memory* **58**

5 CONCLUSÃO 61

6 TRABALHOS FUTUROS 63

REFERÊNCIAS 64

1 INTRODUÇÃO

Prever fluxo de tráfego urbano com acurácia e rapidez é uma tarefa muito importante para motoristas, empresas, e agências governamentais (LV et al., 2015). A partir dessa informação, motoristas podem tomar decisões para evitar congestionamentos, empresas podem otimizar serviços de entrega, e agências governamentais podem melhorar o fluxo através de alterações em semáforos ou vias. Além disso, congestionamentos causam diversos problemas nas cidades, dentre eles o aumento no tempo de viagem dos motoristas e o aumento na emissão de dióxido de carbono (CO_2) na atmosfera. Um estudo lançado em 2019 pela Agência Internacional de Energia (IEA – *International Energy Agency*) mostra que, em 2017, o transporte rodoviário foi responsável por cerca de 18% da emissão de CO_2 no mundo (IEA, 2019). Portanto, implementar medidas de mitigação para o tráfego urbano torna-se relevante.

O problema de predição de fluxo do tráfego urbano consiste em prever, a partir de dados históricos, a quantidade de veículos passando por um determinado ponto de observação em um determinado intervalo de tempo. De acordo com Wang et al. (2019), esse problema pode ser classificado em três tipos, dependendo do intervalo de tempo definido:

- **curto prazo:** de 5 a 30 minutos;
- **médio prazo:** de 30 minutos a 1 hora;
- **longo prazo:** acima de 1 hora.

Este trabalho aborda um problema de curto prazo, pois define-se um intervalo de tempo de 15 minutos.

Porém, conforme aponta Lv et al. (2015), tal problema não é trivial, devido aos diversos fatores que influenciam o fluxo do tráfego, como os fatores espaciais e temporais. Do ponto de vista espacial, o fluxo em uma via afeta diretamente o fluxo em vias vizinhas, ou até mesmo em vias um pouco mais distantes. Já do ponto de vista temporal, o fluxo é afetado tanto pelo fluxo atual como pelo fluxo recente, além de apresentar comportamentos diferentes dependendo do horário - em horários de pico, espera-se um fluxo maior.

Historicamente, dois tipos de modelos têm sido aplicados a este problema: paramétricos e não-paramétricos (LV et al., 2015).

Os modelos paramétricos utilizam análise de séries temporais para realizar as predições. Alguns exemplos mais utilizados são o Auto-Regressivo Integrado de Médias Móveis (ARIMA – *Autoregressive Integrated Moving Average*) com suas extensões, e o Filtro de Kalman (KM – *Kalman Filtering*). O problema desses modelos

é que, como eles consideram apenas correlações temporais, eles não conseguem lidar muito bem com redes de tráfego complexas (LV et al., 2015).

Por isso, surgiu-se a necessidade de utilizar modelos não-paramétricos. Esses modelos fazem parte da área de Inteligência Artificial (IA), e incluem tanto os algoritmos estatísticos e as Redes Neurais Artificiais (RNAs) simples do Aprendizado de Máquina (ou *Machine Learning*) como as RNAs profundas do Aprendizado de Máquina Profundo (ou *Deep Learning*).

Exemplos de modelos de *Machine Learning* mais utilizados são o K-Vizinhos Mais Próximos (KNN – *K-Nearest Neighbors*), a Máquina de Vetores de Suporte (SVM – *Support Vector Machine*), e as Redes Bayesianas (BNs – *Bayesian Networks*). Apesar de apresentarem uma acurácia superior aos modelos paramétricos, esses modelos não conseguem descrever a natureza estocástica e não-linear do fluxo do tráfego, demoram para treinar e consomem muitos recursos computacionais, o que os torna inviáveis para aplicações com uma grande quantidade de dados (LV et al., 2015).

Para suprir esses problemas, RNAs passaram a ser utilizadas. As RNAs simples apresentam uma boa capacidade para aprendizagem, generalização e predição, com uma acurácia superior e um tempo de treinamento reduzido, mas ainda não são capazes de extrair características relevantes de redes complexas com uma grande quantidade de dados (WANG et al., 2019). Com o aumento na quantidade de dados coletados pelos Sistemas Inteligentes de Transporte (ITSs – *Intelligent Transportation Systems*), a pesquisa na área de *Deep Learning* se intensificou, e mostrou-se que as RNAs profundas conseguem lidar melhor com o problema e produzem resultados satisfatórios (LV et al., 2015).

Nos últimos anos, vários trabalhos têm sido feitos nesta área, com diversos modelos de *Deep Learning* diferentes, mas Lv et al. (2015) sugere que não existe um modelo que se sobressai em todas as situações, devido às características específicas de cada conjunto de dados e os fatores externos. Um estudo feito por Wang et al. (2019) mostra que a Rede Neural Convolucional (CNN – *Convolutional Neural Network*) e a Rede Neural com Memória de Longo Prazo (LSTM – *Long Short-Term Memory*) apresentam a melhor performance.

Dessa forma, este trabalho pretende comparar modelos dessas duas redes, aplicados a um conjunto de dados específico, para identificar qual modelo se sobressai, através de métricas adequadas.

1.1 Objetivo Principal

O objetivo principal deste trabalho é realizar uma análise comparativa entre os modelos de *Deep Learning* CNN e LSTM para demonstrar qual modelo melhor se

adapta ao problema de predição de fluxo do tráfego urbano na cidade de Recife.

1.2 Objetivos Secundários

Em decorrência do objetivo principal, é possível identificar e tratar os seguintes objetivos secundários:

- aplicação de técnicas de pré-processamento dos dados;
- implementação dos modelos de predição supracitados;
- aplicação de técnicas para treinamento dos modelos;
- aplicação de técnicas para teste dos modelos;
- análise dos resultados através de métricas.

1.3 Justificativa

Diante do exposto, e visando minimizar o tempo de viagem dos motoristas e a emissão de CO_2 na atmosfera, justifica-se a necessidade de comparar os modelos de *Deep Learning* CNN e LSTM, aplicados a um conjunto de dados específico, para identificar qual modelo melhor se adapta ao problema de predição de fluxo do tráfego urbano.

Para este trabalho, foi selecionado um conjunto de dados da cidade de Recife, Pernambuco. Tal opção foi escolhida porque é o único conjunto de dados brasileiros relevante disponível publicamente, e existe uma escassez de trabalhos feitos com conjuntos de dados brasileiros nesta área. Como as características do fluxo do tráfego urbano variam de região para região, é interessante considerar uma região brasileira para estudo.

1.4 Organização do Texto

Para alcançar o objetivo principal, optou-se pela seguinte organização deste trabalho: o capítulo 2 apresenta a revisão bibliográfica, incluindo os conceitos básicos, os trabalhos relacionados e o ambiente de desenvolvimento; o capítulo 3 descreve a metodologia utilizada, incluindo as técnicas de pré-processamento dos dados, as estruturas dos modelos e as métricas de erro; o capítulo 4 descreve brevemente a implementação dos modelos e apresenta os resultados; o capítulo 5 traz a conclusão e as considerações finais; e o capítulo 6 oferece sugestões para trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta alguns conceitos básicos necessários para o entendimento do trabalho, tais como IA, *Machine Learning*, *Deep Learning*, RNA, CNN e LSTM, assim como os trabalhos relacionados usados como referência e um breve resumo do ambiente de desenvolvimento.

IA é a área geral que engloba todos os demais conceitos, e será definida a seguir.

2.1 Inteligência Artificial

De acordo com Russell e Norvig (2009), define-se IA com base em quatro categorias diferentes: sistemas que pensam como seres humanos, sistemas que pensam racionalmente, sistemas que agem como seres humanos, e sistemas que agem racionalmente.

Este trabalho aborda um sistema que age racionalmente. Em tais sistemas, um agente racional deve ser capaz de operar de forma autônoma, percebendo o seu ambiente e se adaptando a mudanças, de modo que o melhor resultado possível seja alcançado, através de modelos matemáticos bem definidos (RUSSELL; NORVIG, 2009).

Esse é o conceito que *Machine Learning* apresenta, e será descrito a seguir.

2.2 *Machine Learning*

Machine Learning é a capacidade de um agente racional de adquirir conhecimento identificando padrões em dados. Um conjunto de dados é uma coleção de exemplos, que, por sua vez, é uma coleção de características desses dados (GOODFELLOW; BENGIO; COURVILLE, 2016).

Segundo Russell e Norvig (2009), existem três tipos principais de aprendizado: não supervisionado, supervisionado e reforçado. No aprendizado não supervisionado, o agente é capaz de identificar padrões no conjunto de entrada sem ter um conjunto de saída como referência. Já no aprendizado supervisionado, o objetivo do agente é gerar uma função que mapeia, o mais próximo possível, o conjunto de entrada ao conjunto de saída. E no aprendizado reforçado, o agente aprende através de uma série de tentativas e erros, em um sistema de recompensas. O aprendizado abordado neste trabalho é do tipo supervisionado.

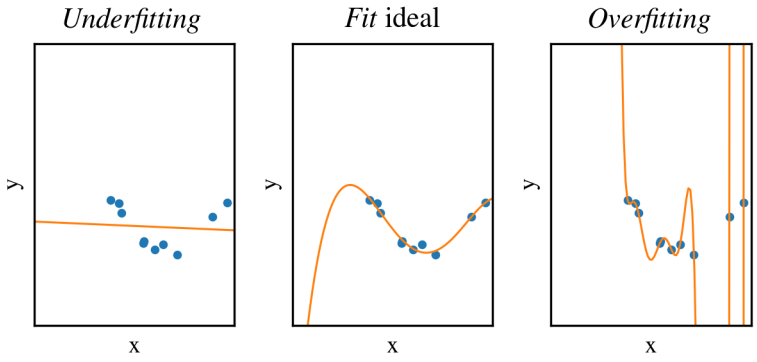
Em um aprendizado supervisionado, a função mapeadora é chamada de hipótese. Nesse caso, o objetivo é selecionar um modelo e uma técnica de otimização, de modo que o modelo define o domínio da hipótese e a técnica de otimização é

responsável por encontrar a melhor hipótese nesse domínio. O processo de otimização consiste em minimizar uma função de perda, e consequentemente a taxa de erro, durante o treinamento. A taxa de erro representa o quão longe a hipótese está da solução ideal (GOODFELLOW; BENGIO; COURVILLE, 2016).

Para verificar a performance da hipótese, divide-se o conjunto de dados em um subconjunto de treinamento e um subconjunto de teste. O treinamento ocorre através de várias iterações pelo subconjunto de treinamento, denominadas épocas. A cada época, o subconjunto de treinamento é utilizado para gerar uma nova hipótese e obter a taxa de erro. Ao final do treinamento, tem-se a hipótese final, que é, então, avaliada utilizando o subconjunto de teste (RUSSELL; NORVIG, 2009).

É importante ressaltar que uma taxa de erro baixa não necessariamente significa que a hipótese conseguirá generalizar bem para valores desconhecidos. Esse é um dos principais desafios em *Machine Learning*, que pode levar aos problemas de sub-ajuste (ou *underfitting*) e sobre-ajuste (ou *overfitting*). *Underfitting* acontece quando a solução é muito simples para um problema complexo, deixando a hipótese muito genérica. Já *overfitting* acontece quando a solução é muito complexa para um problema simples, deixando a hipótese muito específica (GOODFELLOW; BENGIO; COURVILLE, 2016). A Figura 1 mostra exemplos desses problemas.

Figura 1 – Exemplos de *underfitting* e *overfitting*, em comparação com o ajuste (ou *fit*) ideal.



Fonte: Elaborado pelo autor, com base em Scikit-learn (2020).

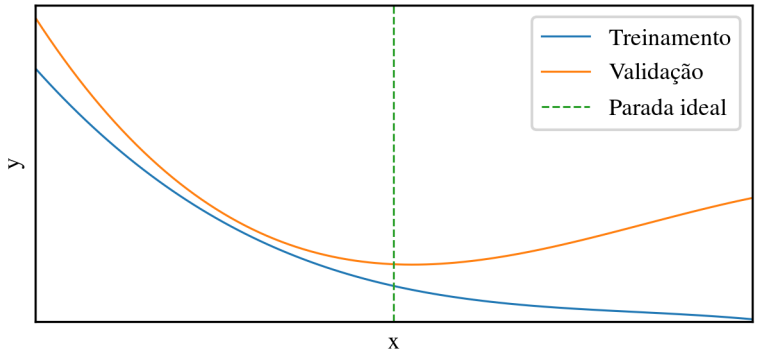
Chama-se de regularização qualquer modificação feita na hipótese para reduzir o erro de generalização (GOODFELLOW; BENGIO; COURVILLE, 2016). Uma técnica de regularização eficiente, segundo Russell e Norvig (2009), é escolher um método adequado para a divisão do conjunto de dados. O método mencionado anteriormente, onde o conjunto é dividido em dois subconjuntos, é chamado de Validação Cruzada por *Holdout*, e não é muito eficiente para alguns casos, pois não utiliza o conjunto inteiro para o treinamento. Já o método de Validação Cruzada por *K-Fold* resolve esse problema, pois divide o conjunto em *k* subconjuntos e realiza *k* iterações, onde, para

cada iteração, um subconjunto diferente é utilizado para o teste e os demais $k - 1$ subconjuntos são utilizados para o treinamento, sendo que a acurácia é a média de acurácias das iterações.

Outra técnica relevante é dividir o subconjunto de treinamento em mais dois subconjuntos, um de treinamento e outro de validação. Dessa forma, a hipótese pode ser continuamente avaliada a cada época, ao invés de ser avaliada somente no final do treinamento (RUSSELL; NORVIG, 2009). Tipicamente, utiliza-se uma proporção de 80% dos dados para treinamento e 20% para validação (GOODFELLOW; BENGIO; COURVILLE, 2016).

Para reduzir o erro de generalização, também pode ser empregada uma técnica chamada de parada antecipada (ou *early stopping*), que consiste em salvar a hipótese toda vez que a taxa de erro de treinamento for reduzida e encerrar o treinamento assim que a taxa de erro de validação começar a crescer. Dessa forma, o treinamento pode ser finalizado mais rapidamente, obtendo-se a melhor hipótese ao invés da hipótese gerada pela última época (GOODFELLOW; BENGIO; COURVILLE, 2016). Um exemplo de ponto de parada ideal é demonstrado na Figura 2.

Figura 2 – Exemplo de um ponto de parada ideal utilizando *early stopping*, com relação às taxas de erro de treinamento e validação ao longo de um treinamento. Do lado esquerdo do ponto, a hipótese tende a *underfitting*, e do lado direto a *overfitting*.



Fonte: Elaborado pelo autor, com base em Goodfellow, Bengio e Courville (2016).

Além disso, os hiperparâmetros do modelo também podem influenciar no resultado. Hiperparâmetros em *Machine Learning* são as diversas configurações que podem ser feitas em um modelo para alterar seu comportamento (GOODFELLOW; BENGIO; COURVILLE, 2016).

Russell e Norvig (2009) destaca que um problema em *Machine Learning* pode ser de dois tipos: classificação ou regressão. Em um problema de classificação, a saída é um conjunto finito de valores (por exemplo, “tem congestionamento” ou “não tem congestionamento”), enquanto a saída de um problema de regressão é um

único valor numérico (por exemplo, “grau de congestionamento”). Como a saída do problema abordado neste trabalho é a quantidade de veículos, tem-se um problema de regressão.

