Pontifícia Universidade Católica
do Rio de Janeiro

Eduardo Betine Bucker

Redes Convolucionais aplicadas à Classificação de Ruído Sísmico

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática da PUC-Rio.

Orientador: Prof. Sérgio Colcher

Rio de Janeiro
setembro de 2020

Pontifícia Universidade Católica
do Rio de Janeiro

Eduardo Betine Bucker

Redes Convolucionais aplicadas à Classificação de Ruído Sísmico

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo.

Prof. Sérgio Colcher
Orientador
Departamento de Informática – PUC-Rio

Prof. Ruy Luiz Milidiú
Departamento de Informática – PUC-Rio

Prof. André Bulcão
Petróleo Brasileiro – Rio de Janeiro - Matriz

Rio de Janeiro, 14 de setembro de 2020

Eduardo Betine Bucker

Graduado em Engenharia de Produção pela Universidade Federal do Rio de Janeiro.

Ficha Catalográfica

Betine, Eduardo

Redes Convolucionais aplicadas à Classificação de Ruído Sísmico / Eduardo Betine Bucker; orientador: Sérgio Colcher. – Rio de janeiro: PUC-Rio, Departamento de Informática, 2020.

v., 65 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Aprendizado Profundo;. 3. Aprendizado de Máquinas;. 4. Classificação de Imagens.. 5. Redes Neurais Convolucionais;. I. Colcher, Sérgio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Agradecimentos

Agradeço à minha família pelo apoio e suporte. Sem eles nada seria possível.

Gostaria de agradecer ao meu orientador Sérgio Colcher e ao Ruy Luiz Milidiú por todas as horas de discussões, ensinamentos e conselhos. Os dias foram bem produtivos e vocês foram fundamentais para a evolução deste trabalho.

Agradeço ao doutor André Bulcão pela paciência e atenção na condução deste trabalho. A sua participação e contribuição foi importante na revisão e melhoria do trabalho.

Agradeço à todos os meus amigos do TELEMÍDIA e do LEARN por toda ajuda concedida, à PUC pela oportunidade concedida que me permite melhorar minha qualificação e a Petrobras pelo ajuda na construção do dataset e disponibilidade dos profissionais.

Em especial, gostaria de agradecer ao Antônio Busson, pelo convite de participação ao projeto Petrobras e por todas as aulas e introdução ao mundo do Machine Learning, e ao Luís Felipe Müller por todos os conselhos e discussões realizadas ao longo deste trabalho. A ajuda de vocês foi essencial para a tomada de decisão e melhorias do trabalho.

Gostaria também de agradecer ao Raphael Rocha, Matheus Cabral e Ivan Jesus, que colaboraram ativamente ao longo do projeto. Além disso, agradeço aos meu amigos João Forny, Gabriel Noronha, Pedro Almeida, Bruno Leitão e Lucas Gomes por contribuírem com ideias e conhecimentos para minha formação.

Também devo agradecimentos à todos os professores do Departamento de Informática da PUC, pois estes contribuíram na minha formação. Em especial, gostaria de agradecer ao professor Sérgio Lifschitz que teve contribuições direta para melhoria da dissertação.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Resumo

Betine, Eduardo; Colcher, Sérgio. **Redes Convolucionais aplicadas à Classificação de Ruído Sísmico**. Rio de Janeiro, 2020. 65p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Modelos baseados em redes neurais profundas como as Redes Neurais Convolucionais proporcionaram avanços significativos em diversas áreas da computação. No entanto, essa tecnologia é ainda pouco aplicada à predição de qualidade sísmica, que é uma atividade relevante para exploração de hidrocarbonetos. Ser capaz de, rapidamente, classificar o ruído presente em aquisições de dados sísmicos permite aceitar ou rejeitar essas aquisições de forma eficiente, o que além de economizar recursos também melhora a interpretabilidade dos dados. Neste trabalho apresenta-se um dataset criado a partir de 6.918 aquisições manualmente classificadas pela percepção de especialistas e pesquisadores, que serviu de base para o treinamento, validação e testes de um classificador, também proposto neste trabalho, baseado em uma rede neural convolucional. Em resultados empíricos, observou-se-se um F1 Score de 95,58 % em uma validação cruzada de 10 folds e 93,56 % em um conjunto de holdout de teste.

Palavras-chave

Aprendizado Profundo; Aprendizado de Máquinas; Classificação de Imagens. Redes Neurais Convolucionais;

Abstract

Betine, Eduardo; Colcher, Sérgio (Advisor). **Convolutional Networks applied to Seismic Noise Classification**. Rio de Janeiro, 2020. 65p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Deep Learning based models, such as Convolutional Neural Networks (CNNs), have led to significant advances in several areas of computing applications. Nevertheless, this technology is still rarely applied to seismic quality prediction, which is a relevant task in hydrocarbon exploration. Being able to promptly classify noise in common shot gather(CSG) acquisitions of seismic data allows the acceptance or rejection of those aquisitions, not only saving resources but also increasing the interpretability of data. In this work, we introduce a real-world classification dataset based on 6.918 common shot gather, manually labeled by perception of specialists and researches. We use it to train a CNN classification model for seismic shot-gathers quality prediction. In our empirical evaluation, we observed an F1 Score of 95,58 % in 10 fold cross-validation and 93,56 % in a Holdout Test.

Keywords

Deep Learning; Machine Learning; Image Classification. Convolutional Neural Network;

Sumário

Lista de figuras

Lista de tabelas

Lista de Abreviaturas

AM – Aprendizado de Máquina
AP – Aprendizado Profundo
CE – Compressão e Excitação
ML – *Machine Learning*
DL – *Deep Learning*
CSG – *Common Shot Gather*
CNN – *Convolutional Neural Network*
ILSVRC – *ImageNet Large Scale Visual Recognition Competition*
FC – *Fully Connected*
CV – *Cross Validation*
ReLU – *Rectified Linear Unit*
SE – *Squeeze and Excitation*
SEG-Y – *Society of Exploration Geophysicist*
SVM – *Support Vector Machines*
CSG – *Common Shot Gather*
CMP – *Common Mid Point*
NMO – *Normal Moveout*
GPU – *Graphical processing unit*

1
Introdução

No contexto da Geofísica, o ruído sísmico pode ser caracterizado por vibrações, de diferentes fontes, que produzem uma componente indesejada no sinal. Dados ruidosos atrapalham a análise e podem até mesmo inviabilizar a sua utilização. Na busca pela melhoria da qualidade dos dados, esta dissertação se propõe a classificar, em termos de intensidade, o ruído presente em reunião de tiros, denominado em inglês por Common Shot Gather (CSG). Uma CSG é uma coleção de traços sísmicos com uma fonte emissora em comum e sua representação é utilizada como ferramenta para inspecionar a qualidade do sinal. No capítulo 2 é detalhado a definição de uma CSG.

A qualidade de uma aquisição de dados sísmicos varia devido a diversos fatores. Ao depender do local da aquisição, do horário de realização, da intensidade das ondas do mar, das intempéries, entre outros motivos, a CSG pode obter um sinal mais ruidoso ou menos ruidoso. A classificação da qualidade é sempre realizada de forma relativa ao nível ruidoso da aquisição, portanto duas representações semelhantes, mas oriundas de aquisições diferentes, podem ter rótulos distintos.

A Classificação do Ruído Sísmico é uma tarefa muito importante, não apenas por permitir, ainda em estágios iniciais da aquisição de dados, aferir a qualidade da sua coleta mas também por possibilitar que os profissionais possam também tratar, de forma diferenciada, o fluxo de processamento dos dados levando em conta o nível de ruído, evitando, por exemplo, esforços desnecessários em dados muito ruidoso (que, a princípio, deveriam ser descartados).

Este trabalho foi desenvolvido por meio de uma parceria, oficializada por um termo de compromisso, entre a PUC(grupo LEARN) e a Petrobras. Nesse contexto, foi criado um *dataset* composto por aquisições de dados sísmicos (obtidas a partir da reunião de tiros) sobre as quais busca-se realizar uma triagem da qualidade utilizando-se as categorias “BOA”, “MÉDIA” e “RUIM”. O dado classificado como de “BOA qualidade” é aquele que possui um nível muito baixo de ruído presente; o de “qualidade MÉDIA” possui pouco ruído, e de “qualidade RUIM” é aquele que apresenta muito ruído. Esse mesmo tipo de classificação já foi utilizado nos trabalhos de Bruno Pereira Dias (1) e Israel Almeida (2) e também foi alvo de pesquisa de Luis Felipe Müller (**?**

) e experimentos de AutoML de Ivan Pereira Dias(sem publicações).

Para realizar esta tarefa de classificação, propõe-se, nesta dissertação, a utilização de *aprendizado profundo* (*Deep Learning – DL)* por meio do método de aprendizado supervisionado. O Deep Learning é uma subárea de *Machine Learning (ML)* que se diferencia por utilizar múltiplas camadas de neurônios interligados.

Dentre os modelos classificados como de aprendizado profundo, as Redes Neurais Convolutivas (Convolutional Neural Networks – CNNs), também conhecidas como Convnets, têm se destacado por sua simplicidade e por bons resultados, sendo consideradas como o estado-da-arte para muitas tarefas. Um uso particular das CNNs se dá no campo de visão computacional, onde elas, devido ao grande poder de abstração de padrões e de lidar com grandes volume de dados, têm mostrado bons resultados.

No Capítulo 3, detalha-se o recente avanço das CNNs, contextualizando os desafios e as soluções que fizeram parte do seu processo evolutivo. Confrontando esta história, este trabalho utiliza-se do avanço das CNNs para investigar e apresentar uma arquitetura de rede neural que seja específica à tarefa de classificação de ruído sísmico.

1.1 Objetivo

O objetivo deste trabalho é aplicar técnicas de Aprendizado Profundo ao problema de classificação ruído em Reunião de Tiros Comuns, realizando experimentos dentre uma aquisição e depois buscando generalizar esses resultados para aquisições diferentes.

Primeiramente, objetiva-se replicar o experimento de Bruno Dias (1) e aplicar arquiteturas de redes neurais já conhecidas como, por exemplo, a VGG (3)(VGG 16 e VGG 19) e a Inception V3 (4) como baselines para a tarefa. Estas redes já reportaram, respectivamente, 7,3%(em 2014) e 3,6%(em 2015) de taxa de erro em uma tarefa de classificação na competição da *ImageNet Large Scale Visual Recognition Competition* (ILSVRC).

Na sequência, propõe-se investigar e utilizar uma arquitetura composta por módulos reduzidos da rede Inception V3 (4), também conhecida como Miniception. A proposta é criar variações da Miniception utilizando diversas técnicas conhecidas por melhorar seus resultados, como as técnicas de caminho residual (5), de bloco de atenção (6) e Compressão e Excitação (*squeeze and excitation* (7)) e buscar-se-á realizar um ajuste fino dos hiperparâmetros. Para a avaliação, será utilizado o F1 Global com uma validação cruzada com 10 folds.

Por fim, busca-se avaliar o classificador em uma aquisição sísmica diferente. Para isso, será desenvolvido um procedimento de *Holdout*, onde se treina com todos os dados de uma aquisição e faz uma predição em outra aquisição.

1.2 Contribuições

A contribuição deste trabalho é a proposição de uma rede neural específica para a classificação de ruído sísmico. Ao alterar arquiteturas e utilizar técnicas recentes de *machine learning*, foi alcançado o resultado de 95,58 %, que supera os obtidos com redes tradicionais, como as VGGs e a Inception treinadas na ImageNet.

Um ponto de dificuldade deste trabalho é a carência de dados, especialmente de dados anotados, que são um requisito para o aprendizado supervisionado. A situação se agrava quando a procura é por dados públicos anotados, que são uma raridade, principalmente em áreas relacionadas à campos produtores. Devido a grande competição neste setor e ao valor comercial dos dados, existe uma legislação rigorosamente restritiva para a publicação da informação. Portanto, uma das principais contribuições deste trabalho é também a técnica de aumento de dados, cuja utilização permite gerar novos exemplos, e assim melhorar a performance das modelagens.