Deep Learning é uma extensão de *Machine Learning*, e será descrito a seguir.

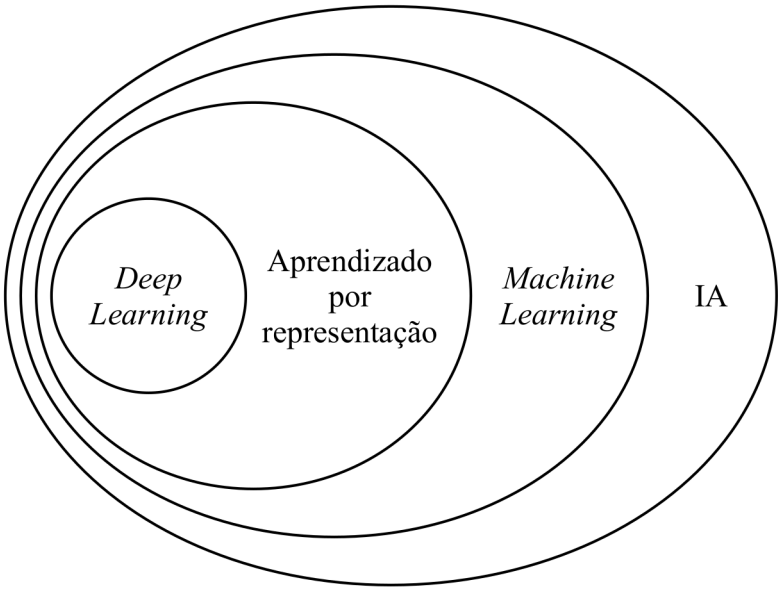
2.3 *Deep Learning*

Em *Machine Learning*, geralmente, as características dos dados devem ser fornecidas manualmente, pois o modelo não é capaz de extraí-las. Quando deseja-se que um modelo extraia essas características de forma autônoma, tem-se aprendizado por representação. Esse aprendizado consiste em fornecer uma grande quantidade de exemplos para que o modelo identifique padrões relevantes. *Deep Learning* é um tipo de aprendizado por representação capaz de solucionar problemas complexos, por oferecer um certo nível de abstração: uma representação complexa é reduzida a várias representações simples (GOODFELLOW; BENGIO; COURVILLE, 2016).

O autor ainda aponta que, por causa dessa capacidade, *Deep Learning* vem demonstrando ótimos resultados em diversas áreas, e alcançando performances bem próximas, ou até mesmo superiores, ao ser humano.

Portanto, *Deep Learning* é um tipo de aprendizado por representação, que, por sua vez, é um tipo de *Machine Learning*, que, por sua vez, está contido em IA. Essa relação pode ser visualizada no diagrama da Figura 3.

Figura 3 – Diagrama de Euler mostrando a relação entre *Deep Learning* e IA.



Fonte: Elaborado pelo autor, com base em Goodfellow, Bengio e Courville (2016).

Uma das diferenças entre *Machine Learning* e *Deep Learning* está na profundidade do modelo e na quantidade de exemplos fornecidos. Goodfellow,

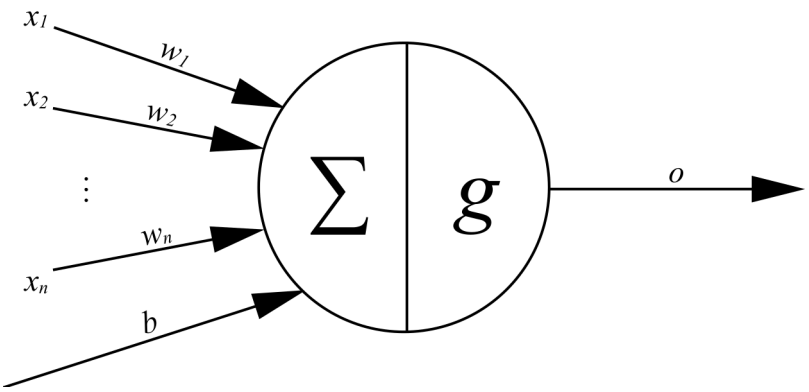
Bengio e Courville (2016) sugere pelo menos 5 mil exemplos para uma performance satisfatória em *Deep Learning*, e pelo menos 10 milhões de exemplos para uma performance próxima à humana.

Deep Learning tem como base a RNA, que é detalhada na próxima seção.

2.4 Rede Neural Artificial

Uma RNA é uma rede composta por neurônios artificiais, também chamados de perceptrons. Esses neurônios artificiais foram modelados com base nos neurônios biológicos do ser humano, e são conectados através de vínculos, sendo que um neurônio pode ser ativado ou desativado por uma função de ativação. Cada vínculo tem um peso associado que determina a intensidade da conexão (RUSSELL; NORVIG, 2009). De certa forma, uma RNA pode ser vista como um grafo, onde os neurônios são os vertíces e os vínculos entre os neurônios são arestas ponderadas. A Figura 4 mostra o modelo matemático de um neurônio artificial.

Figura 4 – Modelo matemático de um neurônio artificial.



Fonte: Elaborado pelo autor, com base em Russell e Norvig (2009).

A partir da Figura 4, um neurônio artificial pode ser representado, matematicamente, da seguinte forma:

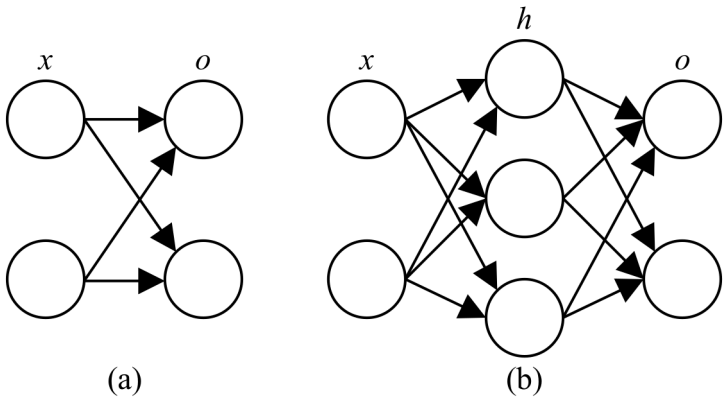
$$o = g(Wx + b) \tag{2.1}$$

onde x é o vetor de entrada de tamanho $n \times 1$, W é o vetor de pesos de tamanho $1 \times n$, b é uma constante denominada *bias*, g é uma função de ativação, e o é a saída do neurônio. O vetor de pesos determina o quanto cada elemento do vetor de entrada influencia no resultado, e o *bias* serve para garantir que a função não passe pela origem. Os pesos são geralmente inicializados com valores aleatórios pequenos, enquanto o *bias* é geralmente inicializado com 0 ou um valor positivo pequeno (GOODFELLOW; BENGIO; COURVILLE, 2016).

RNAs são geralmente estruturadas em camadas, sendo que a principal diferença entre uma RNA simples e uma RNA profunda é o número de camadas e o número de unidades por camada que a rede possui (GOODFELLOW; BENGIO; COURVILLE, 2016).

A RNA mais simples é a Perceptron de Única Camada (SLP – *Single-Layer Perceptron*), que possui apenas uma camada de entrada e uma camada de saída (RUSSELL; NORVIG, 2009). Já a Perceptron de Múltiplas Camadas (MLP – *Multiple-Layer Perceptron*) é um pouco mais complexa, possuindo uma ou mais camadas ocultas entre a camada de entrada e a camada de saída. Quando trata-se de *Deep Learning*, é exatamente nessas camadas ocultas que o modelo aprende a extrair características dos dados (GOODFELLOW; BENGIO; COURVILLE, 2016). A Figura 5 exemplifica a diferença entre uma SLP e uma MLP.

Figura 5 – Diferença entre uma (a) SLP e uma (b) MLP. *x* representa a camada de entrada, *o* representa a camada de saída e *h* representa a camada oculta.

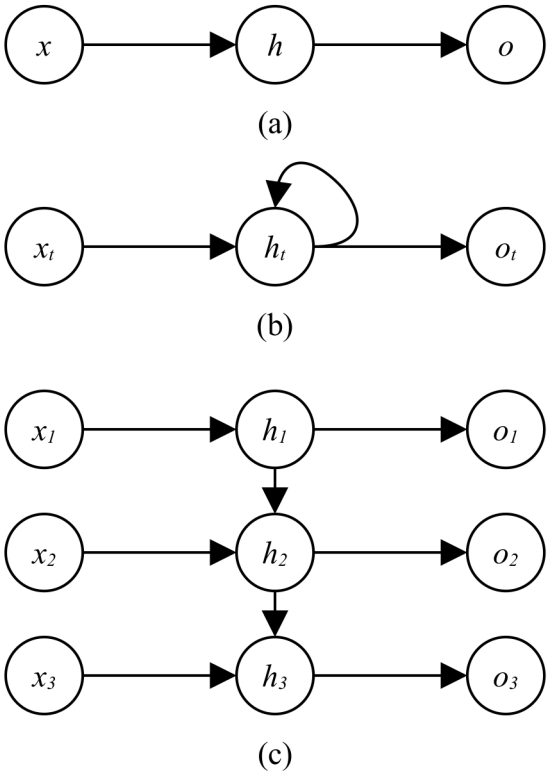


Fonte: Elaborado pelo autor, com base em Russell e Norvig (2009).

De acordo com Russell e Norvig (2009), existem dois tipos principais de RNA: Rede Neural *Feed-Forward* (FFNN – *Feed-Forward Neural Network*) e Rede Neural Recorrente (RNN – *Recurrent Neural Network*). A diferença entre os dois tipos é que na FFNN, uma unidade conecta somente com unidades posteriores, enquanto na RNN, uma unidade pode conectar com ela mesma, resultando em uma recorrência. Utilizando a analogia do grafo apresentada anteriormente, uma FFNN é similar a um grafo acíclico direcionado, enquanto uma RNN é similar a um grafo cíclico direcionado. A Figura 6 exemplifica a diferença entre uma FFNN e uma RNN.

Em uma RNA, o processo de otimização da hipótese se baseia na técnica de retro-propragação (ou *back-propagation*), onde, no final de cada época, calcula-se o gradiente da função de perda com respeito a cada peso da rede, partindo da camada de saída e propragando-se pelas camadas ocultas de trás para frente. Os gradientes calculados são usados por algoritmos de otimização para atualizar os pesos da rede (RUSSELL; NORVIG, 2009).

Figura 6 – Diferença entre uma (a) FFNN e uma (b) RNN. Utiliza-se a notação simplificada para a FFNN, onde uma unidade representa uma camada inteira. Na RNN, cada camada oculta é, na verdade, um estado oculto que serve como uma memória para a rede. (c) Ao “desdobrar” a RNN, é mais fácil visualizar que, a cada passo de tempo, o estado oculto é atualizado com base no estado anterior.

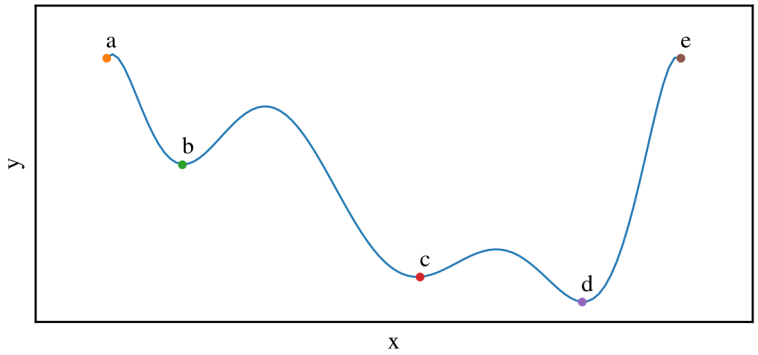


Fonte: Elaborado pelo autor, com base em Goodfellow, Bengio e Courville (2016).

Um dos algoritmos de otimização mais utilizados em *Deep Learning* é a Descida de Gradiente Estocástica (SGD – *Stochastic Gradient Descent*), que consiste em mover, em passos pequenos, na direção negativa do gradiente, para encontrar o ponto mínimo ideal. Esses passos pequenos são chamados de *mini-batches*, e consistem em treinar o subconjunto de treinamento em lotes de dados, ao invés de treinar dado por dado. O SGD utiliza uma taxa de aprendizagem que diminui ao decorrer do tempo, permitindo que passos relativamente grandes sejam dados no início e passos menores sejam dados no final, o que garante uma convergência mais rápida. Além disso, o SGD usa uma taxa de impulso (ou *momentum*) que controla a aceleração de cada passo com base nos passos anteriores, garantindo que o algoritmo não fique preso em um ponto mínimo local (GOODFELLOW; BENGIO; COURVILLE, 2016). A Figura 7 exemplifica o processo do SGD.

Quando se trata de funções de ativação, a Unidade Linear Retificada (ReLU – *Rectified Linear Unit*) é a mais utilizada para FFNNs, pois ela preserva somente valores positivos. Isso reduz a complexidade do cálculo dos gradientes na

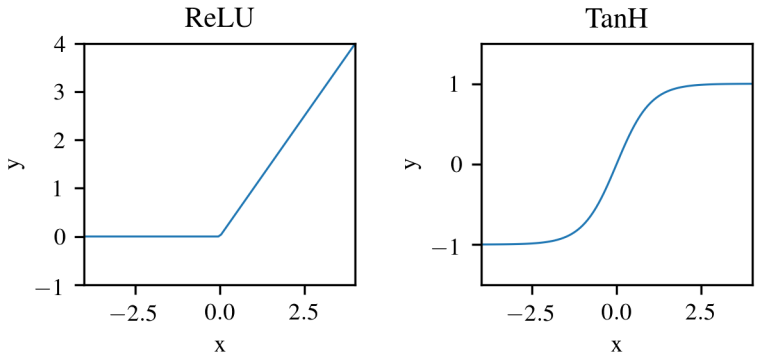
Figura 7 – Exemplo do processo do SGD. Partindo do ponto *a*, o objetivo é encontrar o ponto mínimo ideal, no caso *c* ou *d*, mas percebe-se que existe a possibilidade do algoritmo ficar preso no ponto *b*, por ser um ponto mínimo local. A taxa de aprendizagem e o *momentum* servem exatamente para resolver esse problema.



Fonte: Elaborado pelo autor, com base em Goodfellow, Bengio e Courville (2016).

rede, uma vez que a derivada da ReLU é sempre 1 para valores positivos e 0 para valores negativos, e resulta em um treinamento mais rápido. Já para RNNs, a Tangente Hiperbólica (TanH) é mais utilizada, por sempre gerar valores no intervalo $[-1, 1]$, fazendo com que os gradientes da rede se sustentem por um período de tempo maior (GOODFELLOW; BENGIO; COURVILLE, 2016). Gráficos dessas funções podem ser visualizados na Figura 8.

Figura 8 – Gráficos das funções de ativação ReLU e TanH.



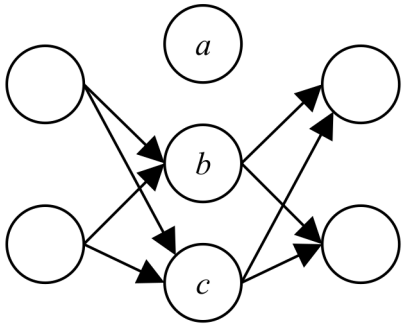
Fonte: Elaborado pelo autor, com base em Goodfellow, Bengio e Courville (2016).

Em relação a técnicas de regularização, L^1 e L^2 são comumente utilizadas em *Deep Learning* para penalizar pesos muito altos na rede, resultando em uma queda de peso (GOODFELLOW; BENGIO; COURVILLE, 2016).

Outra técnica de regularização comum em *Deep Learning* é chamada de *dropout*, e consiste em desativar certas unidades das camadas ocultas durante o

processo de treinamento, de modo que apenas o resto da rede seja treinado. As unidades a serem desativadas são escolhidos com base em uma porcentagem de probabilidade. Um método para desativar uma unidade é simplesmente garantir que sua saída seja 0. Essa técnica garante que a hipótese seja capaz de generalizar melhor, pois com algumas unidades ausentes, outras unidades precisam intervir e aprender as representações necessárias no lugar das unidades ausentes (GOODFELLOW; BENGIO; COURVILLE, 2016). A Figura 9 apresenta um exemplo de *dropout*.

Figura 9 – Exemplo de *dropout*. A unidade ***a*** é desativada, forçando as unidades *b* e *c* a intervirem em seu lugar.



Fonte: Elaborado pelo autor, com base em Goodfellow, Bengio e Courville (2016).

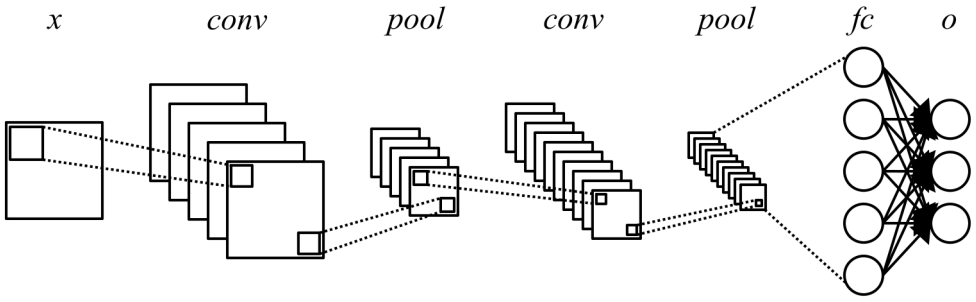
O entendimento do funcionamento do neurônio artificial, conforme descrito nesta seção, é fundamental para a essência das redes CNN e LSTM definidas a seguir.

2.5 *Convolutional Neural Network*

De forma resumida, uma CNN é uma FFNN onde as camadas ocultas são substituídas por camadas convolucionais. Essas camadas são localmente conectadas ao invés de serem totalmente conectadas (ou *fully connected*), permitindo que diferentes camadas se especializem em diferentes características. Esse tipo de rede é adequado para dados representados em formato bidimensional, como, por exemplo, imagens. Por isso, CNNs têm sido vastamente aplicadas em problemas de classificação e reconhecimento de imagens (GOODFELLOW; BENGIO; COURVILLE, 2016). A estrutura típica de uma CNN é apresentada na Figura 10.

A operação de convolução consiste em “passar” um filtro convolucional, chamado de *kernel*, de tamanho $k \times k$, sobre a matriz de entrada, para filtrar características indesejáveis ou amplificar características de interesse. A cada passo, delimita-se uma área da matriz de mesmo tamanho do *kernel*, obtendo-se o produto

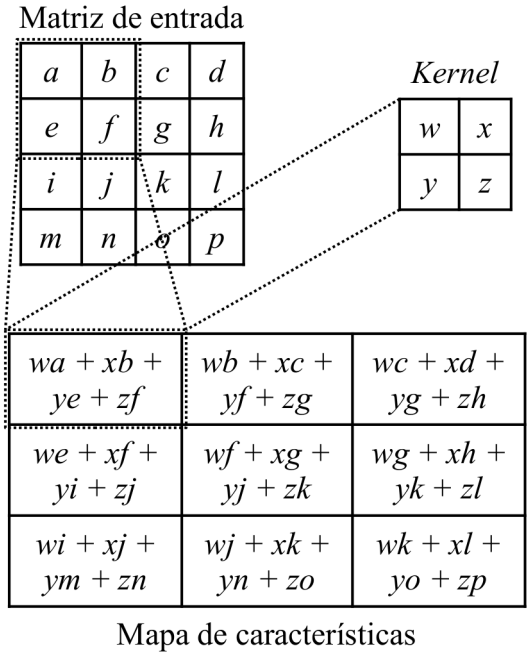
Figura 10 – Estrutura típica de uma CNN. *conv* representa uma operação de convolução, *pool* representa uma operação de *pooling* e *fc* representa uma camada *fully connected*. Percebe-se que, à medida que as operações de convolução e *pooling* são aplicadas, a matriz de entrada diminui em largura e altura, e aumenta em profundidade.



Fonte: Elaborado pelo autor, com base em Goodfellow, Bengio e Courville (2016).

escalar entre o *kernel* e essa área, que é adicionado em uma nova matriz, chamada de mapa de características. O número de linhas e colunas “puladas” pelo *kernel* a cada passo é chamado de *stride*. (GOODFELLOW; BENGIO; COURVILLE, 2016). Uma exemplificação desse procedimento pode ser encontrada na Figura 11.

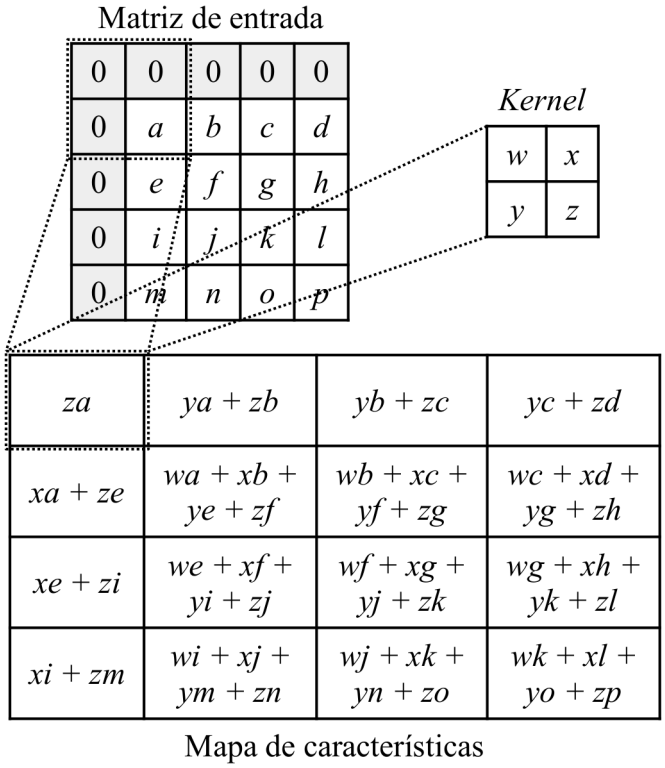
Figura 11 – Exemplo de operação de convolução com um *kernel* de tamanho 2×2 e *stride* $= 1$. A cada passo, o *kernel* “desliza” pela matriz de entrada, gerando um novo elemento no mapa de características.



Fonte: Elaborado pelo autor, com base em Goodfellow, Bengio e Courville (2016).

Goodfellow, Bengio e Courville (2016) ressalta que, conforme visualizado na Figura 11, a operação de convolução reduz o tamanho da matriz de entrada por $k - 1$ em cada dimensão, porque o filtro não excede as margens da matriz. Uma solução para esse problema é a aplicação de preenchimento (ou *padding*). *Padding* consiste em adicionar linhas e colunas à matriz, geralmente preenchidas com 0, para controlar o tamanho do mapa de características. A Figura 12 mostra um exemplo de aplicação de *padding*.

Figura 12 – Exemplo de aplicação de *padding* para manter o tamanho da matriz de entrada.

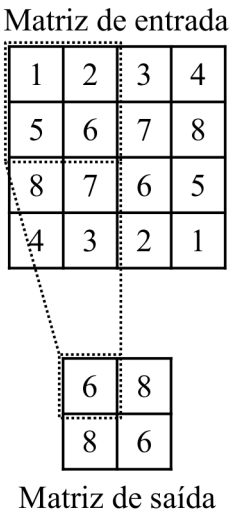


Fonte: Elaborado pelo autor, com base em Goodfellow, Bengio e Courville (2016).

Após a operação de convolução, é aplicada uma operação chamada de *pooling*. O procedimento dessa operação é similar ao da convolução, porém, ao invés de obter-se o produto escalar entre um filtro e a área delimitada na matriz de entrada, é obtido um elemento representativo dessa área, de modo que o mapa de características possa ser reduzido, obtendo-se apenas características relevantes. Dois tipos comuns de *pooling* são o *pooling* máximo (ou *max pooling*), que consiste em obter o maior elemento da área, e o *pooling* médio, que consiste em obter a média dos elementos da área (GOODFELLOW; BENGIO; COURVILLE, 2016). Uma exemplificação desse procedimento pode ser encontrada na Figura 13.

No domínio de imagens, os *kernels* são responsáveis por detecção de bordas e outros traços importantes, e o *pooling* garante que a posição ou o ângulo de um objeto

Figura 13 – Exemplo de operação de *pooling* de tamanho 2×2, do tipo *max pooling*. A cada passo, o maior elemento da área delimitada na matriz de entrada é adicionado à matriz de saída.



Fonte: Elaborado pelo autor, com base em Goodfellow, Bengio e Courville (2016).

não seja um fator significante para a identificação desse objeto (GOODFELLOW; BENGIO; COURVILLE, 2016).

Percebe-se que, por causa de sua estrutura, a CNN consegue extrair características espaciais de dados com mais facilidade. Já a LSTM utiliza uma abordagem diferente, e é definida a seguir.

2.6 *Long Short-Term Memory*

Uma RNN possui um estado oculto que age como uma memória para a rede, e é atualizado a cada passo de tempo, com base no estado oculto do passo anterior, sendo que os pesos desse estado são compartilhados ao decorrer do tempo. Isso permite que a RNN retenha informações recentes sobre dados (GOODFELLOW; BENGIO; COURVILLE, 2016).

Porém, Goodfellow, Bengio e Courville (2016) aponta que, à medida que o tempo aumenta, torna-se mais difícil reter informações antigas em uma RNN, pois, pelo fato dos pesos serem compartilhados, os gradientes da rede tendem a desaparecer, ou seja, tendem a 0. Esse problema é chamado de desaparecimento de gradiente (ou *vanishing gradient*), e é exatamente o que a LSTM surgiu para resolver.

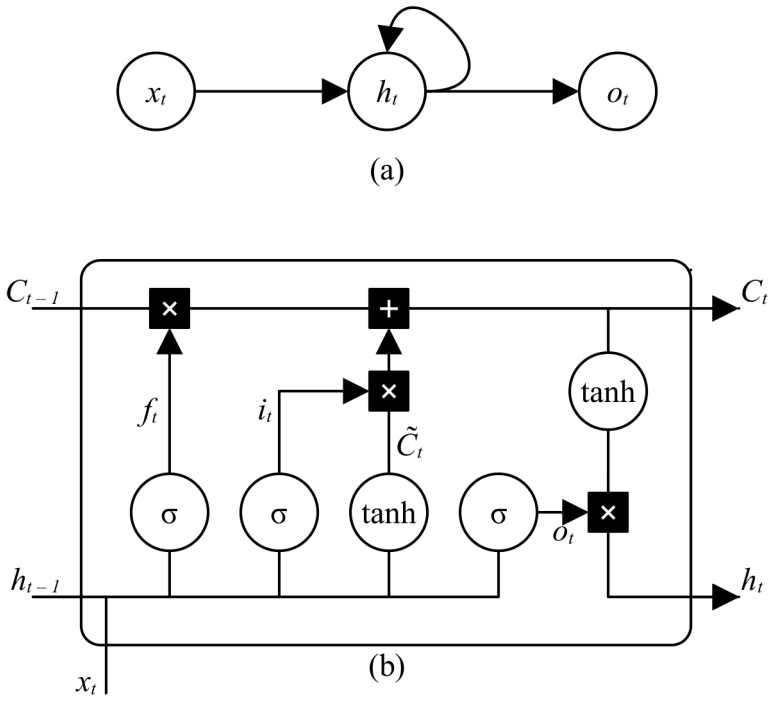
Uma LSTM é uma RNN que possui uma unidade oculta um pouco mais complexa, denominada de célula LSTM. Ela corrige o problema do *vanishing gradient* ao adicionar um estado interno nessa célula, em adição ao estado oculto externo, que

consegue armazenar informações por um período de tempo maior. Isso faz com que a LSTM seja capaz de aprender dependências a longo prazo com mais facilidade do que RNNs (GOODFELLOW; BENGIO; COURVILLE, 2016).

Por essa natureza, esse tipo de rede é adequado para dados sequenciais, como, por exemplo, séries temporais. Consequentemente, LSTMs são bastante utilizadas em problemas de reconhecimento de fala e tradução (GOODFELLOW; BENGIO; COURVILLE, 2016).

A estrutura típica de uma célula LSTM, conforme apresentada na Figura 14, é basicamente composta pelo estado de célula (C) e por três portões: portão de esquecimento (ou *forget gate*), portão de entrada (ou *input gate*) e portão de saída (ou *output gate*), representados, respectivamente, por f, i e o. Cada um desses portões é um neurônio artificial que é ativado pela função sigmoide (σ). Dessa forma, para um passo de tempo t, a célula recebe o estado de célula anterior (C_{t-1}), o estado oculto anterior (h_{t-1}) e o conjunto de entrada atual (x_t), e através de seus portões internos, calcula o novo estado de célula (C_t) e o novo estado oculto (h_t) a serem fornecidos para o próximo passo de tempo $t + 1$ (OLAH, 2015).

Figura 14 – Estrutura típica de uma LSTM, dividida entre a (a) visão geral e a (b) visão da célula LSTM.



Fonte: Elaborado pelo autor, com base em Olah (2015).

A função sigmoide gera apenas valores no intervalo $[0, 1]$, e é escolhida porque ao multiplicar um outro valor por um valor nesse intervalo, informações podem ser

“esquecidas” se o valor for próximo ou igual a 0, ou “lembradas” se o valor for próximo ou igual a 1 (OLAH, 2015).

O processo completo de execução da célula LSTM exemplificada na Figura 14, com base em Olah (2015), é detalhado a seguir:

1. Primeiramente, determina-se que informações serão esquecidas do estado de célula anterior. Isso é feito pelo *forget gate*, que combina o estado oculto anterior com o conjunto de entrada atual, gerando um vetor f_t, que é, então, multiplicado pelo estado de célula anterior, fazendo com que o estado de célula perca informações consideradas irrelevantes.

2. Em seguida, determina-se que informações novas serão lembradas pelo estado de célula. Um vetor i_t é gerado da mesma forma que o vetor f_t, porém, dessa vez, no *input gate*, para definir quais valores serão atualizados. Ao mesmo tempo, outro vetor $\widetilde{C}_t$ é gerado ao passar o estado oculto anterior e o conjunto de entrada atual pela função TanH, para efeito de regularização. O vetor $\widetilde{C}_t$ é considerado um candidato ao novo estado de célula, e é multiplicado pelo vetor i_t para que seja definido as informações do candidato a serem mantidas. O resultado dessa multiplicação é somado ao resultado do item 1, obtendo-se o novo estado de célula.