1.3 Organização

Esta Dissertação está organizada da seguinte forma. No Capítulo 2 descreve-se a tarefa de classificação sísmica. No Capítulo 3, são apresentados os conceitos teóricos em que este trabalho se baseia, apresentando também alguns dos trabalhos relacionados à tarefa. O Capítulo 4 é dedicado à descrever a metodologia utilizada para desenvolver um preditor de qualidade sísmica. Dentre as seções desse capítulo apresentam-se a construção do dataset, as arquiteturas utilizadas como baseline, a arquitetura da rede proposta, incluindo suas respectivas variantes, e os fluxograma de avaliação. No capítulo 5 os resultados são apresentados e realizam-se comentários sobre eles. Em 6 apresenta-se o produto desenvolvido, encapsulando o melhor preditor. Por fim, no Capítulo 7 pontuam-se os resultados obtidos e apresentam-se as possíveis direções a serem seguidas para a melhoria deste trabalho.

2
Classificação de Ruídos Sísmicos

A tarefa de Classificação de Ruído é um dos processos dentro da cadeia de processamento de dados sísmicos. Para ter uma boa compreensão da aplicação da tarefa, descreve-se neste capítulo o processo de aquisição sísmica, explica-se o que é uma reunião de tiros e caracteriza-se o tipo de ruído analisado.

2.1
A Aquisição de dados Sísmicos

A aquisição de dados sísmicos pode ser feita ou no ambiente terrestre (*onshore*) ou no marítimo (*offshore*). Ambos os processos são bem semelhantes, mas neste trabalho dá-se ênfase à aquisição de dados sísmicos marítimos. Como exemplo, a Figura 2.1 ilustra o processo de aquisição de dados sísmicos em alto mar. O navio localizado em (4) reboca um canhão de ar, exposto em (1), e uma série de sensores sísmicos, chamados *hidrofones*, que estão distribuídos de forma proporcional ao longo de um cabo (3), em inglês chamado de *streamer*. Estes sensores são os responsáveis pela aquisição dos dados sísmicos. Quanto mais profundo for o terreno geológico, maior deve ser o tamanho do cabo

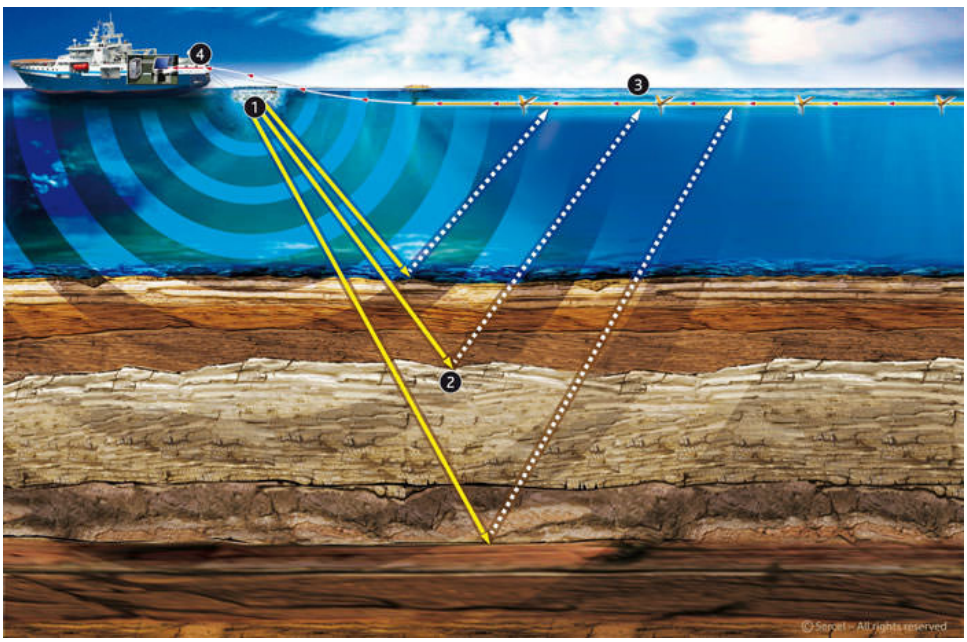

Figura 2.1: Aquisição de Dados sísmicos em alto mar.

que costuma variar de 3 a 6 km, segundo AAPG Wiki.[1] Cabos maiores são preferidos para pequisas mais relevantes, enquanto cabos menores são mais adequados a ambientes com restrição de espaço.

O procedimento padrão consiste em navegar pela região de interesse e disparar uma sequência de tiros de ondas de ar em direção ao solo marítimo (2). Estas ondas, ao alcançarem o solo, são em parte diretamente refletidas, voltando à superfície; outras sofrem refração, conseguindo atingir camadas mais profundas. O mesmo processo físico de reflexão e refração é repetido diversas vezes para outras camadas abaixo. Por fim, parte das ondas refletidas durante esse processo voltam a superfície e atingem os sensores, que captam o sinal.

2.2 Reunião de Tiros Comuns

Os dados sísmicos coletados por vários receptores oriundos de um mesmo tiro são chamados de "reunião de tiros comuns" ou *common shot gather (CSG)*. A Figura 2.2 ilustra uma reunião de tiros.

[1] https://wiki.aapg.org/Marine$_s eismic_d ata_a cquisition$

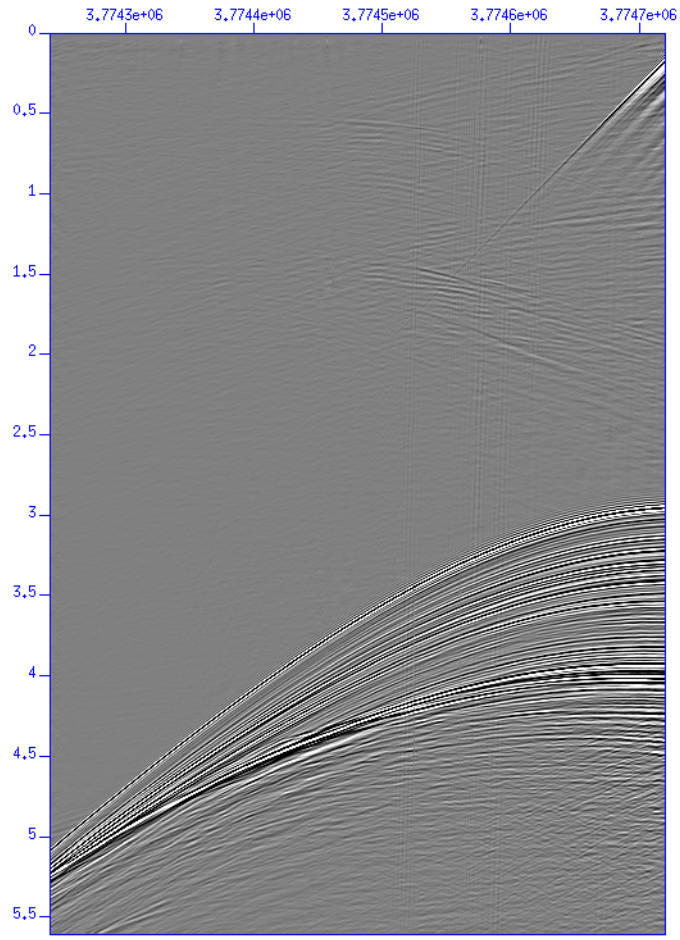

Figura 2.2: Reunião de tiros comuns com a distância *offset* em coordenada, sendo um total de 479 sensores gravados por cerca de 5,5 segundos.

Em uma reunião de tiros, tem-se o dado pré-stack, onde cada uma das colunas representa um sinal sísmico captado por um sensor. Nesta imagem marítima foram empregados um total de 479 hidrofones, onde cada um desses sensores capturou um traço sísmico. Quanto mais hidrofones forem empregados em uma aquisição, maior será a distância horizontal da representação, que é medida em quilômetros e é chamada de *offset*. Já o eixo vertical é um eixo temporal, medido em milissegundos (10^{-3} s) e indica o tempo de chegada dos eventos. Por exemplo, em casos do pré-sal, onde a distância da superfície até a camada de sal costuma ser grande, o tempo de registro deverá ser maior, para poder contemplar um sinal de elevada profundidade.

2.3 Os Tipos de Ruídos

De acordo com uma pesquisa sísmica marinha feita por Elboth et al. (8) conseguiu-se identificar cerca de 5 tipos diferentes de ruídos em reunião de

tiros. A Figura 2.3 mostra a localização destes ruídos.

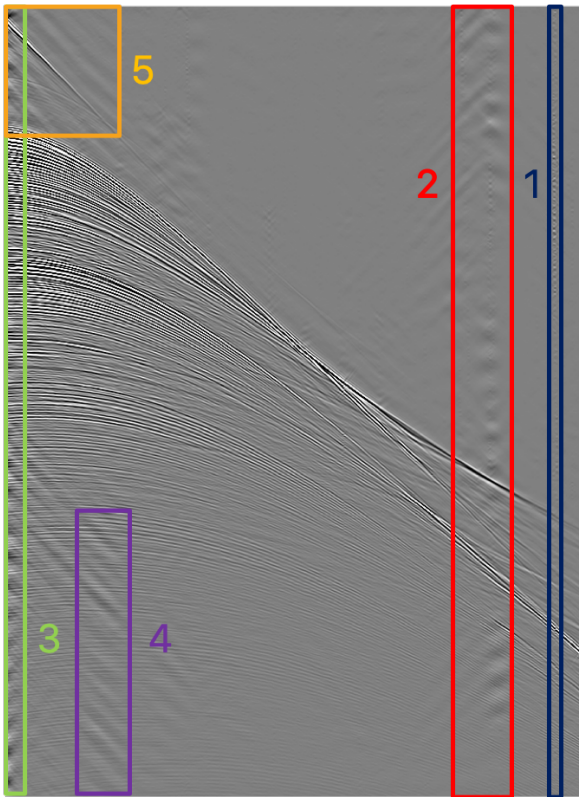

Figura 2.3: Tipos de Ruído em Reunião de Tiros. (1) Ruído da Variação da Pressão Hidrostática; (2) Ruído "Bolha"; (3) Ruído de Reboque; (4) A Interferência Sísmica; (5) Ruído de Propulsão do Navio

A Variação da Pressão Hidrostática (1) é uma das fontes de ruído e tem relação com mudança da altura da coluna d'água sob o cabo. Esta variação é causada pelo movimento das ondas e a força de compressão do cabo, que acabam gerando ondas destrutivas do sinal. Este ruído é perceptível na figura na posição 1 e tem a característica de ser bem fino.

O Ruído "Bolha"(2), em inglês chamado de *Swell Noise*, é assim chamado pois produz diversas bolhas ou inchaços na representação. Diferentemente do ruído de pressão hidrostática, o *Swell Noise* costuma ser largo e bem marcante, contaminando vários traços de forma severa, conforme visto na figura na posição 2. Existem dois mecanismos diferentes que criam este tipo de ruído. O primeiro, tem sua origem no movimento das ondas do mar, que acabam perturbando os cabos, que por sua vez geram ondas transversais. O segundo mecanismo é chamado de fluxo cruzado ou *Cross-flow*, e ocorre principalmente em ambiente de mar revolto. Este efeito ocorre quando a angulo entre o cabo e direção da onda excede 6-15°, gerando assim ondas assimétricas, que se refletem em um redomoinho.

O Ruído de Reboque (3), em inglês chamado de *Tugging/Strumming Noise*, é causado por movimentos bruscos ou vibrações na embarcação. Este ruído é mais visível nas primeiras seções do cabo e mostrado na posição 3.

A Interferência Sísmica(4) é causada pelo operação de outra embarcação no mesmo local. Este ruído pode ser visto na imagem como tiras e está localizado na posição 4 da figura.

O Ruído de Cavitação pela Propulsão do Navio (4), em inglês chamado de *Proppeller Cavitation Noise*, é causado por áreas de baixa pressão proporcionadas pela hélice do navio. Quando estas áreas de baixa pressão atingem a pressão de vapor, o líquido começa a vaporizar e formar bolhas. Esse efeito é visto na parte na esquerda superior da figura, na posição 5.

Muitos dos ruídos mencionados são retirados ou tem seu efeito reduzido com o uso de filtros de frequência temporal (9) (10) ou com filtro preditivo f-x (11). Contudo, a remoção do ruído ou o detalhamento dos tipos de ruído não é alvo desta dissertação, portanto sugere-se que consulte Elboth et al. (8) para aprofundamento no assunto.

2.4 A Definição da Tarefa

Define-se como a tarefa de classificação analisar a reunião de tiros comuns e fazer um julgamento subjetivo do nível de intensidade de ruído presente.