3. Com o novo estado de célula calculado, resta somente calcular o novo estado oculto. Novamente, um vetor o_t é gerado utilizando a mesma metodologia dos outros portões, dessa vez no *output gate*. O novo estado de célula é regularizado pela função TanH e, então, multiplicado pelo vetor o_t, finalmente gerando o novo estado oculto.

Conclui-se, então, que a CNN é mais adequada para a extração de características espaciais, enquanto a LSTM é mais adequada para a extração de características temporais. Porém, alguns trabalhos relacionados apresentam algumas modificações para que essas redes sejam capazes de extrair tanto características espaciais como temporais. Esses trabalhos são abordados em seguida.

2.7 Trabalhos Relacionados

Para fazer com que a CNN fosse capaz de extrair características temporais de dados, Ma et al. (2017) simplesmente alterou a representação dos dados. Foi desenvolvido um método que consiste em construir uma matriz, chamada de matriz espaço-temporal, a partir dos dados do tráfego, onde cada linha representa a velocidade do tráfego de um único ponto de observação em vários intervalos de tempo, e cada coluna representa a velocidade do tráfego de todos os pontos de observação

em um único intervalo de tempo. Esse método mostrou-se eficiente e se tornou a base para aplicações de CNN em problemas relacionados ao tráfego urbano.

Percebe-se que Ma et al. (2017) abordou o problema de predição de velocidade do tráfego, ao invés de fluxo. Porém, Wang et al. (2019) ressalta que os dois problemas são extremamente similares, e que metodologias aplicadas em um problema também podem ser aplicadas no outro.

Ma et al. (2017) aplicou esse método em um conjunto de dados de Pequim, China, abrangendo 37 dias. Definiu-se um intervalo de tempo de 2 minutos para as amostras, sendo que os 30 primeiros dias foram utilizados para o treinamento e os 7 últimos dias foram utilizados para o teste, resultando em 21.600 amostras para treinamento e 5.040 para teste. Foram testadas duas redes de tráfego diferentes, uma mais simples e outra mais complexa. Várias tarefas de predição foram testadas, com intervalos de tempo diferentes:

- **tarefa 1:** predição de 10 minutos com base nos últimos 30 minutos;
- **tarefa 2:** predição de 10 minutos com base nos últimos 40 minutos;
- **tarefa 3:** predição de 20 minutos com base nos últimos 30 minutos;
- **tarefa 4:** predição de 20 minutos com base nos últimos 40 minutos.

Para a avaliação do modelo, utilizou-se o Erro Quadrático Médio (MSE – *Mean Squared Error*). Os resultados obtidos são apresentados na Tabela 1.

Tabela 1 – Resultados obtidos por Ma et al. (2017).

Modelo	Rede	MSE			
		Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4
CNN	Simples	22,825	24,345	30,593	31,424
CNN	Complexa	27,163	28,479	37,987	38,816

Fonte: Elaborado pelo autor, com base em Ma et al. (2017).

Nota: Quanto menor o valor, melhor o resultado.

Zhang et al. (2019) aprimorou a metodologia de Ma et al. (2017) e desenvolveu um Algoritmo de Seleção de Características Espaço-Temporais (STFSA – *Spatiotemporal Feature Selection Algorithm*) para melhoria da acurácia, além de abordar um problema específico de fluxo. Utilizou-se dados de 50 dias de semana coletados em Seattle, Estados Unidos, com um intervalo de tempo de 5 minutos. Os 40 primeiros dias foram utilizados para o treinamento e os 10 últimos dias foram utilizados para o teste, resultando em 11.520 amostras para treinamento e 3.600 para teste. Tarefas de predição diferentes também foram abordadas:

- **tarefa 1:** predição de 5 minutos;
- **tarefa 2:** predição de 10 minutos;
- **tarefa 3:** predição de 15 minutos;
- **tarefa 4:** predição de 20 minutos.

Para a avaliação do modelo, utilizou-se o Erro Absoluto Médio (MAE – *Mean Absolute Error*) para a tarefa 1 e o Erro Absoluto Médio Percentual (MAPE – *Mean Absolute Percentage Error*) para todas as tarefas. Os resultados obtidos são apresentados na Tabela 2, comparando-se os resultados sem a utilização do STFSA e os resultados com a utilização do STFSA.

Tabela 2 – Resultados obtidos por Zhang et al. (2019).

Modelo	MAE	MAPE			
	Tarefa 1	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4
CNN	14,003	6,181	8,523	8,763	9,423
CNN + STFSA	13,336	5,763	7,429	7,953	8,243

Fonte: Elaborado pelo autor, com base em Zhang et al. (2019).

Nota: Quanto menor o valor, melhor o resultado.

Com base nesses resultados, demonstrou-se que o STFSA garantia melhores resultados para o modelo.

Já para fazer com que a LSTM fosse capaz de extrair características espaciais de dados, Zhao et al. (2017) desenvolveu um método similar ao de Ma et al. (2017), que também consiste em construir uma matriz a partir dos dados do tráfego, denominada de Correlação entre Origem e Destino (ODC – *Origin Destination Correlation*). Porém, ao invés de conter os dados puros do tráfego, ela contém os coeficientes de correlação entre os pontos de observação.

Além disso, Zhao et al. (2017) propôs uma estrutura bidimensional para a LSTM, com uma célula LSTM para cada ponto de observação, sendo que essas células se conectam, a cada passo de tempo, através de camadas *fully connected*.

Essa estrutura foi aplicada em um conjunto de dados de Pequim, China, abrangendo 6 meses. O conjunto contava com uma quantidade enorme de dados: 25.11 milhões, obtidos por mais de 500 pontos de observação, com um intervalo de tempo de 5 minutos. Dividiu-se o conjunto em dados dos 5 primeiros meses para o treinamento e dados do último mês para o teste. Esse trabalho também abordou tarefas de predição diferentes:

- **tarefa 1:** predição de 15 minutos;
- **tarefa 2:** predição de 30 minutos;
- **tarefa 3:** predição de 45 minutos;
- **tarefa 4:** predição de 60 minutos.

Para a avaliação do modelo, utilizou-se o Erro Relativo Médio (MRE – *Mean Relative Error*). Os resultados obtidos são apresentados na Tabela 3.

Tabela 3 – Resultados obtidos por Zhao et al. (2017).

Modelo	MRE			
	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4
LSTM	6,22	9,38	13,11	17,03

Fonte: Elaborado pelo autor, com base em Zhao et al. (2017).

Nota: Quanto menor o valor, melhor o resultado.

Porém, Luo et al. (2019) considerou a estrutura apresentada por Zhao et al. (2017) muito complexa, e identificou que ela produzia resultados instavéis. Foi proposta, então, uma metodologia melhor, utilizando-se KNN para identificar os pontos de observação mais próximos ao ponto de observação sendo testado, juntamente com estrutura multi-camadas para a LSTM. Essa estrutura consiste em treinar células LSTM para os pontos de observação separadamente, ponderando os resultados individuais com uma técnica chamada de expoente de classificação (ou *rank-exponent*) para obter as predições finais.

Luo et al. (2019) aplicou essa metodologia em dados de Minnesota, Estados Unidos. Optou-se por utilizar dados de apenas um dia específico por semana, para manter uma certa estabilidade na predição. Dessa forma, dados de 8 dias foram utilizados para o treinamento e dados de 1 dia foram utilizados para o teste, amostrados em intervalos de tempo de 5 minutos, resultando em 2.304 amostras para treinamento e 288 amostras para teste. Foi realizada uma única tarefa de predição de 5 minutos.

Para a avaliação do modelo, utilizou-se a Raiz Quadrada do Erro Médio (RMSE – *Root Mean Square Error*). Os resultados obtidos são apresentados na Tabela 4, comparando-se um modelo LSTM tradicional com o modelo KNN-LSTM proposto.

Pelo fato de cada trabalho utilizar intervalos de tempo e métricas de erro diferentes, não é possível comparar todos os modelos com facilidade. Além disso, não foi encontrado um trabalho relacionado que utiliza o conjunto de dados utilizado neste trabalho, o que dificulta ainda mais a comparação dos modelos. Porém, percebe-se

Tabela 4 – Resultados obtidos por Luo et al. (2019).

Modelo	RMSE
LSTM	2,284
KNN-LSTM	2,143

Fonte: Elaborado pelo autor, com base em Luo et al. (2019).

Nota: Quanto menor o valor, melhor o resultado.

que, independentemente das escolhas feitas, em todos os casos, a taxa de erro aumento à medida que o intervalo de tempo aumenta, com a maioria dos trabalhos optando por um intervalo de tempo de 5 minutos.

Por questões de simplicidade, este trabalho utiliza uma metodologia simplificada com base nas metodologias abordadas nesses trabalhos relacionados, que será descrita em detalhe no capítulo 3.

Para implementar esses modelos, é necessário um ambiente de desenvolvimento, que é brevemente descrito a seguir.

2.8 Ambiente de Desenvolvimento

Nesta seção, é feito um breve resumo do ambiente de desenvolvimento e das ferramentas utilizadas para a implementação dos modelos.

O ambiente de desenvolvimento utilizado neste trabalho é o Google Colab, descrito a seguir.

2.8.1 Google Colab

Google Colab é um serviço gratuito da Google que oferece um Jupyter Notebook na nuvem, sem necessidade de qualquer configuração, com acesso a Unidades de Processamento Gráfico (GPUs – *Graphics Processing Units*), e com fácil compartilhamento. É um ambiente designado especialmente para *Machine Learning* e análise de dados. Por ser um serviço gratuito, as GPUs fornecidas pelo Colab variam de acordo com a demanda, e incluem os modelos K80, T4, P4 e P100 da NVIDIA (COLABORATORY, 2020).

Ao utilizar esse serviço, não é preciso se preocupar com o *hardware* utilizado, pois essa parte é totalmente gerenciada pelo Colab.

Um Jupyter Notebook, por sua vez, é uma aplicação *web* que permite a criação de documentos que contêm código vivo, equações, visualizações e texto narrativo. Esses documentos possibilitam a execução de código iterativo, e são bastante utilizados em *Machine Learning*. Apesar de oferecer suporte para várias

linguagens de programação, Python é a linguagem padrão em um Jupyter Notebook (PROJECT JUPYTER, 2020).

Essa linguagem é apresentada a seguir.

2.8.2 Python

Python é uma linguagem de programação interpretada desenvolvida com foco em simplicidade. Esse é um dos motivos que faz com que ela seja a linguagem universalmente adotada em *Machine Learning*, pois ela permite que o desenvolvedor possa focar no problema em si, ao invés de ter que se preocupar com a linguagem. Python se assemelha bastante com pseudocódigo, o que a torna uma linguagem extremamente fácil de entender (GUPTA, 2019). Esse nível de legibilidade pode ser visualizado na Figura 15.

Figura 15 – Diferença entre um (a) pseudocódigo e (b) Python para realizar a soma de todos os números pares em um intervalo de 1 a 10. Em Python, o intervalo gerado por *range* não inclui o último valor do intervalo, por isso soma-se um a 10.

```
soma ← 0
para i de 1 até 10 passo 1 faça
    se i MOD 2 = 0 então
        soma ← soma + i
    fim se
fim para
escreva soma
```

(a)

```
soma = 0
for i in range(1, 10 + 1, 1):
    if i % 2 == 0:
        soma = soma + i
print(soma)
```

(b)

Fonte: Elaborado pelo autor.

Outro motivo pelo qual Python é bastante utilizado em *Machine Learning* é sua flexibilidade. Por ser uma linguagem interpretada, não é preciso esperar que o código seja compilado para executá-lo - os resultados podem ser visualizados imediatamente (GUPTA, 2019).

Além disso, Gupta (2019) destaca que Python possui um rico ecossistema de bibliotecas, que auxiliam em aplicações de *Machine Learning* e reduzem drasticamente o tempo de desenvolvimento. Uma biblioteca é basicamente um código Python que pode ser reutilizado com facilidade.

Algumas dessas bibliotecas utilizadas neste trabalho são: NumPy, Pandas, Matplotlib e PyTorch. Elas são detalhadas nos subtópicos a seguir.

2.8.2.1 NumPy

NumPy é uma biblioteca fundamental para a computação científica, por oferecer uma série de rotinas que simplificam a manipulação de vetores de diversas dimensões, através de uma estrutura base chamada de *ndarray*, facilitando operações matemáticas avançadas em grande quantidade de dados. Ela utiliza código C otimizado e pré-compilado para oferecer essas rotinas, e tem como base duas características principais: vetorização e transmissão. Vetorização consiste na ausência de laços de repetição explícitos e transmissão consiste na capacidade de executar operações elemento-por-elemento sem ter que explicitar tal comportamento (NUMPY, 2020). A Figura 16 demonstra os benefícios de utilizar NumPy.

Figura 16 – Exemplo de multiplicação entre dois vetores (a) sem NumPy e (b) com NumPy.

```
c = []
for i in range(len(a)):
    c.append(a[i] * b[i])
```

(a)

```
c = a * b
```

(b)

Fonte: Elaborado pelo autor.

2.8.2.2 Pandas

Pandas é uma biblioteca que disponibiliza uma estrutura rápida e flexível para trabalhar com dados relacionais, chamada de *DataFrame*, facilitando diversas operações feitas com esse tipo de dado, tais como: adição e remoção de linhas e colunas; alinhamento, agrupamento e preenchimento de dados; entre outras. Um *DataFrame* possui características semelhantes a uma *ndarray*, sendo que um

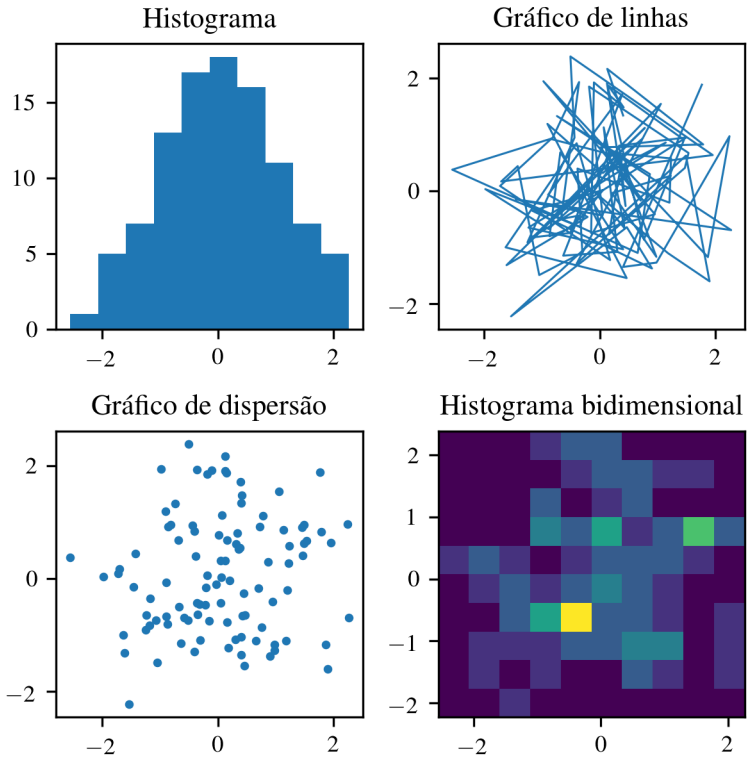
DataFrame pode ser facilmente convertido para uma *ndarray*, e vice-versa. Essa estrutura faz dessa biblioteca uma ferramenta poderosa para a etapa de limpeza e pré-processamento dos dados em *Machine Learning* (PANDAS, 2020).

Além disso, Pandas oferece uma interface para carregar um conjunto de dados, a partir de um arquivo no formato Valores Separados por Vírgula (CSV – *Comma-Separated Values*), para um *DataFrame*, e vice-versa (PANDAS, 2020).

2.8.2.3 Matplotlib

Matplotlib é uma biblioteca que facilita a criação de visualizações estáticas, animadas e iterativas em Python, sendo essencial para produzir gráficos (MATPLOTLIB, 2020). A Figura 17 apresenta alguns tipos de visualizações que podem ser criadas com essa biblioteca.

Figura 17 – Exemplos de visualizações criadas com Matplotlib.



Fonte: Elaborado pelo autor, com base em Matplotlib (2020).

Todos os gráficos apresentados neste trabalho são produzidos com Matplotlib.

2.8.2.4 PyTorch

PyTorch é a biblioteca principal utilizada neste trabalho, por fornecer as ferramentas necessárias para a implementação dos modelos. Ela é uma biblioteca de alta performance, com otimização para a computação científica, que oferece grande flexibilidade e velocidade para trabalhar com *Deep Learning*. PyTorch utiliza uma estrutura base, chamada de *tensor*, que é um vetor multidimensional similar a *ndarrays*, e permite a realização de operações matemáticas aceleradas em *hardware* dedicado. Essa estrutura também permite que um modelo seja facilmente movido entre as Unidades Centrais de Processamento (CPUs – *Central Processing Units*) e as GPUs (STEVENS; ANTIGA; VIEHMANN, 2020).

Essa computação acelerada é possível pelo fato de que PyTorch é escrita em C++ com a Arquitetura de Dispositivo de Computação Unificada (CUDA – *Compute Unified Device Architecture*), que é uma Interface de Programação de Aplicação (API – *Application Programming Interface*) desenvolvida pela NVIDIA que pode ser compilada com um alto nível de paralelismo em GPUs (STEVENS; ANTIGA; VIEHMANN, 2020).

Uma característica importante dos *tensors* é que eles são capazes de manter um histórico de operações feitas neles, o que facilita o cálculo de derivadas, um processo essencial em *Deep Learning* para a obtenção dos gradientes de uma rede durante a etapa de *back-propagation* (STEVENS; ANTIGA; VIEHMANN, 2020).

Todos esses aspectos fazem de PyTorch uma biblioteca ideal para aplicações de *Deep Learning*.

Com o ambiente de desenvolvimento definido, o próximo capítulo apresenta a metodologia abordada neste trabalho.

3 METODOLOGIA

Este capítulo descreve a metodologia abordada neste trabalho. Primeiramente, é especificado o problema. Em seguida, detalha-se o conjunto de dados utilizado. São definidas, então, as técnicas de pré-processamento dos dados aplicadas e as estruturas dos modelos CNN e LSTM utilizadas. Por último, define-se as métricas de erro selecionadas para a análise dos modelos.

Este trabalho propõe a investigação de um tema não trivial, com foco na pesquisa bibliográfica para sua melhor compreensão. É utilizada uma abordagem quantitativa com análise de amostra de dados, caracterizando-se, portanto, como uma pesquisa exploratória (MORETTI, 2017).

O problema é especificado a seguir.

3.1 Problema

Zhang et al. (2019) define o problema de predição de fluxo do tráfego urbano por:

$$X = x_{p,t} \quad | \quad p \in P, t \in \{1, 2, ..., T\} \tag{3.1}$$

onde P é o conjunto de pontos de observação, T é o intervalo de tempo atual, e $x_{p,t}$ indica a quantidade de veículos passando pelo ponto de observação p no intervalo de tempo t.

Dessa forma, no intervalo de tempo T, os dados históricos do fluxo de $(T-k)+1$ até T são utilizados para prever o fluxo em $T+1$, onde k é o número de intervalos utilizados para a predição.

Neste trabalho, utiliza-se os dados históricos de quatro intervalos (60 minutos) para prever o próximo intervalo (15 minutos), ou seja, $k = 4$. Portanto, para um ponto de observação qualquer $p = 2$, e um intervalo de tempo atual qualquer $T = 10$, utiliza-se os fluxos em $x_{2,7}$, $x_{2,8}$, $x_{2,9}$ e $x_{2,10}$ para prever o fluxo em $x_{2,11}$.

O conjunto de dados utilizado para resolver esse problema é detalhado em seguida.

3.2 Conjunto de Dados

Foi escolhido para este trabalho um conjunto de dados da cidade de Recife, Pernambuco, disponibilizado pela Autarquia de Trânsito e Transporte Urbano de Recife (CTTU) através do Portal de Dados Abertos da Cidade de Recife. Esse é o único conjunto de dados brasileiros relevante disponível publicamente - por isso sua escolha.

O conjunto de dados contém quantitativos de veículos por velocidade média, em intervalos de 15 minutos, capturados por 72 equipamentos (que incluem fotossensores e lombadas), de 365 dias (de janeiro a dezembro de 2019), em diversas vias da cidade. Inicialmente, são oferecidas 1.936.239 amostras (CTTU, 2020). Para obter o número total de amostras esperado, multiplica-se o número de amostras por dia, o número de dias e o número de equipamentos. Como cada dia tem 24 horas, cada hora tem 60 minutos, e as amostras estão divididas em intervalos de 15 minutos, o número de amostras por dia é $24 \cdot 60 \div 15 = 96$. O número total de amostras esperado é, então, $96 \cdot 365 \cdot 72 = 2.522.880$. Percebe-se, portanto, que o conjunto possui 23,25% de amostras ausentes.

Um conjunto de dados pode ser visualizado como uma tabela, onde as linhas contém as amostras dos dados e as colunas contém as informações das amostras. As informações disponíveis nesse conjunto podem ser visualizadas na Tabela 5. Além disso, também são disponibilizadas as coordenadas geográficas para a latitude e a longitude aonde cada equipamento está localizado.

Tabela 5 – Informações disponíveis no conjunto de dados.

Coluna	Informação
equipamento	O código do equipamento da via (e.g. “FS037REC” para fotossensor e “5972” para lombada).
velocidade_via	A velocidade máxima permitida na via (e.g. “60 km/h”).
faixa	A faixa da via lida pelo equipamento (de “1” a “4”).
ano	O ano de leitura pelo equipamento (“2019”).
mes	O mês de leitura pelo equipamento (de “1” a “12”).
data	A data de leitura pelo equipamento (e.g. “2019-01-01”).
hora	A hora de leitura pelo equipamento (de “0” a “23”).
minutos_intervalo	O intervalo de minutos considerado para leitura (e.g. “0-15”).
qtd_XaYkm	A contagem de veículos que passaram no equipamento descrito, no intervalo de data, hora e minutos, com velocidade média de *X* a *Y* km/h (e.g. “14”).

Fonte: Elaborado pelo autor.

O conjunto é pré-processado utilizando as técnicas definidas a seguir.

3.3 Pré-processamento dos Dados

A etapa de pré-processamento dos dados é a mais importante em qualquer problema de *Deep Learning*, pois ao alimentar um modelo com dados sujos, mal-formatados, mal-representados e incompletos, ele não terá utilidade quando aplicado no mundo real. Os passos de pré-processamento utilizados neste trabalho são detalhados nos próximos subtópicos.

3.3.1 Limpeza dos Dados

Como visto na Tabela 5, o conjunto possui algumas colunas redundantes, como “ano” e “mes”, uma vez que o conjunto contém dados somente de 2019 e o mês já está presente na coluna “data”. Essas colunas são, portanto, removidas.

Para manter consistência com a nomenclatura utilizada neste trabalho, as colunas “equipamento” e “minutos_intervalo” são renomeadas para “ponto” e “intervalo”, respectivamente.

Já outras colunas precisam ser combinadas. Combina-se as colunas “data”, “hora” e “intervalo” na coluna “data”. Além disso, todas as colunas referentes aos quantitativos de veículos por velocidade média são combinadas em uma única coluna denominada “fluxo”. Em seguida, como os dados estão divididos por faixa para cada ponto de observação, combina-se os dados de faixas diferentes, removendo a coluna “faixa”.

Após essas alterações, restam apenas as colunas “data”, “ponto” e “fluxo” no conjunto, que são as únicas informações necessárias para realizar as predições.

Detecta-se que vários pontos de observação possuem uma grande quantidade de amostras ausentes. Para reduzir essa quantidade, são removidos todos os pontos de observação com mais de 12,5% de amostras ausentes. Esse valor específico é escolhido por resultar em um número par de pontos de observação, o que facilita o processo de treinamento.

Com essa remoção, restam apenas 48 dos 72 pontos de observação. As localizações dos pontos restantes podem ser visualizadas na Figura 18, obtidas a partir das coordenadas disponibilizadas pelo conjunto de dados. Percebe-se que alguns pontos não estão muito próximos uns dos outros. Isso pode afetar a performance dos modelos, pois o ideal para esse tipo de problema é que os pontos estejam todos adjacentes, para facilitar a extração das características espaciais.

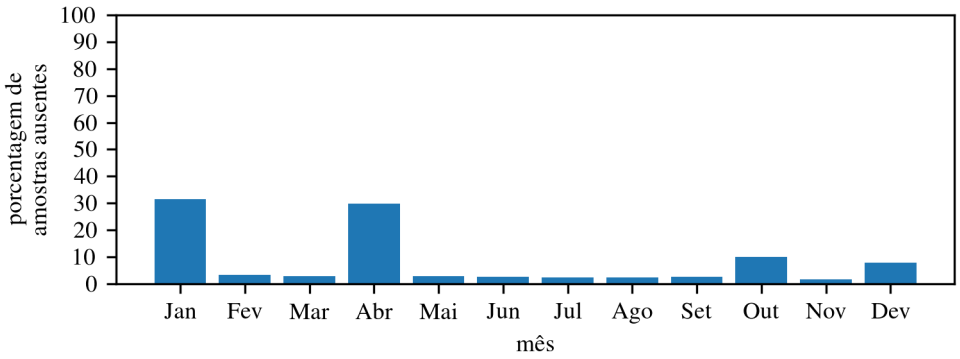
Finalmente, mantém-se apenas a sequência de meses com a menor porcentagem de amostras ausentes por mês, para reduzir ainda mais a quantidade de amostras ausentes, garantindo maior estabilidade. Com base na Figura 19, percebe-se que essa sequência vai de maio a dezembro. Portanto, as amostras dos

Figura 18 – Localizações dos pontos de observação no mapa.

Fonte: Elaborado pelo autor.

outros meses são removidas, e é utilizado especificamente um período de 236 dias (de 6 de maio a 27 de dezembro).

Figura 19 – Porcentagem de amostras ausentes por mês.



Fonte: Elaborado pelo autor.

No final desse passo, restam apenas 1.058.917 das 1.936.239 amostras iniciais. Porém, o conjunto se apresenta em um estado melhor, sem informações

irrelevantes, e com uma porcentagem menor de amostras ausentes. Como o número de equipamentos foi reduzido a 48 e o número de dias foi reduzido a 236, o novo número total de amostras esperado é $96 \cdot 236 \cdot 48 = 1.087.488$. Isso indica que, após a limpeza, o conjunto possui apenas 2,63% de amostras ausentes.

Com os dados limpos, é feita uma divisão no conjunto, que é detalhada em seguida.

3.3.2 Divisão de Subconjuntos

Para que seja possível avaliar cenários diferentes, o conjunto é dividido em quatro subconjuntos, cada um contendo amostras específicas:

- **subconjunto A:** amostras de todos os dias;
- **subconjunto B:** amostras de todos os dias, de 05:00 a 23:00;
- **subconjunto C:** amostras de dias de semana;
- **subconjunto D:** amostras de dias de semana, de 05:00 a 23:00.

O objetivo dessa divisão é avaliar o impacto de amostras fora dos horários e dias com maior movimentação nos resultados.

Para os subconjuntos A e C, espera-se 96 amostras por dia para cada ponto de observação, conforme calculado anteriormente, pois são utilizadas todas as 24 horas do dia. Já para os subconjuntos B e D, como são utilizadas apenas 18 horas do dia, espera-se $18 \cdot 60 \div 15 = 72$ amostras por dia.

Após essa divisão, é feito o preenchimento de amostras ausentes para cada subconjunto, conforme definido a seguir.

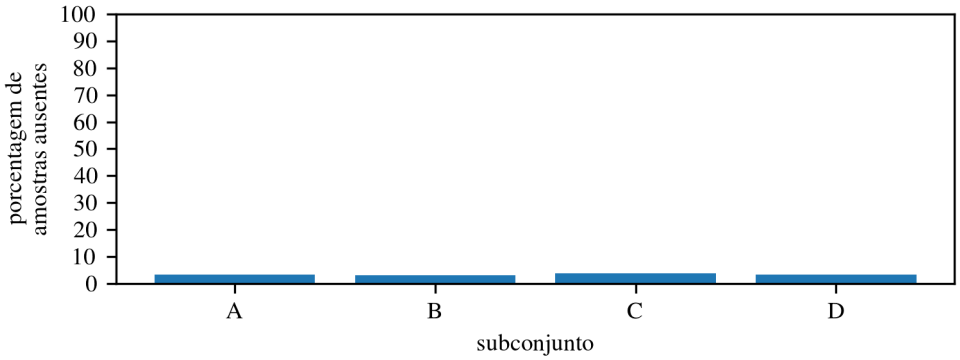
3.3.3 Preenchimento de Amostras Ausentes

Devido à limpeza feita anteriormente, os subconjuntos apresentam porcentagens extremamente baixas de amostras ausentes, o que remove a necessidade de utilizar uma técnica de preenchimento de dados mais complexa. Essas porcentagens podem ser visualizadas na Figura 20.

É aplicada, então, a técnica de interpolação linear, que consiste em espaçar os valores ausentes igualmente com base nos dois valores mais próximos.

Após as amostras ausentes serem preenchidas, são construídas as matrizes espaço-temporais, definidas em seguida.

Figura 20 – Porcentagem de amostras ausentes por subconjunto.



Fonte: Elaborado pelo autor.

3.3.4 Construção de Matrizes Espaço-Temporais

Para melhor representar os dados, em relação às características espaciais e temporais, são construídas matrizes espaço-temporais, onde cada linha representa a evolução do fluxo de um único ponto de observação ao decorrer do tempo, e cada coluna representa o fluxo de todos os pontos de observação em um mesmo intervalo de tempo (MA et al., 2017). Representa-se uma matriz espaço-temporal por:

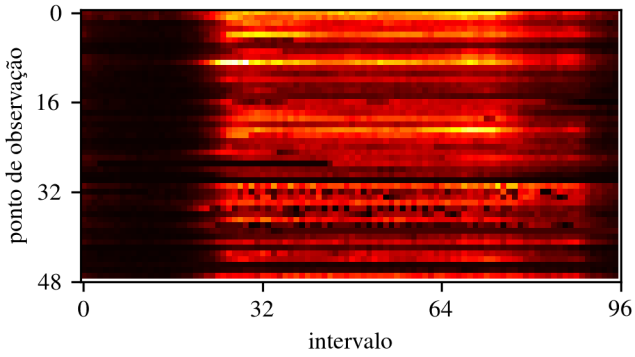
$$M_{p,t} = \begin{bmatrix} m_{1,1} & m_{1,2} & \dots & m_{1,t} \\ m_{2,1} & m_{2,2} & \dots & m_{2,t} \\ \dots & \dots & \dots & \dots \\ m_{p,1} & m_{p,2} & \dots & m_{p,t} \end{bmatrix} \tag{3.2}$$

onde p é o número de pontos de observação e t é o número de intervalos. Dessa forma, o elemento $m_{i,j}$ contém a quantidade de veículos que passou pelo ponto de observação i no intervalo de tempo j.