O diferencial deste trabalho é que ao invés de fazer uma classificação baseada nos traços, busca-se fazer uma avalização da imagem como um todo, obtendo uma perspectiva global da qualidade.

Para esta dissertação, optou-se pela classificação das representações nas três seguintes categorias:

- BOA. É aquela representação que apresenta pouco ou nenhum ruído. É considerada aceitável para produção.
- MÉDIA. Aquela que apresenta um maior nível ruído, assim necessitando de um tratamento dos dados.
- RUIM. A que apresenta muito ruído, com baixa qualidade e pouco poder de representação.

A imagem na Figura 2.4 exemplifica cada uma destas três categorias, classificadas por um especialista.

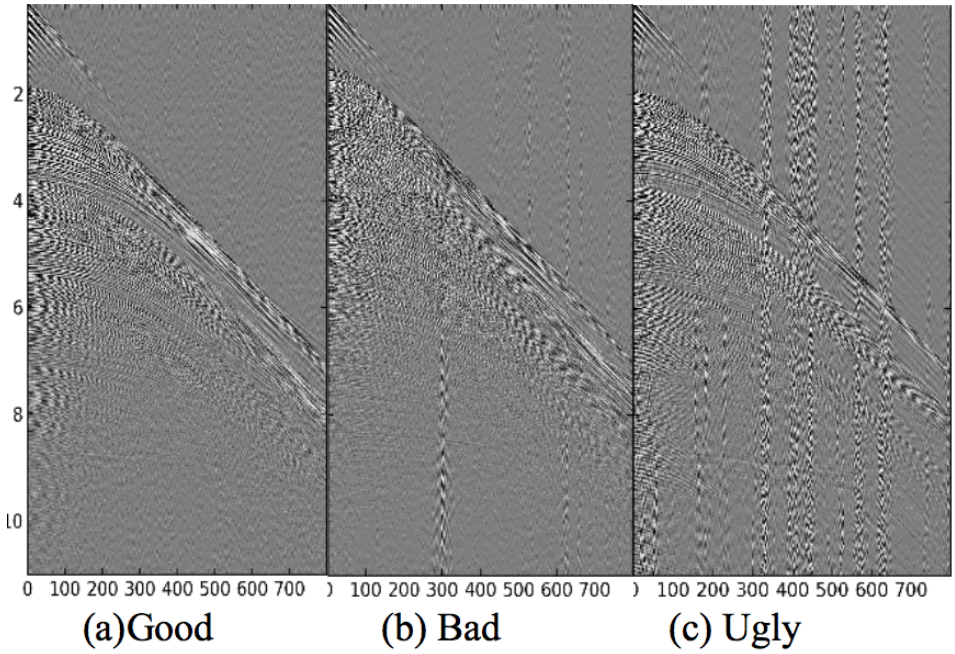

Figura 2.4: Exemplos de representações classificadas como Boas, Médias e Ruins, respectivamente, por um especialista em geofísica.

Essa classificação foi elaborada e validada em conjunto com profissionais da Petrobras. Os especialistas geofísicos julgaram essa separação interessante pois poderiam, desta forma, aceitar as representações BOAS, tratar as representações MÉDIAS por meio de filtros de frequências ou algoritmos de *denoising* e, pragmaticamente, descartar as representações RUINS dada sua raridade e dificuldade de remoção de ruído. Além disso, um especialista geofísico fez a separação de uma amostra com 1.389 imagens e classificou cada uma delas em BOA, MÉDIA e RUIM, amostras que denominamos as *golden labels*. Baseando-se nestas imagens, as outras imagens que compõem o dataset foram marcadas pelo autor desta proposta. A descrição do dataset está presente no capítulo 4, dedicado à metodologia.

3
Conceitos Teóricos e Trabalhos Relacionados

Neste capítulo descrevem-se outros artigos que se encontram na literatura referente à classificação de ruído em sismogramas. O protocolo de busca desta dissertação foi feita sobre o termo "seismogram noise classification"dentre as bases do Google Scholar e na IEE Transactions Seismic and Remote Sensing.

O problema de classificação de ruído tem uma atenção especial no contexto de Alerta Prévio de Terremotos ou *Earthquake Early Warning (EEW)*. Um sistema de Alerta de Terremotos deve de forma rápida e confiável fornecer respostas sobre eventos de terremotos e assim possibilitar que os danos do evento sejam minimizados. Contudo, conforme relatado por Meier (12), um sinal muito ruidoso pode se parecer com um sinal de terremoto e confundir o classificador, o que vai indicar mais falsos positivos. Por isso, existe um grande interesse da comunidade e vários artigos seguem essa linha, de melhorar os preditores de ruído entre dados ruidosos e de terremoto.

O artigo de Meier (13), motivado pelos avanços do Aprendizado Profundo e seus resultados promissores, avalia como classificadores modernos performam na discriminação entre sinal e ruído. O autor treinou arquiteturas como fully connected neural network(FCNN), Redes Neurais Convolucionais (CNN), Redes Neurais Recorrentes (RNN) e um modelo que combina rede generativa com Floresta Aleatória (GAN+RF) em um dataset com 374 mil sinais de terremotos e 946 mil sinais de ruído. Pelos resultados dos experimentos tanto a CNN quanto a GAN+RF obtiveram os melhores resultados. Por exemplo, a GAN+RF apresentou 99.5% de precisão e 99.3% de recall em um conjunto de validação.

Já o artigo de Wiszniowski (14) explora outra aplicação. Wiszniowski relata que muitos dos classificadores convencionais acabam performando mal quando há intensa presença de ruído. Na realização de seu trabalho na Polônia, o autor relata que conseguiu ter bons resultados com uma Rede Neural Recorrente de tempo real (RTRN) para classificar eventos sísmicos pequenos, mesmo sobre elevado ruído.

Outro aspecto da pesquisa são as técnicas de aprendizado não-supervisionado, que conseguem aprender padrões mesmo sem labels conhecidas. Os autores Kuyuk, Yildirim, Dogan e Horasan (15) aplicam um Mapa

auto-Organizacional (SOM) ao problema de classificação. A SOM é uma ferramenta para visualização e busca fazer uma redução de dimensionalidade dos dados. Mousavi (16) e Titos (17) procuraram aplicar autoencoders, que são redes neurais que buscam em uma subtarefa aprender como representar de forma mais inteligente um sismogramas. Logo, os AE geram um mapa de valores para os exemplos e depois costumam utilizar classificadores como SVM e Floresta Aleatória para formar um cluster com os dados.

Essa abordagem de aprendizado não supervisionado tem como ponto forte não precisar de dados anotados e conseguir portanto trabalhar com uma maior quantidade de exemplo do que normalmente empregado em uma técnica supervisionada. Para um dataset com 6.600 imagens, conforme utilizado nesta dissertação, o esforço para fazer a marcação manual das imagens não é tão significativo. Contudo, na posse de um dataset maior, a marcação manual fica muito custosa e a alternativa para isso é o aprendizado supervisionado, que permitirá utilizar todas as imagens sem onerar as equipes.

Fazendo também uso de aprendizado não supervisionado, Jain (18) e McBrearty (19) construiram rede neurais convolucionais para formar clusters baseados em similaridade. O primeiro autor utilizou a função de similaridade empregada na Triplet Network (20), estado da arte de aprendizado de máquina para o reconhecimento de faces, para mapear as característica do sismograma e contabilizar a similaridade entre cada um deles. Nesta função, a rede compara para cada classe o sismograma analisado com um ground-truth e um sismograma de outra classe. O desenvolvedor deve apenas no início indicar um sismograma representativo de cada classe, que o resto é feito pela função de perda. Pela análise da Curva Característica de Operação (ROC) e sob a métrica de Área abaixo de Curva (AUC) o artigo obteve 87% na estação K22, que é a estação mais ativa do dataset do USArray.

A similaridade de sismograma se difere da nossa metodologia e de outras metodologias de aprendizado não supervisionado, porque não busca, de fato, a melhor representação para os dados sísmicos e sim aproximar ou afastar os sismogramas com base em suas distâncias. A grande vantagem é que, idealmente, com apenas um exemplo de cada classe anotados pelo especialista, já é possível utilizar a similaridade e remove a fonte de erros que existe no momento da marcação das imagens. Esta é uma técnica muito intuitiva de ser utilizada e possui sistema de visualização por meio do t-sne (21), que é uma técnica de redução de dimensionalidade. Contudo, sua métricas empregadas são distintas das nossas, o que dificulta comprar os resultados.

O artigo de Paitz (22) apresenta um experimento, no qual foi aplicado uma Rede Neural Artifical (ANN) para classificar o ruído de um conjunto de

sismogramas. Os autores utilizaram um sistema binário de classificação (tem ou não tem ruído) e compararam a classificação de sismogramas realizada pela rede neural e outros 4 estudantes de sismologia. Empiricamente observa-se que existem amplas discrepâncias pelas classificações realizadas pelos humanos em oposição a uma classificação consistente feita pela ANN. O artigo conclui e dá ênfase que usar uma Rede Neural aumenta a qualidade e a reprodutibilidade da classificação.

Esta última informação corrobora para a confecção de um rede neural específica para a classificação de ruído em sismogramas conforme realizado neste trabalho. Em experimentos semelhantes nota-se também discrepâncias realizadas por humanos e procura-se sempre manter a classificação do especialista como a Golden Labels.

3.1 A Evolução das CNNs

Na evolução das CNNs, pode-se citar algumas redes importantes, como a LeNet-5, por exemplo, que foi uma rede convolucional pioneira desenvolvida por Yann LeCun em 1998 e aplicada na famosa tarefa de reconhecimento de dígitos da base de dados do Modified National Institute of Standards and Technology(MNIST). Hoje em dia, as CNNs já atingem resultados de 99.3% de acurácia (23) para o problema do MNIST.

Em 2012, a rede AlexNet, desenvolvida por Alex Krizhevsky, Ilya Sutskever e Geoffrey E. Hinton, superou significativamente o estado-da-arte e foi a campeã da competição Large Scale Visual Recognition Challenge (ILSVRC) (24) com 15.3% de erro top-5. No contexto da competição, o erro top-5 significa que os algoritmos teriam até 5 tentativas para acertar cada imagem; se mesmo assim errassem, isso contaria como um erro. [1]

A AlexNet (26) tem uma arquitetura bem similar a rede proposta por LeCun. Porém, além de mais profunda, ela apresentava mais filtros por camada, totalizando cerca de 60 milhões de parâmetros. Esta rede combinava convoluções com kernels 11x11, 5x5, 3x3, max pooling, dropout, ativação ReLU, data augmentation e SGD com momentum. Toda convolução e camada *fully connected (FC)* era seguida por uma função de ativação ReLU e a rede foi dividida em dois caminhos principais, com uma GPU Nvidia Geforce GTX 580 atuando em cada ramo.

Em 2014, uma arquitetura, chamada de VGG (3), chamou atenção na competição ILSVRC. Embora esta arquitetura não tenha sido a campeã no

[1] A ILSVRC utiliza como banco de dados a ImageNet (25), que contém cerca de 1,2 milhões de imagens genéricas, subdivididas em mais de 1000 categorias.

ano, ela se destaca por sua simplicidade de implementação, sendo a escolha principal da comunidade para extrair características de imagens. A VGG foi desenvolvida por Simonyan et al. e Andrew Zisserman, nasceu no Visual Geometry Group da Universidade de Oxford, o qual deu origem ao nome da arquitetura.

Similar a AlexNet, a VGG contém uma sequência de convoluções 3x3 seguida por um max pooling, porém esta rede apresenta bem mais filtros que as antecessoras. Totalizando 138 milhões de parâmetros a rede foi treinada em 4 GPUs por durante 2–3 semanas.

A arquitetura campeã de 2014 foi a rede Inception, também chamada por GoogleLeNet (27), pois foi desenvolvida por um time de pesquisadores do Google. A rede alcançou um erro top-5 de 6.67% e utilizou uma arquitetura semelhante a usada na LeNet, porém introduziu o conceito do módulo Inception, que mistura convoluções com diferentes tamanhos de kernels. Dada uma imagem de entrada o módulo Inception tem 4 ramificações, onde a diferenciação dos ramos é, respectivamente, uma convolução com kernel 1x1, 3x3, 5x5 e em outro ramo um max pooling 3x3. A estratégia do módulo Inception é disponibilizar kernels de diferentes tamanhos e permitir que a rede neural selecione qual kernel é o melhor para uma dada imagem. A rede Inception contribuiu para a reflexão que em imagens onde o objeto de interesse tem tamanho reduzido, kernels com tamanhos menores serão mais apropriados. Já em imagens onde o objeto de interesse é grande, kernels com tamanhos maiores serão melhores. A arquitetura da Inception é composta por 11 blocos Inceptions e o seu bloco composto diversas convoluções pequenas contribuiu para reduzir o número de parâmetros, que no total são apenas 4 milhões.