Essas matrizes se assemelham a uma imagem, e são utilizadas como base para o modelo CNN, mas também como referência para o modelo LSTM. A Figura 21 exemplifica uma matriz espaço-temporal em forma de imagem para um único dia. Essa imagem é construída a partir de um mapa de cores gerado pelos valores do fluxo, que vão de 0 a 1.347. Dessa forma, as regiões mais escuras indicam um fluxo mais baixo e as regiões mais claras indicam um fluxo mais alto. Percebe-se, então, que o fluxo tende a ser menor no início e no final do dia.

O próximo subtópico trata das correlações espaciais utilizadas para reordenar os pontos de observação nas matrizes espaço-temporais.

Figura 21 – Exemplo de uma matriz espaço-temporal em forma de imagem para um único dia.



Fonte: Elaborado pelo autor.

3.3.5 Reordenação dos Pontos de Observação

Como os pontos de observação não estão adjacentes uns aos outros no conjunto de dados utilizado neste trabalho, tanto por falta de equipamentos como pela limpeza dos dados, os pontos são reordenados com base na correlação espacial, para que essa proximidade seja simulada e melhore a performance dos modelos. Essa correlação espacial é calculada, a partir do primeiro ponto de observação, com relação aos demais pontos, através do Coeficiente de Pearson, dado por:

$$\rho_{x,y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{3.3}$$

onde $X = x_1, x_2, ..., x_n$ e $Y = y_1, y_2, ..., y_n$ representam as sequências de fluxo de dois pontos de observação diferentes. $\rho_{x,y}$ é um valor no intervalo $[-1, 1]$, e quanto mais próximo de -1 ou 1, maior é o índice de correlação entre as duas sequências (ZHANG et al., 2019).

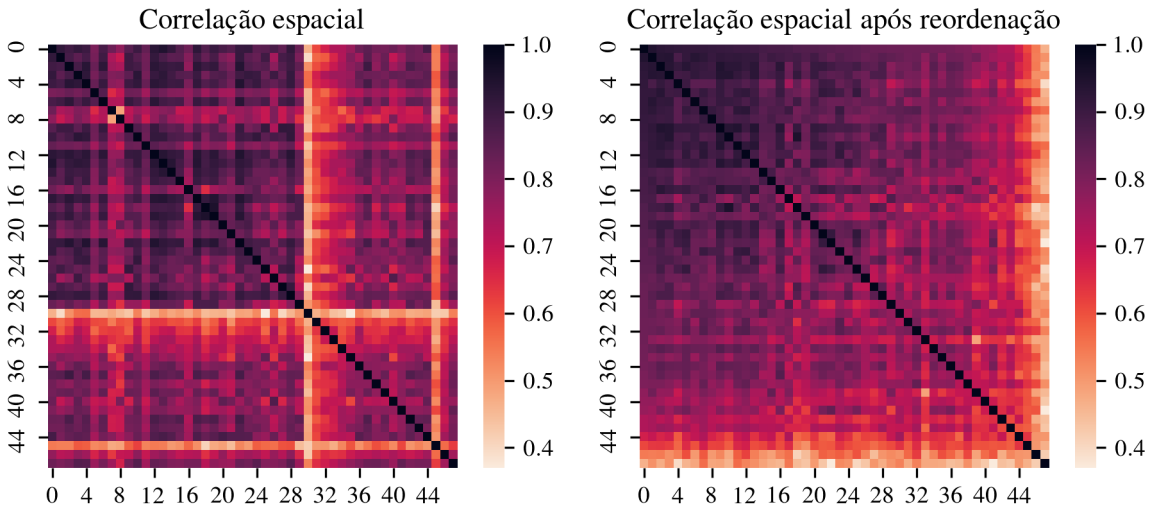
Utilizando os índices de correlação, os pontos são reordenados, de modo que os pontos com maior correlação fiquem mais próximos. A Figura 22 mostra o mapa de correlações antes e depois dessa reorganização, mostrando que os pontos ficam agrupados de uma forma melhor.

O próximo passo é dividir os subconjuntos em mais três subconjuntos, como trata o próximo subtópico.

3.3.6 Treinamento, Validação e Teste

Cada subconjunto é divido em mais três subconjuntos: um para treinamento, um para validação e um para teste. O subconjunto para treinamento é fornecido aos

Figura 22 – Mapa de correlação espacial antes e depois da reorganização dos pontos de observação.



Fonte: Elaborado pelo autor.

modelos durante o treinamento, de modo a obter as predições. Também durante o treinamento, utiliza-se o subconjunto de validação para verificar a performance dos modelos e, através da técnica de *early stopping*, determinar quando encerrar o treinamento. Já o subconjunto de teste é utilizado durante o teste para avaliar a performance final dos modelos.

A divisão utilizada neste trabalho é de aproximadamente 80% para treinamento, 10% para validação e 10% para teste. A quantidade de amostras por subconjunto é apresentada na Figura 23.

Resta, então, apenas um passo na etapa de pré-processamento dos dados: a normalização do fluxo, que é definida a seguir.

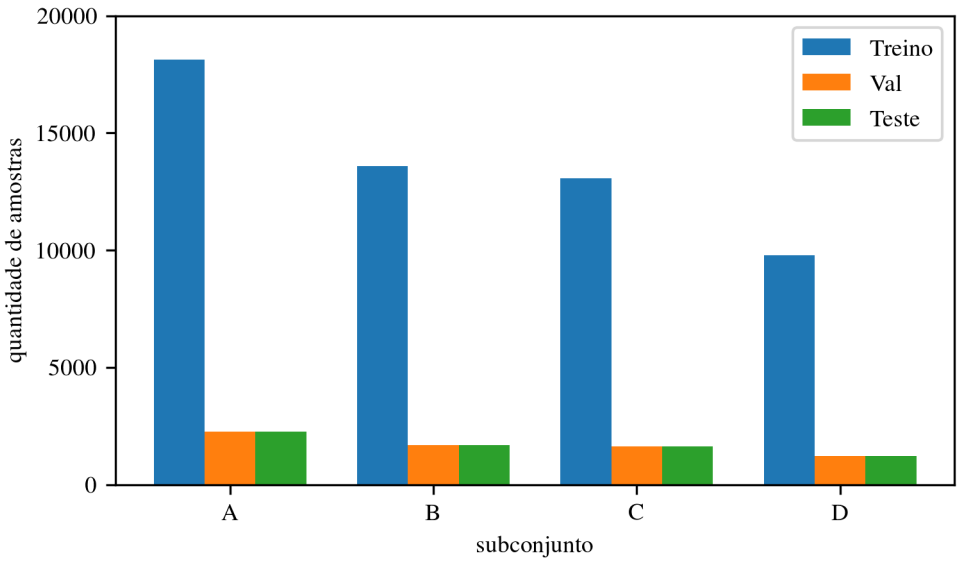
3.3.7 Normalização do Fluxo

Em problemas de *Deep Learning*, é comum normalizar valores numéricos muito grandes para que fiquem em um intervalo menor, o que reduz a complexidade e o tempo de treinamento dos modelos (BROWNLEE, 2019).

No conjunto de dados utilizado neste trabalho, há uma única coluna que possui valores numéricos: “fluxo”. Portanto, esses valores são normalizados utilizando a técnica de *min-max normalization*, que consiste em escalar os valores para que fiquem no intervalo $[0, 1]$:

$$valor = \frac{valor - min}{max - min} \tag{3.4}$$

Figura 23 – Quantidade de amostras por subconjunto.



Fonte: Elaborado pelo autor.

onde *valor* é o valor a ser escalado, *min* é o valor mínimo de todos os valores e *max* é o valor máximo de todos os valores.

É importante ressaltar que se normaliza os subconjuntos de treinamento, validação e teste, porém, utilizando os valores *min* e *max* do subconjunto de treinamento para todos. Isso garante que informações dos subconjuntos de validação e teste não sejam "vazadas" durante o treinamento, pois, do ponto de vista prático, esses subconjuntos são desconhecidos ao treinamento.

Também vale ressaltar que, assim como os valores são normalizados para serem alimentados aos modelos, é preciso inverter a normalização para as predições feitas pelos modelos, antes que elas possam ser comparadas com os valores reais. A inversão da normalização é dada por:

$$valor = valor \cdot (max - min) + min \tag{3.5}$$

Esse passo encerra o pré-processamento dos dados. Portanto, os dados estão prontos para serem alimentados aos modelos. Para isso, define-se as estruturas dos modelos nas próxima seções.

3.4 *Convolutional Neural Network*

Nesta seção, define-se a estrutura utilizada para o modelo CNN, assim como os hiperparâmetros, a função de ativação, a função de perda e o algoritmo de otimização selecionados.

3.4.1 Estrutura

O modelo CNN é composto por cinco tipos de camadas: a camada de entrada, a camada convolucional, a camada de *pooling*, a camada *fully connected*, e a camada de saída. Cada uma dessas camadas é detalhada matematicamente a seguir.

A camada de entrada é a matriz espaço-temporal definida anteriormente. A partir da camada de entrada, tem-se, então, as camadas convolucionais e as camadas de *pooling*. Considerando um *kernel* convolucional de tamanho $m \times n$, a operação de convolução é dada por:

$$y_i^{conv} = g(\sum_{l=1}^{c_1} \sum_{m=1}^{m} \sum_{n=1}^{n} ((W_i^{conv})_{m,n,l} \times (X)_{m,n,l}) + b_i^{conv}), i \in [1, 2, ..., c_2] \tag{3.6}$$

onde g é a função de ativação, l é a camada convolucional, W é o conjunto de pesos, X é a matriz de entrada, b é o *bias*, e c_l representa o número de *kernels* convolucionais na camada convolucional l, que, para a primeira camada, é igual a 1.

Para a camada de *pooling*, este trabalho utiliza uma operação do tipo *max pooling*. Com um tamanho $p \times q$, essa operação é dada por:

$$y_j^{pool} = max((y_j^{conv})_{p,q}), j \in [1, 2, ..., c_2] j \in [1, 2, ..., c_2] \tag{3.7}$$

A camada convolucional serve para extrair características da matriz, e a camada de *pooling* serve para reduzir o número de parâmetros na rede, consequentemente reduzindo a complexidade do modelo, e ao mesmo tempo, extraindo apenas as características relevantes (ZHANG et al., 2019).

Após a camada de *pooling*, a matriz precisa ser convertida para um vetor de única dimensão, que é a entrada da camada *fully connected*. Essa operação é chamada de *flatten*:

$$y^{flatten} = flatten([y_i^{pool}, y_2^{pool}, ..., y_{c2}^{pool}]) \tag{3.8}$$

Finalmente, a saída da camada *fully connected* é dada por:

$$\hat{y} = g(W^{fc} \cdot y^{flatten} + b^{fc}) \tag{3.9}$$

A camada de saída é, portanto, um vetor contendo as predições de fluxo para cada um dos pontos de observação.

Seguindo essa estrutura, os hiperparâmetros do modelo são definidos a seguir.

3.4.2 Hiperparâmetros

Para o modelo CNN, os hiperparâmetros a serem definidos são: número de camadas convolucionais, tamanho das camadas convolucionais, tamanho do *kernel* convolucional, tamanho da camada de *pooling*, número de camadas *fully connected*, e tamanho das camadas *fully connected*.

Não existe uma configuração de hiperparâmetros ideal para todos os casos. Isso deve ser ajustado para cada problema, de preferência por alguém especializado no problema que consiga definir os parâmetros ótimos. Com base nos resultados obtidos em Zhang et al. (2019), este trabalho utiliza os hiperparâmetros definidos na Tabela 6.

Tabela 6 – Hiperparâmetros do modelo CNN.

Camada	Nome	Parâmetros	Dimensão
1	Entrada	-	(1, 48, 4)
2	Convolução	(16, 3, 3)	(16, 48, 4)
3	*Pooling*	(2, 2)	(16, 24, 2)
4	Convolução	(32, 3, 3)	(32, 24, 2)
5	*Pooling*	(2, 2)	(32, 12, 1)
6	*Fully connected*	-	(384,)
7	*Fully connected*	-	(96,)
8	Saída	-	(48,)

Fonte: Elaborado pelo autor.

Para a operação de convolução, é aplicado um *padding* para manter o mesmo tamanho da matriz. E antes de cada camada *fully connected*, é aplicado um *dropout* de 0,2.

Em seguida, define-se a função de ativação.

3.4.3 Função de Ativação

É utilizada a função de ativação ReLU para as camadas convolucionais e para a primeira camada *fully connected*:

$$g(x) = \begin{cases} x, & \text{se } x \geq 0 \\ 0, & \text{caso contrário} \end{cases} \tag{3.10}$$

Já a segunda camada *fully connected* utiliza a função de ativação sigmoide, para que as predições do modelo estejam no intervalo $[0, 1]$, e possam ser escaladas

conforme visto na seção 3.3.7. Ela é dada por:

$$g(x) = \frac{1}{1 + e^{-x}} \tag{3.11}$$

A função de perda é definida a seguir.

3.4.4 Função de Perda

Para função de perda, utiliza-se a MAE, por não ser muito influenciada por valores discrepantes (ou *outliers*):

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |\hat{y}_i - y_i| \tag{3.12}$$

onde $\hat{y}_i$ é o valor de predição e y_i é o valor esperado.

O próximo subtópico define o algoritmo de otimização.

3.4.5 Algoritmo de Otimização

O algoritmo de otimização utilizado é o Estimação de Momento Adaptativo (Adam – *Adaptive Moment Estimation*), que é basicamente uma versão do SGD com taxas de aprendizado adaptativas para diferentes parâmetros (ZHANG et al., 2019). Este trabalho utiliza um *mini batch* de tamanho 4, uma taxa de aprendizado de 0,01 e uma normalização *L2* com coeficiente de 0,1.

A próxima seção define a estrutura do modelo LSTM.

3.5 *Long Short-Term Memory*

Esta seção define a estrutura utilizada para o modelo LSTM. Também são definidos os hiperparâmetros e a função de ativação.

3.5.1 Estrutura

O modelo LSTM é composto por três tipos de camadas: a camada de entrada, a camada oculta, e a camada de saída. Cada uma dessas camadas é detalhada matematicamente a seguir.

Como o modelo LSTM não é capaz de extrair características espaciais, por natureza, não é possível alimentar a matriz espaço-temporal definida anteriormente, pois o modelo é configurado para receber sequências temporais e suas características. Em consequência, é necessário treinar um modelo LSTM para cada um dos pontos de observação.