Em 2015, a rede ResNet foi a campeã da competição com um erro top-5 de 3.57% (4). Esta foi a primeira arquitetura que conseguiu superar o erro top-5 de 5.1% de um humano especialista(Andrej Karpathy) na tarefa. Desenvolvida pelo time de pesquisadores da Microsoft, a ResNet fez uso de *batch normalization* e caminhos residuais para atacar o problema de dissipação do gradiente, ou em inglês *Vanishing Gradient Problem*, que é muito presente em redes neurais profundas. Ao adicionar muitas camadas à rede, percebe-se que a atualização excessiva dos pesos da rede acaba extinguindo o gradiente, que faz consequentemente a rede parar de aprender. A ResNet propõe então fazer uso de um caminho residual para reutilizar o sinal de saída da função de ativação da camada anterior para somar no cálculo de atualização da camada seguinte. Desta forma, a ResNet é capaz de continuar aprendendo durante um treinamento por mais épocas em uma rede mais profunda.

Em 2017 foi o desfecho da competição ILSVRC, quando a tarefa foi dada

como concluída. Com um resultado de 2.36% de erro top-5, a SE ResNet, desenvolvida por pesquisadores de Oxford, foi a campeã deste ano. A rede combina o termo SE com o termo ResNet, que já foi explicada anteriormente e foi a campeã do ano de 2015. O termo SE, que significa *Squeeze and Excitation* ou Aperto e Excitação em português, é um mecanismo de atenção, que procura recalibrar os pesos dos canais utilizando uma atenção espacial. Primeiro é realizado uma operação de Global Average Pooling, que faz uma média de cada um dos canais de entrada, esse resultado serve como entrada para uma camada densa que fará a compressão dos canais, após isso o resultado volta ao shape original depois de passar por outra densa. O resultado calculado pelo bloco de excitação é então multiplicado pela saída da convolução. Todo esse processo será mais detalhado ao longo da dissertação.

4
Metodologia Proposta

Este capítulo relata a metodologia utilizada para construir um preditor de qualidade sísmica, descrevendo a aquisição e tratamento dos dados e os modelos e técnicas computacionais empregadas. O capítulo fica dividido em 6 seções, onde na primeira seção relata-se o processo de construção do dados, na segunda seção apresenta-se um modelo base para a tarefa, na terceira seção detalha-se a modelagem proposta, a quarta seção dedica-se para as variações da modelagem proposta, a quinta seção destina-se a descrever o aumento de dados empregados, e na sexta e ultima seção é diagramado a divisão do dataset e as métricas de análise dos resultados.

4.1
O Dataset

O dataset descrito a seguir é uma construção feita pela parceria entre o Departamento de Informática da PUC-Rio (Laboratórios Learn e Telemídia) e a Petrobrás – que forneceu os dados sísmicos e disponibilidade de profissionais envolvidos. De imediato, esclarece-se que as imagens expostas neste trabalho correspondem a dados reais, obtidos *in loco* e, por termos de sigilo, servem apenas para exemplificar, não podendo o dataset ser publicado nem tampouco revelado o local de sua aquisição.

Para fins desta dissertação, será utilizada uma amostra de 6.918 CSG, contendo dados de duas aquisições distintas, denominadas A e B. A Tabela 4.1 descreve a composição do dataset das aquisições A e B e suas classes.

	BOM	**MÉDIO**	**RUIM**	**TOTAL**
Aquisição A	4.410	2.086	117	6.613
Aquisição B	267	36	2	305
TOTAL	4.677	2.122	119	6.918

Tabela 4.1: Composição do Dataset de Aquisição A e B.

Os dados originais são armazenados em arquivos SEG-Y, que seguem um protocolo desenvolvido pela *Society of Exploration Geophysicists* (SEG) para armazenar dados geofísicos. Neste formato, os dados são disposto em matrizes,

onde cada linha representa um sismograma, e cada célula contém um número *float* com a medição do sensor, que tem valor aproximado entre -3.10^4 e 3.10^4. Esta informação deve ser portanto processada para transformar o SEG-Y para as imagens em escala de cinza. O código abaixo descreve este processo.

```
import os
import numpy as np
from obspy.io.segy.segy import _read_segy

def load_img(self, img_path, label):
    """
    Reads and Normalize a seismogram from the given segy file.
    :param img_path: a path to the segy file.
    :param label: a label to the segy.
    :return: seismogram image as numpy array normalized between
    0-1.
    """
    segy = _read_segy(img_path)
    traces = list()
    for trace in segy.traces:
        traces.append(trace.data)
    x = np.asarray(traces, dtype=np.float32)
    std = x.std()
    x -= x.mean()
    x /= std
    x *= 0.1
    x += .5
    x = np.clip(x, 0, 1)
    x = np.expand_dims( x, axis = 0 )
    return x.T, label
```

Listing 4.1: SEG-Y Transformation to Image

Na execução deste código, a primeira atividade realizada é a leitura do arquivo SEG-Y, que é feita com auxilio da biblioteca ObsPy. Na sequência, extraem-se os valores dos traços salvando-os em uma lista armazenada na variável traces. Esses valores são então subtraídos da média e divididos pelo desvio padrão, aproximando os dados de uma normal padrão $N(0, 1)$. Para achatar a variância, os valores são todos divididos por 10. Em sequência é somado o valor de 0,5, o que muda o ponto de centralização da informação e desloca a média de 0 para 0,5. Valores que estão fora do intervalo de 0 e 1 são clipados, ou seja, tem seu valor definido em zero, caso sejam menores que zero, e valor igual a um, caso sejam maiores que um. Esta etapa de *clip* pode diminuir a variabilidade dos dados, contudo percebe-se que é um evento raro, já que a variância foi achatada na etapa anterior. A função descrita retorna a

matriz transposta desses valores processados, que serve como dado de entrada para o processamento na rede neural.

Considerando que neste trabalho propõe-se a utilização de uma abordagem por aprendizado supervisionado, necessita-se de uma etapa adicional de atribuição de rótulos de classificação para as representações. Essa rotulação é feita de forma manual e necessita da experiência e habilidade de um especialista geofísico. Com base nesses rótulos, buscou-se classificar as demais representações de forma semelhante. Esse trabalho manual de revisão e marcação das 6.918 CSG levou cerca de 12 horas de trabalho.

Outro ponto a se ressaltar é que as reuniões de tiros, para uma mesma aquisição, podem ter tamanho variado, já que, durante uma aquisição, é possível que alguns dos sensores falhem e acabem não captando aquela porção do sinal. Como resultado, o número de colunas geradas em cada reunião varia dependendo da correta captação dos hidrofones. A figura 4.1 apresenta frequência relativa do número de sensores por imagem na aquisição A, que no total contabilizam 6.613 Imagens.

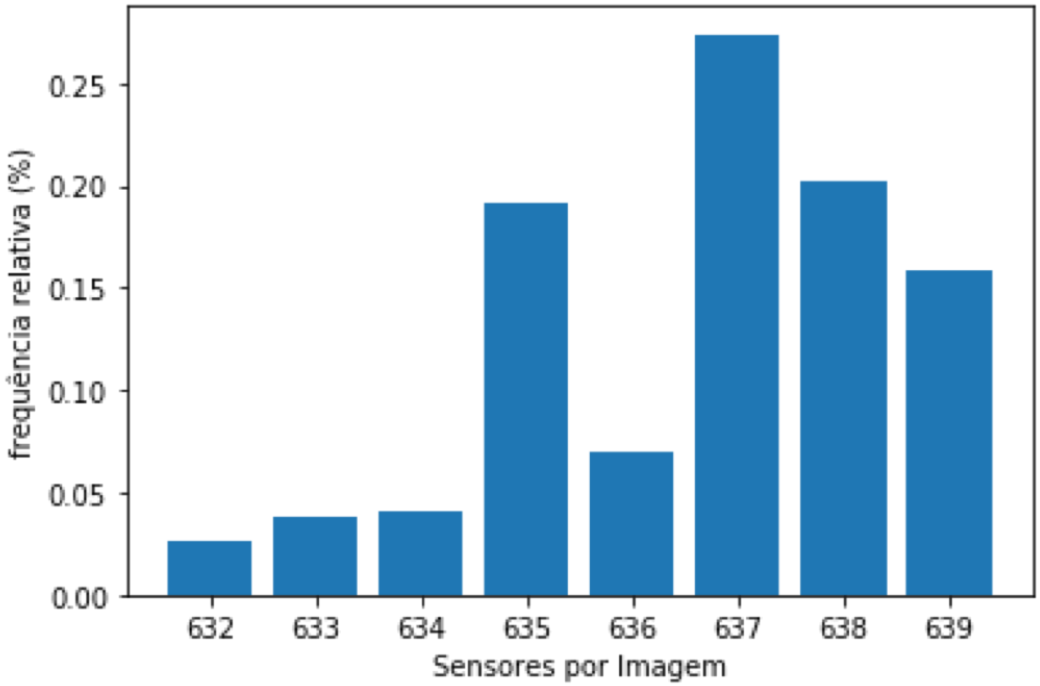

Figura 4.1: Distribuição da frequência de sensores por imagens na aquisição A.

A maior representatividade está em imagens com 637 sensores, que representa mais de 25% da distribuição, contudo percebe-se que essa distribuição varia de 632 até 639 sensores por imagem. Nesta aquisição, o tamanho vertical, que representa o tempo, é sempre constante e igual a 876, logo tem-se um conjunto de imagens com dimensão que variam dentre a faixa de 632-639 x 876. Na aquisição B o tamanho de todas as imagens é fixo em 479 x 702.

Portanto, percebe-se que o tamanho das imagens varia entre aquisições diferentes e pode variar dentre a mesma aquisição, pois o número de colunas varia de acordo com número de hidrofones ativos na captura. De forma semelhante, existem aquisições onde o número de linhas é elevado, como exemplo em aquisições do pré-sal. Neste tipo de aquisição o tempo de registro é maior, pois deseja-se investigar terrenos mais profundos. Na seção 4.3 explica-se como este problema de variação no tamanho das imagens foi abordado.

4.2 Baseline

O Baseline é uma tarefa importante para o contexto experimental de *Machine Learning*, pois tem como alvo gerar um modelo simples e um resultado satisfatório que sirva de base para comparações e críticas futuras.

O Baseline deste trabalho é desenvolvido com base na metodologia descrita no trabalho de Bruno Dias (1). São utilizadas para o baseline 3 arquiteturas de redes padrão como a VGG16, VGG19 e a Inception V3, que estão disponíveis na biblioteca Keras (28). É possível, em pouco tempo, fazer uma transfêrencia de conhecimento destas arquitetura, que foram treinadas em outros domínios, para o contexto de sismogramas.

É de conhecimento que as redes neurais, quando estudadas em camadas mais rasas, aprendem características mais rudimentares da imagem, como exemplo traços e contornos, que lembram os filtros de Gabor ou manchas coloridas. Por outro lado, quando estudadas em camadas mais profundas, as redes neurais apresentam características mais contextuais da imagem, resultando no aprendizado de padrões mais complexos. Pretende-se, com esse Baseline, não apenas conhecer qual rede apresenta melhores resultados, mas também ter a dimensão de qual camada tem a maior contribuição. É de interesse compreender se a maior contribuição vem de camadas mais rasas, com alta informação dos detalhes, ou de camadas mais profundas, com alta informação sobre contexto. A melhor camada será escolhida como o resultado da rede neural. Desta forma, o baseline proposto está exemplificado para a rede Inception V3 na figura 4.2.