A forma como as características espaciais são incorporadas no modelo LSTM é através da inserção de características adicionais em cada sequência. Para cada sequência de fluxo de um ponto de observação, são inseridas como características as sequências de fluxo dos dois pontos de observação mais próximos. Como a matriz espaço-temporal está ordenada em relação às correlações espaciais, conforme visto no subtópico 3.3.5, a extração desses pontos é facilitada. Dessa forma, a camada de entrada do modelo LSTM é uma matriz definida por:

$$M_{p,t} = \begin{bmatrix} m_{p,1} & m_{p_1,1} & m_{p_2,1} \\ m_{p,2} & m_{p_1,2} & m_{p_2,2} \\ \cdots & \cdots & \cdots \\ m_{p,t} & m_{p_1,t} & m_{p_2,t} \end{bmatrix} \tag{3.13}$$

onde p é o ponto de observação atual, p_1 e p_2 são os dois pontos de observação mais próximos, e t é o passo de tempo. Cada linha da matriz é um passo de tempo, e contém o vetor de características desse passo.

Cada unidade oculta na camada oculta é uma célula LSTM, conforme visto anteriormente. Os portões dessa célula são calculados por:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.14}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.15}$$

$$\tilde{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{3.16}$$

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \tag{3.17}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.18}$$

$$h_t = o_t \times tanh(C_t) \tag{3.19}$$

onde f_t é o *forget gate*, i_t é o *input gate*, $\tilde{C}_t$ é o candidato ao novo estado de célula, C_t é o novo estado de célula, o_t é o *output gate*, e h_t é o novo estado oculto. Percebe-se que cada um dos portões possui seu próprio conjunto de pesos W e seu próprio *bias* b (LUO et al., 2019).

A saída da camada oculta é uma matriz contendo os valores de todos os estados ocultos. Dessa forma, a última coluna da matriz, que contém os valores do último estado, é a entrada da camada *fully connected*.

Finalmente, a saída da camada *fully connected* é dada pela fórmula 3.9. A camada de saída é, portanto, um único valor numérico contendo a predição de fluxo para o ponto de observação em questão.

Seguindo essa estrutura, os hiperparâmetros do modelo são definidos a seguir.

3.5.2 Hiperparâmetros

Para o modelo LSTM, os hiperparâmetros a serem definidos são: número de camadas ocultas, tamanho das camadas ocultas, número de camadas *fully connected*, e tamanho das camadas *fully connected*.

Com base nos resultados obtidos em Luo et al. (2019), este trabalho utiliza os hiperparâmetros definidos na Tabela 7. Após cada camada oculta, é aplicado um *dropout* de 0,2.

Tabela 7 – Hiperparâmetros do modelo LSTM.

Camada	Nome	Parâmetros	Dimensão
1	Entrada	-	(4, 3)
2	Oculta	(2, 4, 6)	(4, 6)
3	Oculta	(2, 4, 6)	(4, 6)
4	*Fully connected*	-	(6,)
5	*Fully connected*	-	(2,)
6	Saída	-	(1,)

Fonte: Elaborado pelo autor.

Em seguida, define-se a função de ativação.

3.5.3 Função de Ativação

É utilizada a função de ativação TanH para a primeira camada *fully connected*:

$$g(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{3.20}$$

Já a segunda camada *fully connected* utiliza a mesma função de ativação sigmoide utilizada pelo modelo CNN, pelos mesmos motivos apresentados anteriormente.

A função de perda e o algoritmo de otimização também são os mesmos utilizados pelo modelo CNN.

Em seguida, são definidas as métricas de erro selecionadas para a análise dos modelos.

3.6 Métricas de Erro

Não existe um conceito exato de acurácia quando se trata de problemas de regressão, pois a predição não é um valor exato. Por isso, utiliza-se métricas de erro que mostram o quão próximo um valor de predição está do valor esperado, com base em fórmulas diferentes. Essa taxa de erro não é muito representativa por si só, pois seu valor depende do problema e do conjunto de dados. Portanto, ela é utilizada para efeitos de comparação com outros modelos aplicados ao mesmo problema. A única conclusão que pode ser feita com uma taxa de erro por si só é que quanto menor a taxa de erro, melhor o modelo (BROWNLEE, 2016).

As métricas de erro utilizadas neste trabalho são: MAE, MSE e RMSE. Elas são selecionadas por serem as métricas mais adequadas para problemas de regressão. A MAE é definida pela fórmula 3.12. Já a MSE e a RMSE são definidas por:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{3.21}$$

$$RMSE = \sqrt{MSE} \tag{3.22}$$

onde $\hat{y}_i$ é o valor de predição e y_i é o valor esperado.

É com essa métricas que são avaliados os resultados apresentados no próximo capítulo.

4 RESULTADOS

Com os dados pré-processados e os modelos definidos, são executadas, então, as etapas de treinamento e teste, para a obtenção dos resultados. Esses resultados são apresentados a seguir.

4.1 *Convolutional Neural Network*

Os resultados do modelo CNN são apresentados na Tabela 8. Percebe-se que os subconjuntos A e C apresentam os melhores resultados. O motivo pode ser pelo fato de que esses conjuntos contam com uma quantidade maior de amostras, em comparação com os subconjuntos B e D, que possuem apenas amostras de 05:00 a 23:00. Outro fator pode ser que o modelo não consegue conectar os períodos ausentes muito bem.

Tabela 8 – Resultados do modelo CNN.

Métrica	Subconjunto			
	A	B	C	D
MAE	48,586	55,220	47,927	56,759
MSE	5.943,580	7.429,992	6.371,275	8.047,616
RMSE	72,077	83,409	73,411	87,647

Fonte: Elaborado pelo autor.

A Tabela 9 apresenta a comparação entre o tempo gasto durante o treinamento e o tempo gasto durante o teste. Essa comparação mostra uma das grandes vantagens de se utilizar *Deep Learning*, pois o modelo demora um pouco para treinar, mas consegue realizar predições em um tempo extremamente curto.

Tabela 9 – Diferença de tempo entre o treinamento e o teste para o modelo CNN, em segundos.

Etapa	Subconjunto			
	A	B	C	D
Treinamento	705,207	276,244	269,586	559,477
Teste	1,133	0,827	0,899	0,455

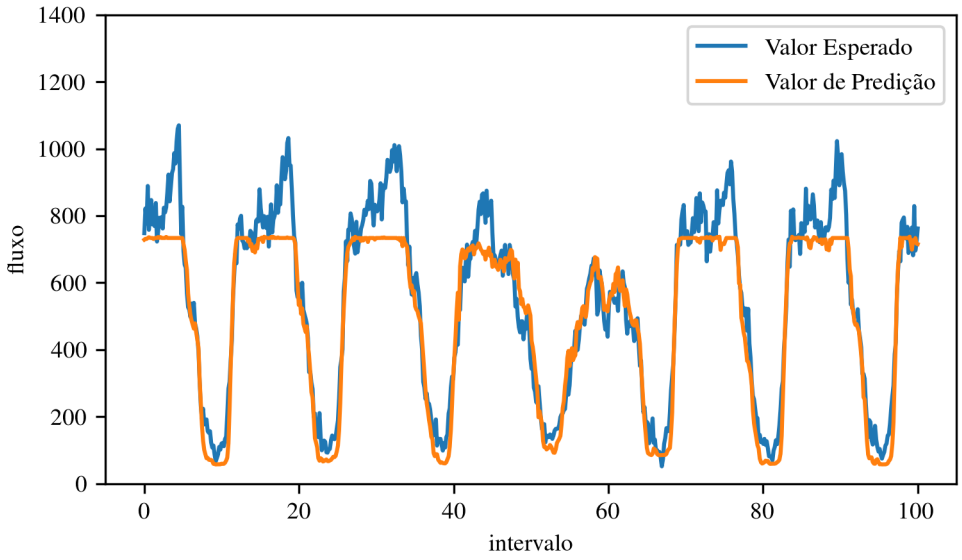
Fonte: Elaborado pelo autor.

As figuras 24-27 mostram a diferença entre os valores de predição e os valores esperados. Por algum motivo, percebe-se que o modelo CNN limita o valor da predição por volta de 750. Não é identificado o motivo exato para esse comportamento, mas

teoriza-se que ele seja causado por *outliers* ou uma limitação na escala de predição, apesar de que o modelo utiliza a mesma escala utilizada durante a normalização do fluxo.

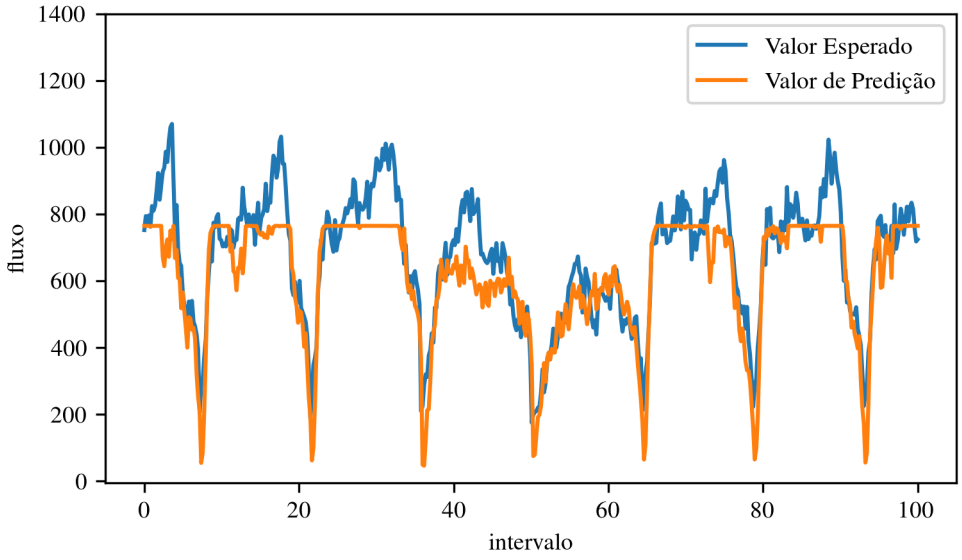
A próxima seção apresenta os resultados do modelo LSTM.

Figura 24 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto A, para o modelo CNN.



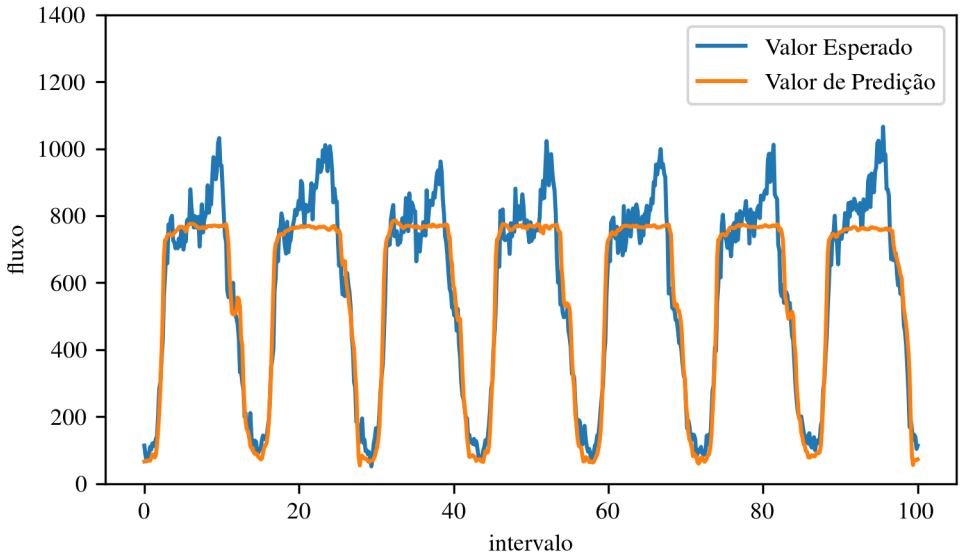
Fonte: Elaborado pelo autor.

Figura 25 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto B, para o modelo CNN.



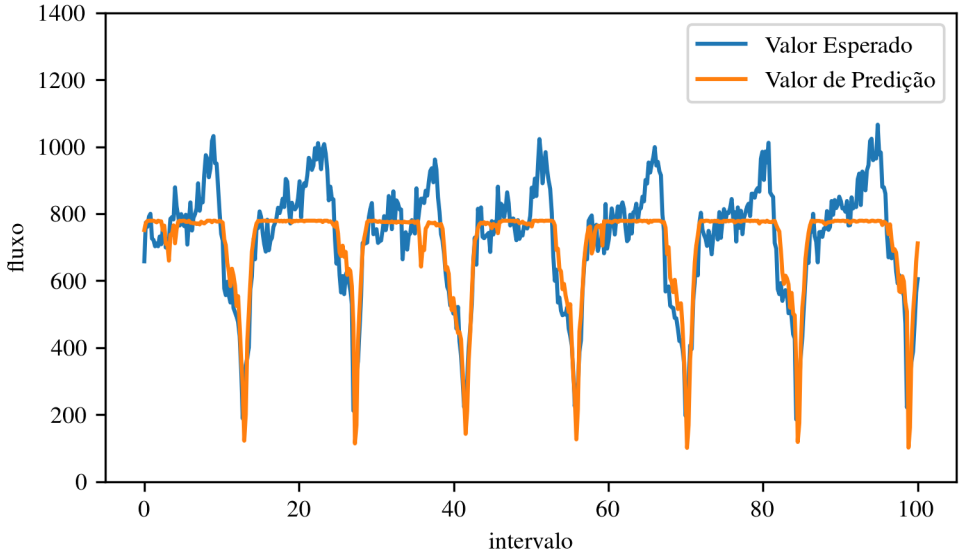
Fonte: Elaborado pelo autor.

Figura 26 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto C, para o modelo CNN.



Fonte: Elaborado pelo autor.

Figura 27 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto D, para o modelo CNN.



Fonte: Elaborado pelo autor.

4.2 *Long Short-Term Memory*

Os resultados do modelo LSTM são apresentados na Tabela 10. Percebe-se que, novamente, os subconjuntos A e C apresentam os melhores resultados. Também identifica-se um problema no subconjunto B, onde o modelo não foi capaz de realizar as predições. Não foi identificado o motivo para esse problema.

Tabela 10 – Resultados do modelo LSTM.

Métrica	Subconjunto			
	A	B	C	D
MAE	36,970	169,725	36,503	41,733
MSE	2.263,583	3.9586,192	2.268,378	3.065,039
RMSE	42,870	175,075	43,160	49,713

Fonte: Elaborado pelo autor.

A Tabela 11 apresenta a comparação entre o tempo gasto durante o treinamento e o tempo gasto durante o teste, que possui as mesmas características do modelo CNN.

Tabela 11 – Diferença de tempo entre o treinamento e o teste para o modelo LSTM, em segundos.