A rede Inception V3 tem um total de 11 camadas, que se estruturam em uma sequência de blocos Inception seguidos por uma operação de *pooling*. O Baseline consiste em selecionar as operações de *pooling* de cada camada e aplicar, para cada uma delas, as imagens de entrada. Com isso, é possível extrair os valores representativos, que são mapeados pela rede neural camada a camada. Então, esses valores servem de entrada para um classificador *Support Vector Machine* (SVM) (29). O SVM é um algoritmo que recebe

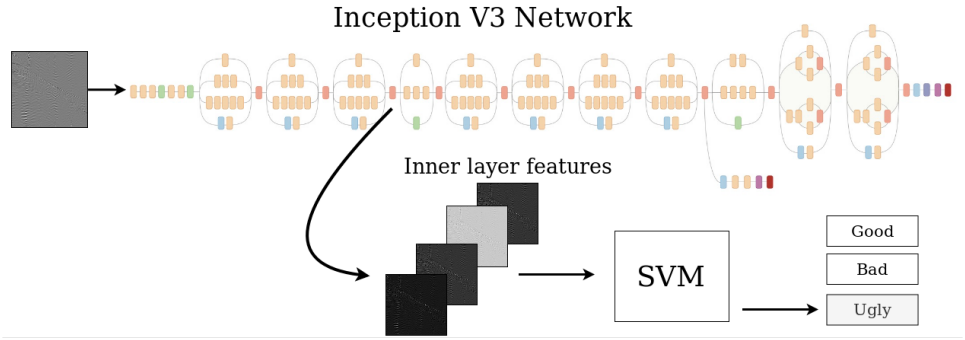

Figura 4.2: Exemplos da metodologia de Transfer Learning com uma rede Inception V3 e um classificador linear SVM.

como entrada um conjunto de dados com seus respectivos grupos rotulados e, conforme detalhado em Boser (30), busca em seu treinamento traçar uma linha de separação que maximize a distância entre os pontos mais próximos dos diferentes grupos apresentados. O hiperparâmetro de regularização C é utilizado para controlar a generalização do algoritmo. Quanto maior for o valor de C, maior será a acurácia de treinamento e, portanto, menor será o erro desse treinamento. Contudo, um C muito elevado tende a não se generalizar quando se passa a utilizar os dados de teste, já que foi feito demasiadamente ajustado para os dados de treinamento. Logo, existe um custo benefício entre a classificação perfeita nos dados de treinamento e uma boa generalização para os dados de teste.

Para este trabalho, pretende-se utilizar um valor de $C = 1$, que é o valor padrão configurado pela biblioteca do SkLearn. Além disso, por razões experimentais, propõe-se alterar o *kernel* de otimização de uma função linear para uma RBF, que realiza curvas com maior grau de liberdade.

4.3 A arquitetura Proposta - Miniception Ordinal

A escolha de tamanho de *kernel* apropriado para um bloco convolucional não é uma tarefa trivial, pois sabe-se que este tem uma relação intrínseca com o tamanho do ruído presente na imagem, que pode variar significativamente. Foi detectado empiricamente que para ruídos maiores, *kernels* de tamanho maior são mais apropriados; enquanto para ruídos de tamanho menor, *kernels* de tamanho reduzido são mais adequados (27).

Inspirado no Bloco Inception criado pela GoogleNet's, aplica-se uma arquitetura que usa na mesma camada simultaneamente *kernels* de tamanho 3x3 e *kernels* 5x5. Desta forma, a rede é obrigada a aprender, durante o treinamento, qual tamanho de convolução é mais relevante para cada representação de entrada.

O Bloco Padrão da rede proposta está apresentada na Figura 4.6.

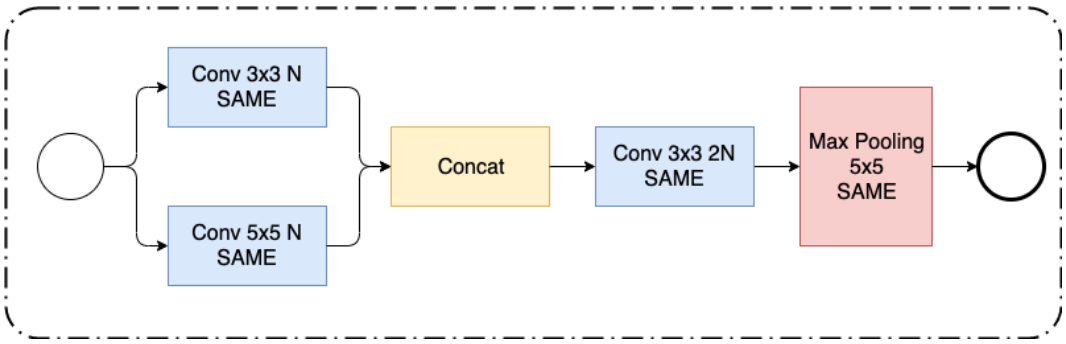

Figura 4.3: Bloco Padrão da Miniception.

Na figura encontra-se o bloco Miniception Padrão. O bloco proposto para a Miniception combina uma convolução 3x3 com outra 5x5 e concatena as suas saídas. Aplica-se então esse tensor na entrada de uma convolução 3x3, gerando novos filtros para a relação concatenada. A saída da convolução 3x3 passa por uma operação de *max pooling* 5x5 com *stride* 2, que atua na redução de dimensionalidade dos filtros.

Em todas as convoluções utiliza-se a função de ativação padrão ReLU, que atua de forma não-linear, e configura-se o stride para 1. Na figura, o atributo *SAME* indica a definição do *padding*. O *padding* igual a *SAME* busca adicionar valores zeros ao redor da imagem para obter a saída da convolução com o mesmo tamanho da imagem de entrada. O hiperparâmetro "N"indica a quantidade de filtros por convolução. Percebe-se que as duas primeiras convoluções (3x3 e 5x5) tem N número de filtros e a ultima convolução 3x3 tem 2N, ou seja, o dobro de números de filtros. O Valor N é definido de acordo com a arquitetura, que pode ser visualizada abaixo.

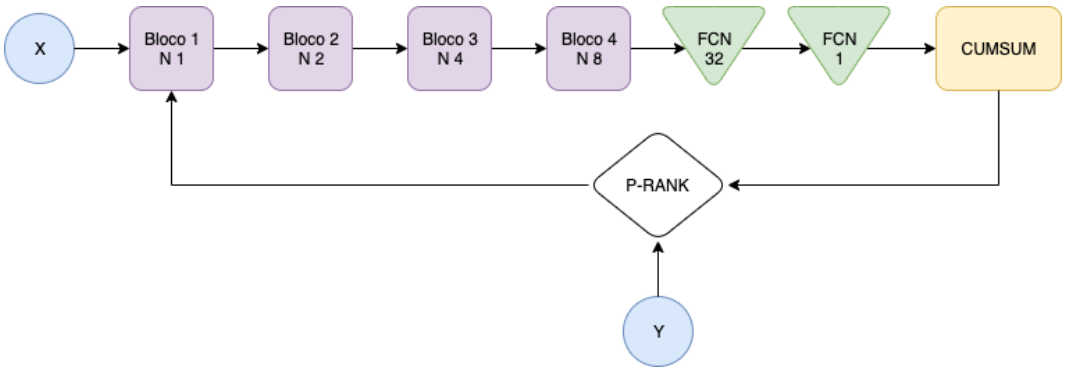

Figura 4.4: Arquitetura da Rede Miniception D ordinal. Função de perda pelo P-rank.

Na arquitetura final, o bloco Miniception é repetido por 4 vezes. O hiperparâmetro "N"é igual a 1,2,4 e 8 para cada respectivo bloco na sequência,

logo o número de filtros é dobrado a cada inserção de bloco miniception. Por fim, as últimas camadas da rede são compostas por duas *fully connected* (FC), com, respectivamente, 32 e 1 unidade, e ambas tem ativação ReLU.

A representação da rede final pode ser vista na Tabela 4.2.

Module	**tamanho de saída**
Imagem de Entrada	NONE x NONE x 1
Bloco Miniception 1 (N=1)	NONE x NONE x 2
Max Pooling 1	NONE x NONE x 2
Bloco Miniception 2 (N=2)	NONE x NONE x 4
Max Pooling 2	NONE x NONE x 4
Bloco Miniception 3 (N=4)	NONE x NONE x 8
Max Pooling 3	NONE x NONE x 8
Bloco Miniception 4 (N=8)	NONE x NONE x 16
Max Pooling 4	NONE x NONE x 16
Fully connected (ReLU)	32
Fully connected (ReLU)	1

Tabela 4.2: Arquitetura da rede Miniception Ordinal.

Ao abordar o problema de imagens de tamanho variado foram testadas diversas técnicas. Um caminho natural seria a utilização de *padding* nas imagens, o que seria equivalente a preencher a imagem com um valor neutro de pixel até completar o tamanho da maior imagem possível. Contudo, esta abordagem foi abandonada, pois há uma grande incerteza no valor de RGB do *padding* a ser introduzido. Em experimentos prévios, percebeu-se que alterar o valor de RGB utilizado do *padding* enviesa o classificador a predizer determinada classe, o que interfere diretamente na performance do classificador, pois valores que se aproximam da cor branca valorizam a predição da classe "BOA"e valores que se aproximam da cor preta valorizam a predição de classes "RUIM". Outra abordagem seria utilizar o *cropping* para reduzir a distribuição de colunas em uma aquisição. Por exemplo, poderia-se reduzir a dimensão da aquisição A para o tamanho de 632 x 876, que refletiria no menor valor de colunas da aquisição, que é 632. Este processo seria repetido para cada aquisição e o valor de entrada estaria flexível. Contudo esta abordagem também não se mostra interessante, pois ao fazer o *cropping* tem-se regiões não classificadas, o que pode degradar os resultados do classificador.

Para atacar problema de variação no tamanho das imagens, a arquitetura proposta busca utilizar a imagem na dimensão original, sem utilizar *padding* ou *cropping*. O tamanho das entradas da rede foram definidos de forma flexível,

com a dimensão de entrada é definida como NONE, e o *batch* configurado como 1. Desta forma, a rede a cada nova imagem realiza uma sequência de convoluções e faz o ajuste dos pesos utilizando o cálculo do *backpropagation*. Um ponto negativo desta estratégia é que o treinamento perde velocidade, pois não há paralelismo no processamento dos *batches*.

A função de perda utilizada é o PRank (31) (32). Assim como em problemas de classificação, o PRank busca atribuir uma das *labels* instância de treinamento. Contudo, este algoritmo tem como objeto de interesse o ranqueamento de classes, sendo adequado para situações em que há uma relação de linear entre as *labels*. Por exemplo, suponha-se que ao assistir um novo filme o usuário pode classificá-lo como "muito bom", "bom", "mediano", "ruim" e "péssimo". Nesta situação é possível perceber que um filme classificado como "muito bom" é superior a qualidade dos demais, seguindo uma ordem decrescente de classificação ("muito bom" > "bom" > "mediano" > "ruim" > "péssimo").

O PRank define limiares para a separação das classes e uma sequência de pesos W, que são utilizados para fazer o ranqueamento. O algoritmo atua de forma online, ou seja, a cada novo exemplo de entrada o PRank calcula o valor predito e recebe um valor alvo. Se for encontrado um erro entre eles, o algoritmo calcula a função de perda e atualiza os limiares e os pesos de ranqueamento. A figura a seguir 4.5 exemplifica uma iteração do PRank.

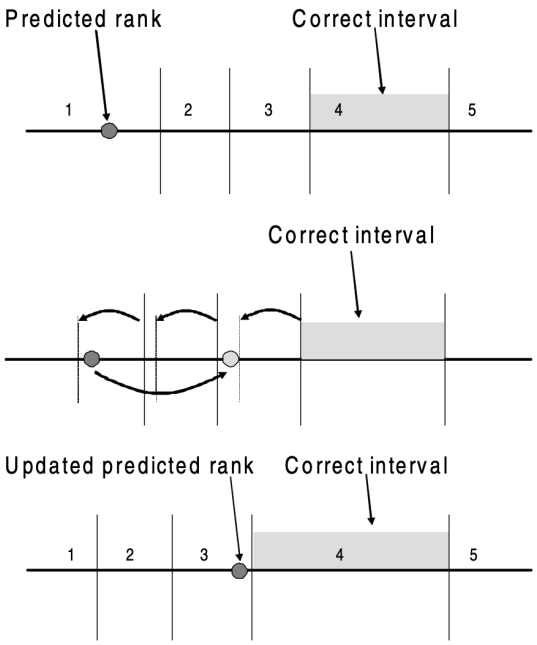

Figura 4.5: Uma ilustração da regra de atualização dos pesos. O algoritmo prevê erroneamente o valor de y = 1 ao invés de y = 4. A atualização provoca a diminuição dos limiares b1,b2,b3 em uma unidade e troca w com w + 3. Depois da atualização do valor de y =3.

4.4 Variações da Miniception

Na busca de aperfeiçoar a arquitetura, são exploradas variações do Bloco Miniception padrão. Estas variações estão apresentadas na figura 4.5.

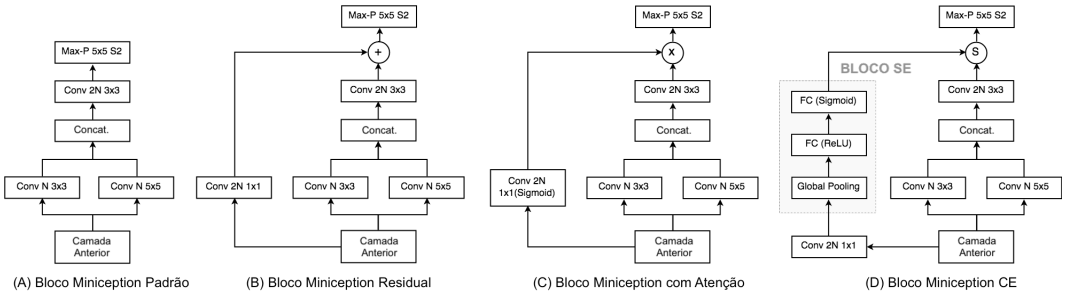

Figura 4.6: Variações do Bloco Miniception. Bloco Miniception Padrão (A), Bloco Miniception com caminho Residual (B), Bloco Miniception com mecanismo de atenção (C) e Bloco Miniception com Compressão e Excitação - CE (D).

Dentre as variações apresentada na figura 4.6 a primeira é o bloco padrão, que está detalhada em Miniception (A). A segunda variação é chamada de Bloco Residual Miniception (B), que utiliza o mecanismo de caminho residual proposto pela rede ResNet(5) para minimizar o problema da degradação do gradiente descendente em redes neurais profundas. Contudo, para poder incorporar este mecanismo de forma apropriada, é preciso adicionar uma convolução 2N 1x1 no caminho residual para que a soma com a saída do bloco convolucional 2N 3x3 tenha as mesmas dimensões.

Outra variante a ser utilizada é a do Bloco Miniception com mecanismo de atenção (C), onde propõe-se inserir um bloco de atenção. Esse mecanismo é similar ao bloco anterior, já que possui um caminho residual, com um bloco convolucional. Porém, neste caso, propõe-se a troca da função de ativação de uma ReLU para uma sigmoide, que realiza uma multiplicação elemento por elemento.

A última variante proposta é usar a rede Miniception com a Compressão e Excitação (CE), que incorpora o mecanismo de atenção do *Squeeze and Excitation*. O CE é uma técnica que procura melhorar a qualidade da representação da rede pelo aprendizado de informações globais dos canais e enfatizando de forma dinâmica características informativas.

4.5 Aumento de Dados

O aumento de dados, em inglês chamado de *Data Augmentation*, é uma técnica utilizada no Aprendizado de Máquina para produzir exemplos artifici-

ais. O principal intuito desta técnica é aumentar o conjunto de treino e permitir que as redes neurais possuam mais exemplos significativos para aprender. Como consequência deste aumento de dados, espera-se que a performance seja melhorada.

Para gerar um aumento de dados para a Reunião de tiros comuns inspirou-se em uma técnica utilizada na geofísica, inclusive presente em livros do domínio (11), que se chama *Normal Moveout (NMO) correction*.

A *Normal Moveout (NMO) correction* é uma transformação matemática, que faz parte do processo de empilhamento de *common mid-point(CMP)*. O processo de empilhamento de CMP é aplicado para suprimir o ruído de reflexões múltiplas e fazer a análise de velocidade. Seu procedimento consiste de três etapas: *CMP binning*, *Normal Moveout (NMO) correction*, e o empilhamento dos traços corrigidos pela NMO. Por meio da primeira etapa, o *CMP binning* é a técnica responsável por produzir uma representação CMP. Conforme ilustra a figura 4.7, o CMP consiste de uma reunião de tiros, onde existe um ponto de reflexão em comum.

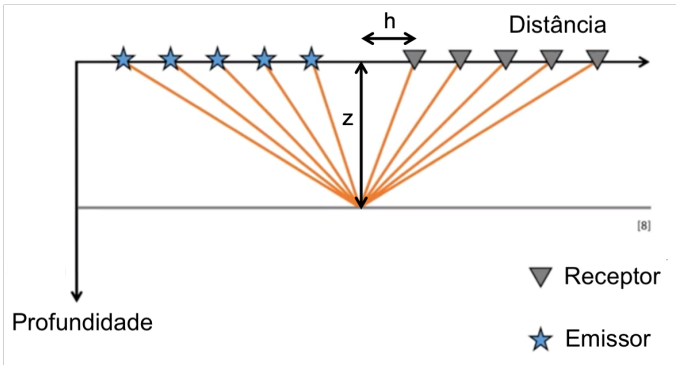

Figura 4.7: Commom Mid-Point (CMP), diversos emissores que compartilham um mesmo ponto de reflexão. Esta imagem é utilizada para a correção *Normal Moveout(NMO)*.

Na próxima etapa, a correção NMO busca alinhar os eventos de reflexões. Para facilitar os cálculos, supõe-se um modelo *layer-cake*, ou seja, um terreno plano e horizontal e também supõe-se que os dados estejam devidamente transformados em uma CMP. Começa-se a explicar a geometria da figura 4.7 pela variável "z", que representa a distância entre a superfície e o ponto no terreno de reflexão comum, a variável "h" representa a distância *offset* entre o CMP e o receptor ou emissor, e a variável "a" representa a distância entre o *commom mid-point* e o receptor ou emissor.

Pela imagem percebe-se que a relação pitagórica pode ser aplicada, pois "a" é a hipotenusa e está oposta ao ângulo perpendicular formado pelos catetos

"h" e "z". Nota-se também que "h" é a metade da distância entre o receptor e o emissor.

$$a^2 = h^2 + z^2 \tag{4-1}$$

Multiplicando-se dos dois lados a equação (4-1) por 4 chega-se ao seguinte:

$$(2a)^2 = (2h)^2 + (2z)^2 \tag{4-2}$$

Mas sabe-se da física que a distância é igual a velocidade multiplicada pelo tempo, então substituindo (4-3) em (4-2), chega-se em (4-4).

$$\Delta s = vt = 2a \tag{4-3}$$

$$(vt)^2 = (2h)^2 + (2z)^2 \tag{4-4}$$

Isolando-se o tempo na equação (4-4), obtém-se a equação (4-5).

$$t = \frac{2}{v}(h^2 + z^2)^{1/2} \tag{4-5}$$

Conforme pode-se perceber, o resultado para o *offset* zero está na equação (4-6)

$$t_0 = t(h = 0) = \frac{2}{v}z \tag{4-6}$$

Finalmente chega-se a equação do NMO (4-7) ao fazer a variação de tempo em relação ao *offset*.

$$\Delta t = t(h) - t_0 = \frac{2}{v}[(h^2 + z^2)^{1/2} - z] \tag{4-7}$$

Trocando z da equação (4-6) por t_0 em (4-7) obtém-se:

$$\Delta t = [(\frac{2h}{v})^2 + t_0^2]^{1/2} - t_0 \tag{4-8}$$

A transformação aqui apresentada não é exatamente a mesma proposta pela NMO, pois ao invés de fazer a transformação no eixo do offset "h", é proposto que se faça a mudança no eixo temporal (x). O código utilizado para o data augmentation está descrito abaixo.

```
import cv2

def augment(image_path, v, To):
    picture = cv2.imread( image_path )
    newImage = picture.copy( )

    x = np.array( [float( i ) for i in range( picture.shape[0] )] )
    transform = np.ceil( np.sqrt( (2x / v) ** 2 + To ** 2 ) - To )
```

```
9     for column in range( picture.shape[1] ) :
10      for line, transformation in enumerate( transform ) :
11        newImage[line, column] = picture[int( transformation ),
      column]
12    return newImage
```

Listing 4.2: Código Python do Augmentation desenvolvido

Nas duas primeiras linhas do código é feita a leitura e a cópia da representação. Após isso é gerado uma lista "x" contendo uma numeração sequencial (1, 2, 3, ..., C-2, C-1, C) de todos os pixels presente em uma coluna C. Por exemplo, se uma imagem tem uma altura de 680, o valor de C vai ser igual a 680 e a lista x igual a (1, 2, 3, ..., 678, 679, 680). Com base na lista "x", é calculada a transformação matemática, expressa pela variável *transform*, que realiza o mapa de transformação das posições dos pixels. Esta transformação segue uma função hiperbólica com a concavidade para cima e centrada em 0, logo, valores muito próximos a zero, devem ter valores baixos, enquanto valores distante de 0 devem ter valores maiores. Além disso, os valores quebrados são arredondados para um valor acima, por exemplo, com um valor de "v" igual a 1 e "To" igual a 100, os valores calculados seriam (1, 1, 1, 1, 2, 2, 3, ..., 586, 587, 588). Pode-se notar que a transformação repetiria a primeira célula da coluna nas próximas quatro posições e depois repetiria a segunda célula original nas posições 5 e 6 do espaço mapeado, assim por seguinte até chegar a célula a ultima célula, que seria ocupada pela célula 588 da imagem original. Com isso, percebe-se que existe uma repetição de valores na parte superior da imagem e que o final da nova representação tem o conteúdo deslocado para baixo. A transformação gerada é replicada para todas as colunas e a nova imagem retornada na função.

A figura 4.8 exemplifica a transformação da fórmula.

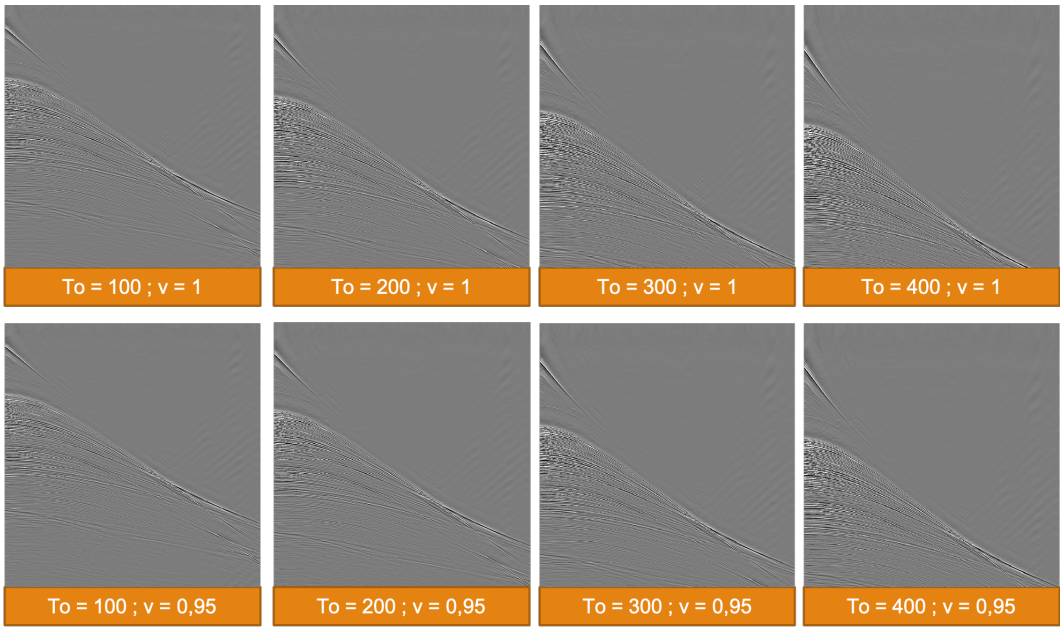

Figura 4.8: Dados Sintéticos da Transformação.

No eixo horizontal da imagem observa-se a alteração do parâmetro To entre 100 e 400, já no eixo vertical observa-se a variação do parâmetro v entre 0.95 e 1.0. Na primeira linha, ao manter o parâmetro v constante e variar apenas o To, percebe-se que a curva típica da CSG é transladada para baixo com o aumento do valor de To. Por sua vez, o parâmetro v é responsável por rotacionar a curva típica da CSG no sentido anti-horário, então ao comparar as imagens no eixo vertical, mantém-se o parâmetro To constante e varia apenas o v. Quanto menor for o valor de v, maior será a rotação da curva.