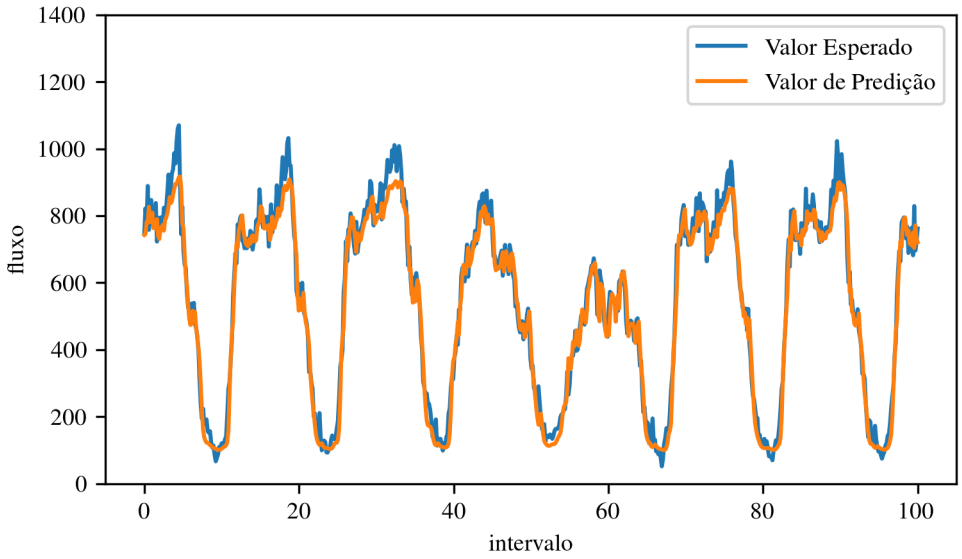
Etapa	Subconjunto			
	A	B	C	D
Treinamento	557,423	377,798	403,485	185,259
Teste	0,835	0,618	0,582	0,442

Fonte: Elaborado pelo autor.

As figuras 28-31 mostram a diferença entre os valores de predição e os valores esperados. Mostra-se que, com a exceção do subconjunto B, as predições estão bem próximas dos valores esperados.

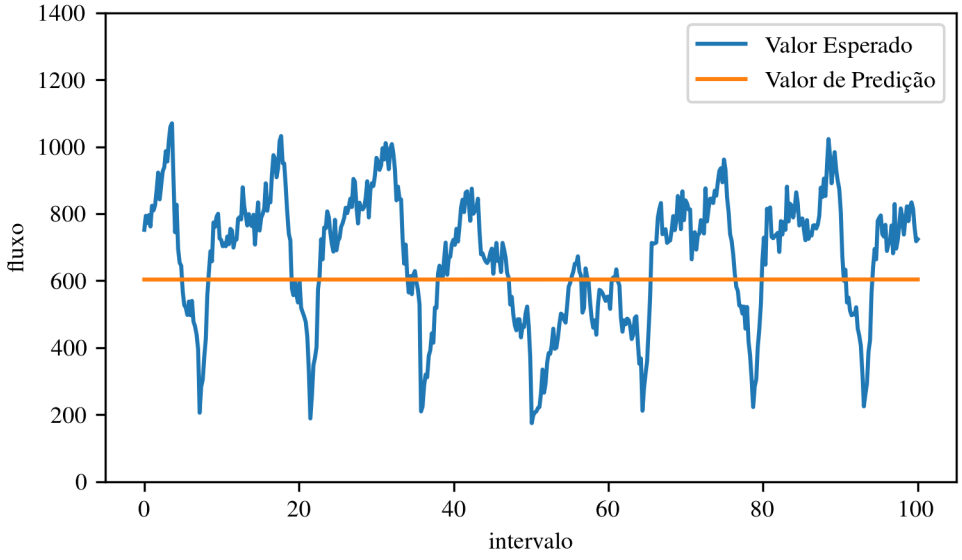
No próximo capítulo, são apresentadas as conclusões e as considerações finais.

Figura 28 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto A, para o modelo LSTM.



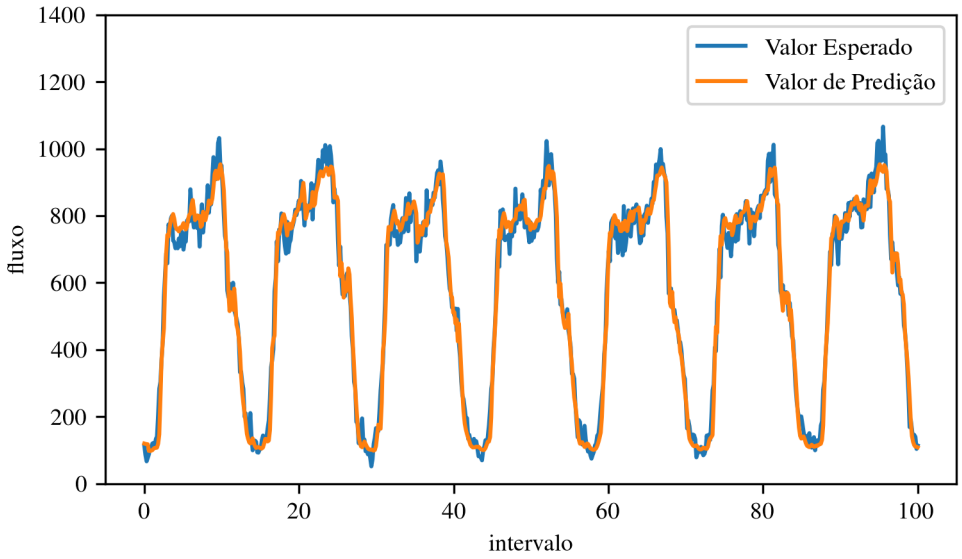
Fonte: Elaborado pelo autor.

Figura 29 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto B, para o modelo LSTM.



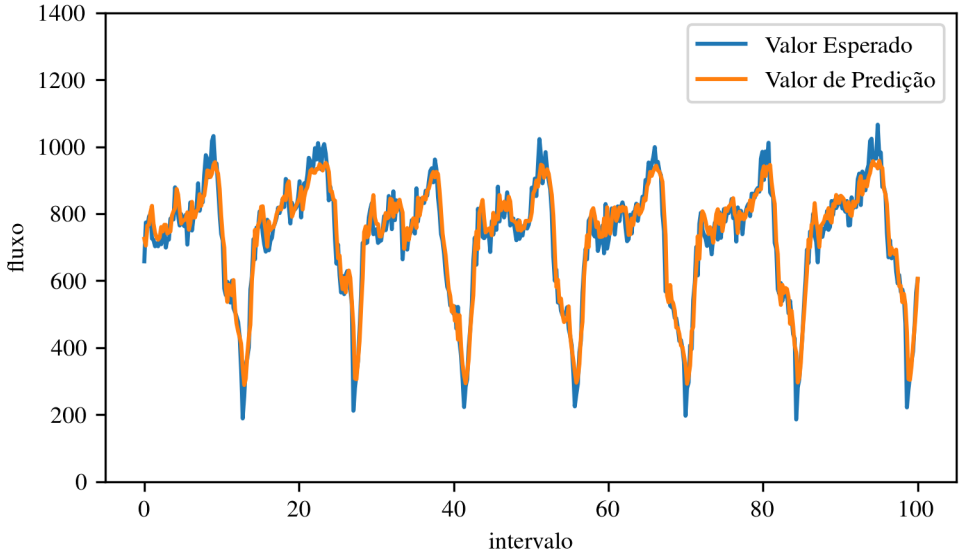
Fonte: Elaborado pelo autor.

Figura 30 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto C, para o modelo LSTM.



Fonte: Elaborado pelo autor.

Figura 31 – Diferença entre os valores de predição e os valores esperados para um ponto específico do subconjunto D, para o modelo LSTM.



Fonte: Elaborado pelo autor.

5 CONCLUSÃO

Visando minimizar o tempo de viagens dos motoristas e a emissão de CO_2 na atmosfera, este trabalho se propôs a realizar uma análise comparativa entre os modelos de *Deep Learning* CNN e LSTM para o problema de predição de fluxo do tráfego urbano, utilizando um conjunto de dados da cidade de Recife, Pernambuco. Os dados foram pré-processados para a otimização dos resultados, e as estruturas dos modelos foram definidas com base nos trabalhos relacionados. Além disso, os dados também foram divididos em quatro subconjuntos diferentes, para a avaliação de cenários diferentes, e o impacto desses cenários nos resultados. Os resultados foram avaliados com base em três métricas de erro diferentes: MAE, MSE e RMSE.

Foi utilizado o ambiente de desenvolvimento Google Colab, juntamente com a aplicação *web* Jupyter Notebook, e a linguagem de programação Python com suas bibliotecas, especialmente a biblioteca PyTorch, para a implementação dos modelos e os experimentos realizados.

Com base nos resultados obtidos, o modelo LSTM apresentou melhores resultados do que o modelo CNN. A menor taxa de erro MAE para o modelo CNN foi de 47,927, para o subconjunto C, que contém apenas amostras de dias de semana. Já para o modelo LSTM, a menor taxa de erro MAE foi de 36,503, também para o subconjunto C. Essa taxa de erro menor para o subconjunto C, em relação aos outros conjuntos, pode estar relacionada à irregularidade do fluxo do tráfego durante os fins de semana.

Também identifica-se que os subconjuntos A e C obtiveram melhores resultados do que os subconjuntos B e D, que incluem amostras apenas de 05:00 a 23:00. Essa diferença pode ser explicada pela quantidade de amostras disponíveis em cada subconjunto, ou também pela inabilidade dos modelos de lidarem com o período ausente.

Destaca-se que ambos os modelos foram capazes de extrair as características espaciais e temporais dos dados, através das modificações feitas para cada um, sendo que o modelo CNN apresentou maior capacidade para extrair essas características por sua natureza convolucional.

Porém, ambos os modelos apresentaram alguns problemas: o modelo CNN não conseguiu realizar predições mais altas do que um valor próximo a 750, e o modelo LSTM não conseguiu realizar predições para o subconjunto B. É possível que, com a resolução desses problemas, as taxas de erro dos modelos estejam mais próximas umas das outras. Entretanto, a causa desses problemas não foi identificada.

Percebe-se também que as taxas de erro obtidas neste trabalho foram mais altas do que as taxas de erro apresentadas pelos trabalhos relacionados. Isso pode estar relacionado ao conjunto de dados específico utilizado neste trabalho, como os

pontos de observação não estavam adjacentes, e foi preciso reorganizá-los de acordo com as correlações espaciais das sequências.

Conclui-se, portanto, que o modelo LSTM se sobressai quando aplicado ao problema de predição de fluxo do tráfego urbano, utilizando um conjunto de dados da cidade de Recife, Pernambuco. Esse resultado pode ser explicado pela natureza do modelo LSTM para lidar com séries temporais. O código fonte da implementação dos modelos e dos experimentos realizados neste trabalho encontram-se no Google Colab: https://colab.research.google.com/drive/1jwxL_zR_byGblOmQQPrEcmLqrFLxFkMV

O próximo capítulo encerra o trabalho ao apresentar as sugestões para trabalhos futuros.

6 TRABALHOS FUTUROS

Por questões de tempo, não foi possível explorar os modelos com maior profundidade neste trabalho. Por isso, uma sugestão para trabalhos futuros é realizar essa exploração, utilizando outras configurações de hiperparâmetros para verificar o impacto nos resultados.

Também recomenda-se realizar técnicas diferentes de pré-processamento dos dados, como outras técnicas para o preenchimento de amostras ausentes e para a normalização do fluxo, de modo a tentar obter uma representação melhor dos dados.

Além disso, é pertinente a investigação do impacto de incluir mais informações no conjunto de dados, como informações meteorológicas e informações relacionadas a feriados.

Finalmente, outra direção relevante é testar modelos diferentes, ou até mesmo uma combinação do modelo CNN com o modelo LSTM, como abordam alguns trabalhos relacionados.

REFERÊNCIAS

BROWNLEE, J. Metrics to evaluate Machine Learning algorithms in Python. **Machine Learning Mastery**, maio 2016. Disponível em: <https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/>. Acesso em: 20 nov. 2020.

BROWNLEE, J. How to use data scaling to improve Deep Learning model stability and performance. **Machine Learning Mastery**, fev. 2019. Disponível em: <https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/>. Acesso em: 20 nov. 2020.

COLABORATORY. Colaboratory. 2020. Disponível em: <https://colab.research.google.com/notebooks/intro.ipynb>. Acesso em: 20 nov. 2020.

CTTU. Velocidade das vias - quantitativo por velocidade média - 2019. **Portal de Dados Abertos da Cidade de Recife**, abr. 2020. Disponível em: <http://dados.recife.pe.gov.br/dataset/velocidade-das-vias-quantitativo-por-velocidade-media-2019>. Acesso em: 6 abr. 2020.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016.

GUPTA, N. Why is python used for Machine Learning? **Hacker Noon**, out. 2019. Disponível em: <https://hackernoon.com/why-python-used-for-machine-learning-u13f922ug>. Acesso em: 20 nov. 2020.

IEA. World. In: ____. **CO2 emissions from fuel combustion highlights (2019 edition)**. [S.l.]: IEA, 2019. p. 23.

LUO, X. et al. Spatiotemporal traffic flow prediction with KNN and LSTM. **Journal of Advanced Transportation**, fev. 2019.

LV, Y. et al. Traffic flow prediction with big data: a Deep Learning approach. **IEEE Transactions on Intelligent Transportation Systems**, v. 16, n. 2, p. 865–873, abr. 2015.

MA, X. et al. Learning traffic as images: a deep convolutional neural network for large-scale transportation network speed prediction. **Sensors**, v. 17, n. 4, p. 818, abr. 2017.

MATPLOTLIB. Matplotlib: Python plotting. 2020. Disponível em: <https://matplotlib.org/>. Acesso em: 20 nov. 2020.

MORETTI, I. Metodologia de pesquisa do TCC: conheça os tipos e veja como definir. **ViaCarreira**, maio 2017. Disponível em: <https://viacarreira.com/metodologia-de-pesquisa-do-tcc/>. Acesso em: 6 abr. 2020.

NUMPY. Numpy: the fundamental package for scientific computing with Python. 2020. Disponível em: <https://numpy.org/>. Acesso em: 20 nov. 2020.

OLAH, C. Understanding LSTM networks. **Colah's Blog**, ago. 2015. Disponível em: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>. Acesso em: 20 nov. 2020.

PANDAS. Pandas: Python data analysis library. 2020. Disponível em: <https://pandas.pydata.org/>. Acesso em: 20 nov. 2020.

PROJECT JUPYTER. Project Jupyter. 2020. Disponível em: <https://jupyter.org/>. Acesso em: 20 nov. 2020.

RUSSELL, S.; NORVIG, P. **Artificial intelligence**: a modern approach. 3. ed. [S.l.]: Prentice Hall, 2009.

SCIKIT-LEARN. Underfitting vs. overfitting. **Scikit-learn**, 2020. Disponível em: <https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html>. Acesso em: 20 nov. 2020.

STEVENS, E.; ANTIGA, L.; VIEHMANN, T. **Deep Learning with PyTorch**. [S.l.]: Manning, 2020.

WANG, Y. et al. Enhancing transportation systems via Deep Learning: a survey. **Transportation Research Part C: Emerging Technologies**, v. 99, p. 144–163, fev. 2019.

ZHANG, W. et al. Short-term traffic flow prediction based on spatio-temporal analysis and CNN Deep Learning. **Transportmetrica A: Transport Science**, v. 15, n. 2, p. 1688–1711, dez. 2019.

ZHAO, Z. et al. LSTM network: a Deep Learning approach for short-term traffic forecast. **IET Intelligent Transport Systems**, v. 11, n. 2, p. 68–75, mar. 2017.