É preciso relatar que o procedimento descrito provoca uma distorção no espectro de frequências dos dados, pois arredonda-se os valores para o inteiro mais próximo. Na seção de Trabalhos Futuros7.3 sugere-se uma nova metodologia para melhorar a transformação descrita.

Para fins deste trabalho, foram geradas um total de 17 transformações para cada imagem escolhida, sendo realizado duas operações de flip(sem flip e com o flip horizontal), com a permutação dos 4 parâmetros de tempo To(100, 200, 300, 400) e com a permutação de 2 velocidades vo (0.95 e 1.0), incluindo por fim a imagem original.

4.6 Fluxograma da Metodologia de Avaliação

A metodologia de avaliação pode ser dividida em duas grandes partes. A primeira parte é uma validação cruzada para o Baseline e a rede Miniception, e a segunda é um aplicação de um *Holdout* com o treinamento em uma aquisição sísmica e o teste em outra aquisição.

A primeira etapa de Validação Cruzada é feita para analisar qual o melhor modelo, checar os melhores hiperparâmetros e definir políticas de aumento de dados. Portanto, todas as etapas referente a prototipação e experimentação da rede neural são realizadas pela validação cruzada. Ao passo que se obtém uma boa configuração de rede, avança-se para a etapa de *Holdout*, que é utilizada como teste final para a rede. Nesta etapa, a rede é treinada em uma aquisição sísmica A e avaliada sobre uma aquisição sísmica B, que ainda não foi utilizada em nenhum momento.

O processo de Validação Cruzada está resumido na Figura 4.9.

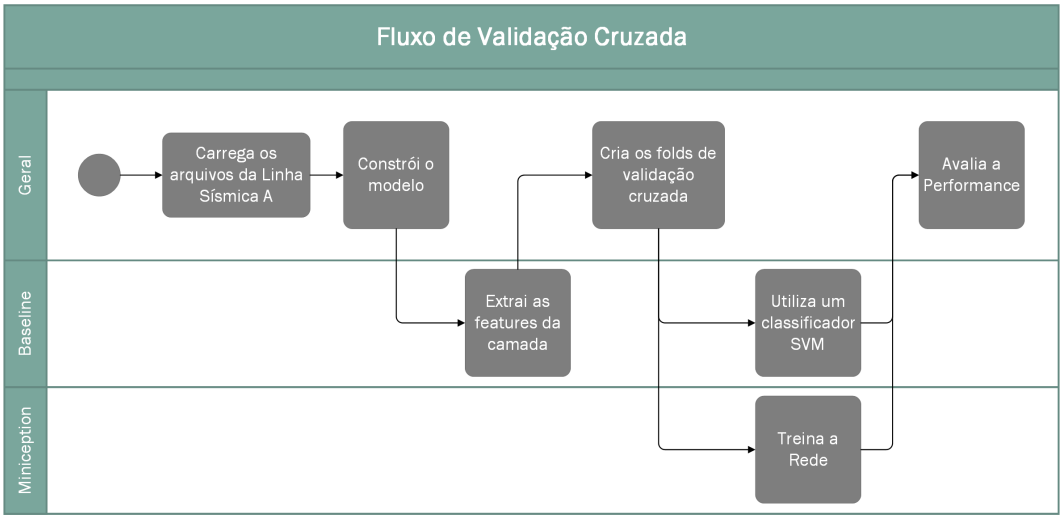

Figura 4.9: Fluxograma de Validação Cruzada para o Baseline e para Rede Miniception.

Neste fluxograma a primeira linha (Geral) representa as operações que são comuns as redes do Baseline e Miniception e, portanto, compartilham o mesmo código de programação. A segunda e terceira linha têm as operações exclusivas a cada arquitetura.

Pode-se perceber que ambos os modelos tem seu início marcado pelo carregamento dos arquivos e pela construção do modelo. Na etapa de carregamento as imagens são processadas e transferidas para a memória RAM junto com suas respectivas *labels*. A construção do modelo varia de modelo para modelo, pois no baseline os modelos são importados da biblioteca Keras, já na Miniception o modelo e as variantes são construídas manualmente pela biblioteca do Tensorflow.

Para classificar o Baseline existe uma etapa adicional de extração das características ou *features* de cada camada. Nesta etapa, o algoritmo busca retirar uma representação rica mapeada pela rede neural.

Após isso existe a criação dos *folds* da validação cruzada, que é feita pelo algoritmo *KFold* do SkLearn.

Para cada um dos *folds*, a informação é enviada para um classificador. No caso do Baseline, basta apenas executar um SVM nas representações e *labels* informadas. Já no Rede Miniception, todo o treinamento precisa ser realizado para então disponibilizar a classificação disponível no *soft-max*.

Ambos os modelos são avaliados pelo mesmo código, que gera as métricas de f1 global; f1, *recall* e *precision* de cada classe; as listas e os desvios padrões de f1, *recall* e *precision*; assim como a acurácia de treinamento e validação; a duração e horário de execução do código. Todas essas informações são salvas e depois analisadas em um notebook realizado no Jupyter.

O processo de *Holdout* está apresentado na figura 4.10.

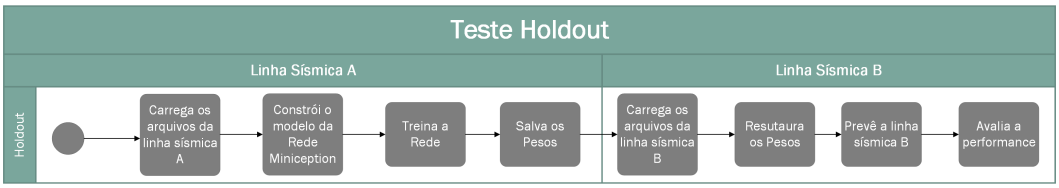

Figura 4.10: Fluxograma para avaliação pelo método *Holdout* entre duas linhas sísmicas distintas.

No *Holdout* busca-se fazer um último teste para a rede Miniception.

Para isso, carrega-se as imagens e *labels* de uma linha sísmica A, que contém 6.613 representações de sismogramas e constrói-se o modelo da Miniception.

Diferentemente da Validação Cruzada, que subdivide os dados em treino com 81%, validação com 9% e teste com 10%, no método de *Holdout* o modelo é treinado com 100% dos dados, ou seja, utiliza todas as 6.613 imagens disponíveis na aquisição sísmica A. O impacto direto deste método é que o modelo tende a ter um ganho de performance devido ao aumento do número de exemplos.

A rede então é treinada na aquisição A e a inferência realizada na aquisição B.

Por fim os valores são preditos e os resultados avaliados. Uma matriz de confusão é desenhada para a avaliação final.

5
Experimentos

Neste Capítulo, avalia-se a efetividade das arquiteturas propostas na seção anterior para a classificação de ruído em reunião de tiros comuns.

Na primeira seção reporta-se os resultados obtidos com os baselines. Utiliza-se as redes VGG 16, VGG 19, Inception V3 como *backbones* para a extração de características da imagem e aplica-se um separador SVM em cada camada da rede.

Na Segunda Seção apresenta-se os resultados da arquitetura proposta, a rede Miniception, detalhando a performance de cada variante. Ainda nesta seção, apresenta-se o efeito marginal do acréscimo de filtros convolucionais na melhor variante.

Na terceira seção resume-se os resultado para que o leitor possa comparar a evolução dos resultados tanto da baseline como da miniception dentre uma mesma aquisição.

Na quarta seção apresenta-se os resultados do aumento de dados. É analisado o efeito marginal do aumento de números de exemplos dentro da aquisição A.

Na última seção mostra-se os resultados da rede em uma outra aquisição. É feito um *Holdout* utilizando os dados da aquisição A para treinamento e feito uma predição para outra aquisição B. Os resultados são analisados e é gerada uma matriz de confusão.

5.1
Baseline

O baseline é um modelo simples, que pode ser realizado rapidamente, e gera um resultado preliminar para a tarefa. Utilizou-se para isto, um processo de *transfer learning* com extração de *features* em cada camada das redes VGG 16, VGG 19 e Inception V3.

Na tabela 5.1 está exposto o resultado da VGG 16. As camadas estão ordenadas em ordem crescente de semântica, onde 0 é a camada mais rasa, com menor semântica e 4 é a camada mais profunda com maior semântica.

Bloco	**F1-G (%)**	**F1-B (%)**	**F1-M (%)**	**F1-R (%)**
0	83.03 ± 0.45	91.8 ± 0.09	66.58 ± 0.29	45.95 ± 0.93
1	88.37 ± 0.24	94.20 ± 0.16	78.21 ± 0.42	50.02 ± 1.34
2	**88.99** ± 0.30	**95.08** ± 0.13	**81.48** ± 0.35	49.96 ± 0.99
3	74.93 ± 0.46	87.99 ± 0.05	48.71 ± 0.38	**50.61** ± 1.36
4	36.51 ± 6.62	47.98 ± 12.39	14.39 ± 6.95	0.36 ± 0.34

Tabela 5.1: Resultado do F1 Global (G) para as classes BOM (B), MÉDIO (M) e RUIM (R) com SVM e blocos da VGG 16

Conforme pode-se perceber, o melhor resultado da tabela está no segundo bloco de convolução, pois este apresenta o maior f1 score global (F1-G), com o valor de 88.99%. Além disso, percebe-se que a segunda camada tem também o melhor resultado para F1 BOM (F1-B) e F1 MÉDIO, perdendo apenas em F1 RUIM (F1-R) para a terceira camada.

Portanto, pode-se notar que as camadas intermediárias tiveram o melhor resultado para a Rede VGG16, evidenciando que *features* com maior semântica contribuíram para melhorar o resultado. Porém, as últimas camadas, que são as mais ricas em semântica, não exibiram bom desempenho. Uma possível explicação pode ser o fato das imagens empregadas não serem imagens naturais mas sim um agregado de sinais, que pode ser compreendido melhor pela decomposição em traços e aspectos mais rudimentares da visão computacional. Portanto, a rede neural, em camadas inciais ou no máximo em camadas intermediárias, já consegue extrair a informação necessária. Este mesmo efeito também ocorre para os demais baselines aqui descritos.

De forma similar o mesmo efeito aconteceu para a rede VGG19, que pode ser vista na tabela 5.2. As camadas estão ordenadas de forma crescente de semântica, onde 0 é a camada mais rasa, com menor semântica e 4 é a camada mais profunda com maior semântica.

Bloco	**F1-G (%)**	**F1-B (%)**	**F1-M (%)**	**F1-R (%)**
0	83.06 ± 0.46	91.86 ± 0.08	66.58 ± 0.28	45.46 ± 0.86
1	87.76 ± 0.38	93.94 ± 0.21	76.87 ± 0.67	48.78 ± 1.01
2	**90.82** ± 0.29	**95.64** ± 0.18	**83.10** ± 0.44	47.21 ± 1.53
3	81.49 ± 0.58	90.8 ± 0.13	63.59 ± 0.64	**50.14** ± 0.88
4	36.51 ± 6.62	47.98 ± 12.39	14.39 ± 6.95	0.36 ± 0.34

Tabela 5.2: Resultado do F1 Global (G) e para as classes BOM (B), MÉDIO (M) e RUIM (R) com SVM e blocos da VGG 19

Assim como no VGG16, a VGG19 teve seu melhor resultado na segunda camada convolucional, melhorando o f1 score global (F1-G) para 90.82%. Novamente os melhores resultados para F1 BOM (F1-B) e F1 MÉDIO se concentram na segunda camada,com exceção do F1 RUIM (F1-R) que ficou na terceira camada. No geral, os resultados da VGG19 foram superiores aos da VGG16, com destaque para o f1 score global (F1-G) e F1 MÉDIO, que tiveram um aumento de 2%.

Por fim, o resultado da InceptionV3 pode ser vista na tabela 5.3. As camadas estão ordenadas em ordem crescente de semântica, onde 0 é a camada mais rasa, com menor semântica e 10 é a camada mais profunda com maior semântica.

Bloco	F1-G (%)	F1-B (%)	F1-M (%)	F1-R (%)
0	**86.02** ± 0.30	**93.23** ± 0.17	**73.07** ± 0.55	**45.56** ± 1.45
1	85.54 ± 0.40	92.99 ± 0.17	72.05 ± 0.60	45.40 ± 1.10
2	81.01 ± 0.52	90.89 ± 0.08	62.13 ± 0.53	45.22 ± 1.3
3	83.21 ± 0.38	91.98 ± 0.14	66.86 ± 0.56	44.35 ± 1.14
4	81.5 ± 0.45	91.23 ± 0.14	63.04 ± 0.59	43.77 ± 1.05
5	81.09 ± 0.39	90.98 ± 0.11	62.25 ± 0.58	44.36 ± 1.52
6	77.92 ± 0.51	89.63 ± 0.04	55.07 ± 0.47	44.19 ± 1.51
7	76.78 ± 0.49	89.16 ± 0.0	52.41 ± 0.40	44.59 ± 1.50
8	83.05 ± 0.41	91.99 ± 0.11	66.36 ± 0.48	43.58 ± 1.19
9	76.10 ± 0.53	89.06 ± 0.0	50.64 ± 0.44	41.46 ± 1.32
10	84.42 ± 0.38	92.73 ± 0.12	69.22 ± 0.51	42.68 ± 1.19

Tabela 5.3: Resultado do F1 Global (G) para as classes BOM (B), MÉDIO (M) e RUIM (R) com SVM e blocos da Inception V3

De forma distinta das Redes VGGs, a InceptionV3 concentrou todos os seus resultados na primeira camada convolucional. Com um resultado de f1 score global (F1-G) para 86.02% essa arquitetura não se saiu melhor que as anteriores.

Os melhores resultados se encontrarem na primeira camada evidencia que a rede optou por *features* rudimentares para fazer a classificação.

5.2 Miniception

A rede Miniception é uma arquitetura engenheirada manualmente para a tarefa de classificação de ruído sísmico. Durante sua elaboração, 3 variantes foram desenvolvidas e testadas. Os resultados das variantes da Miniception podem ser vistos na tabela 5.6.

Modelo	**F1-G**	**F1-B**	**F1-M**	**F1-R**
Miniception CE (D)	**93.84** ± 0.25	**96.58** ± 0.24	90.51 ± 0.55	**50.26** ± 7.24
Miniception com bloco atenção (C)	91.09 ± 1.71	95.2 ± 0.73	85.27 ± 1.28	40.13 ± 7.72
Miniception Residual (B)	93.76 ± 0.21	96.55 ± 0.29	**90.52** ± 0.55	46.88 ± 7.14
Miniception Padrão (A)	92.90 ± 0.39	96.00 ± 0.45	89.20 ± 0.88	42.60 ± 9.19

Tabela 5.4: Comparação do resultado do F1 Global entre as variantes da rede Miniception

Pelos resultados, é possível perceber que a variante Miniception Residual(B) e a Miniception CE(D) obtiveram os melhores resultados de F1-Global, se destacando em relação as demais variantes. Contudo, é possível perceber que o resultado de F1-Global das variantes B e D é próximo quando considera-se o desvio padrão, com uma leve vantagem para a variante Miniception CE, que obteve 93.84%. Um destaque positivo da variante D é em relação ao F1 da classe Ruim, que teve uma média de 50.26%.

Considerando os argumentos acima, a rede Miniception CE foi escolhida como a melhor dentre as variantes , à partir deste momento, todos os experimentos são feitos utilizando essa variante.

A próxima experimentação realizada é a variação do parâmetro α de 1 até 10, Este parâmetro é um multiplicador dos números de filtros utilizados nas camadas convolucionais, cuja utilização com o valor igual a 1 reproduz a arquitetura padrão e a utilização com o valor 10 multiplica em 10 vezes o número de filtros em cada camada convolucional.

		F1-W	F1-G	F1-B	F1-U
α	**1**	93.84 ± 0.25	96.58 ± 0.24	90.51 ± 0.55	50.26 ± 7.24
	2	94.11 ± 0.21	96.68 ± 0.29	90.82 ± 0.57	55.72 ± 4.86
	3	94.38 ± 0.26	96.88 ± 0.30	91.29 ± 0.65	55.15 ± 5.19
	4	94.42 ± 0.32	96.94 ± 0.39	91.32 ± 0.85	54.83 ± 5.35
	5	94.37 ± 0.34	96.88 ± 0.26	91.20 ± 0.60	56.64 ± 5.84
	6	94.60 ± 0.25	97.09 ± 0.24	91.73 ± 0.43	51.92 ± 6.17
	7	94.61 ± 0.36	97.14 ± 0.31	91.71 ± 0.67	51.15 ± 8.10
	8	**94.91 ± 0.29**	**97.27 ± 0.21**	**92.08 ± 0.51**	**56.01 ± 5.59**
	9	94.52 ± 0.33	97.16 ± 0.34	91.34 ± 0.95	51.80 ± 7.09
	10	94.59 ± 0.21	97.05 ± 0.29	91.62 ± 0.75	54.70 ± 5.49

Tabela 5.5: Resultados do F1 Global para a rede Miniception com diferentes α, que são multiplicadores do kernel.

5.3 Resultados do Modelo

Para resumir os experimentos, a próxima tabela 5.6 mostra os resultados das arquiteturas estudadas, tanto do baseline quanto da Miniception.

Modelo	F1-G (%)	F1-B (%)	F1-M (%)	F1-R (%)
Miniception CE-8-99	**94.91** ± 0.29	**97.27** ± 0.21	**92.08** ± 0.51	**56.01** ± 5.59
Miniception CE-1-99	93.84 ± 0.25	96.58 ± 0.24	90.51 ± 0.55	50.26 ± 7.24
SVM+VGG19	90.82 ± 0.29	95.64 ± 0.18	83.10 ± 0.44	47.21 ± 1.53
SVM+VGG16	88.99 ± 0.30	95.08 ± 0.13	81.48 ± 0.35	49.96 ± 0.99
SVM+InceptionV3	86.02 ± 0.30	93.23 ± 0.17	73.07 ± 0.55	45.56 ± 1.45

Tabela 5.6: Tabela Resumo da comparação de F1 Global entre os modelos

Pela tabela, observa-se que a arquitetura Miniception 8-99 é a melhor dentre as listadas, pois além de apresentar o melhor F1 Global com 94.91%, também se destaca devido ao melhor resultado de 50.26% para o F1 da classe ruim. É possível perceber que a variante Miniception 1-99, que possui menor número de filtros, fica em segundo lugar no ranking, seguida adiante pelos resultados das redes baseline. Dentre os baselines se destacam as redes

VGGs, com a VGG19 (90.82%) e VGG16 (88.99%), seguidas pela arquitetura InceptionV3 com 86.02%.

5.4 O Aumento de Dados

Buscando ainda melhorar a performance sobre a aquisição A, neste momento é aplicado um aumento de dados conforme descrito na seção 4.5 do Capítulo de 4. Neste experimento é realizado uma validação cruzada de 10 folds utilizando a arquitetura Miniception CE-8-99 e o aumento de dados sobre a aquisição A. Em cada fold de treinamento são selecionadas, de forma aleatória, N imagens, que são aumentadas pela técnica computacional e adicionadas ao conjunto de treino. Cada imagem escolhida sofre um total de 17 transformações, que incluem o flip horizontal e a permutação dos parâmetros To(100, 200, 300, 400) com vo (0.95 e 1.0).

Imagens	**F1-G (%)**	**F1-B (%)**	**F1-M (%)**	**F1-R (%)**
0	94.91 ± 0.29	97.27 ± 0.21	92.08 ± 0.51	56.01 ± 5.59
50	94.96 ± 0.27	97.18 ± 0.19	92.19 ± 0.55	60.65 ± 5.53
100	95.14 ± 0.28	97.57 ± 0.20	92.08 ± 0.51	61.37 ± 5.67
150	95.37 ± 0.28	97.34 ± 0.21	94.20 ± 0.47	61.68 ± 5.57
200	**95.58** ± 0.28	**98.23** ± 0.18	**94.25** ± 0.48	**62.23** ± 5.49
250	95.23 ± 0.30	97.87 ± 0.22	93.93 ± 0.51	61.01 ± 5.60
300	95.07 ± 0.29	97.65 ± 0.20	93.58 ± 0.51	60.49 ± 5.57
350	94.89 ± 0.32	97.71 ± 0.26	92.77 ± 0.51	60.13 ± 5.58
400	94.67 ± 0.31	97.53 ± 0.24	92.08 ± 0.78	58.94 ± 6.03

Tabela 5.7: Tabela do efeito de aumento de número de imagens de treinamento sobre o F1 Global.

Com base nos valores, é possível notar que com o aumento de número de imagens artificiais o resultado de F1-Global tende a aumentar, chegando até

um ponto de inflexão, quando percebe-se que o resultado começa a decair. Pela análise, percebe-se que o melhor resultado para F1-Global está com o aumento de 200 imagens (95.58%), que também se destacou em outros quesitos, com destaque para o F1-R (62.23%).

5.5 Generalização entre aquisições

Na Tabela 5.8, mostra-se o resultado da rede Miniception-8-99 com $\alpha = 8$ no conjunto de teste.

F1-G (%)	**93.56**		
	F1 (%)	***Recall* (%)**	**Precisão (%)**
BOM	95.94	92.73	99.39
MÉDIO	86.80	93.82	80.76
RUIM	28.57	100.0	16.67

Tabela 5.8: F1-G, F1, *Recall* e Precisão para *BOM* (B), *MÉDIO* (M) e *RUIM* (R) da Miniception-8-99 com $\alpha = 8$ no conjunto de *Holdout*

O modelo alcançou um resultado de F1-Global de 93.56%, F1-Bom de 95.94%, F1-Médio de 86.80% e F1-Ruim de 28.57%. O modelo apresenta um bom resultado para F1-Bom e também mostra valores altos de *Recall*, produzindo um *Recall*-Bom de 92.73%, *Recall*-Médio de 93.82% e um *Recall*-Ruim de 100%. Contudo, nota-se uma diminuição em F1-Médio e F1-Ruim, devido a baixa precisão da classe Média de 80.76% e a precisão da classe Ruim de 16.67%.

A Figura 5.1 apresenta a matriz de confusão.

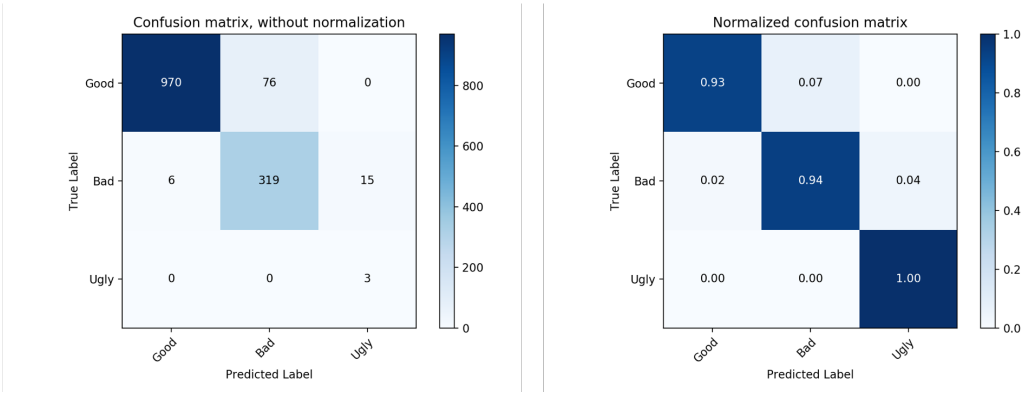

Figura 5.1: Esquerda: Matriz de Confusão para o conjunto de Teste; Direita: Matriz de Confusão com normalização para o conjunto de Teste.

O modelo identificou corretamente 970 imagens como “BOAS”, 319 imagens como “MÉDIAS”, e 3 imagens como “RUIM”. É possível observar

que a maior confusão do modelo se apresenta entre classes contíguas, onde é confundido 76 imagens “BOAS” como “MÉDIA” e 15 imagens “MÉDIA” como “RUIM”. Como um ponte forte, a modelagem produziu valores altos de *Recall* para todas as classes e um valor alto de precisão para a classe “BOA”, porém quando se leva em consideração o resultado de 100% de *Recall* para a classe “RUIM” é muito suspeito, já que apenas 3 imagens foram rotuladas como “RUIM” no conjunto de teste(1.76% do total). Os pontes fracos do modelo são referentes aos baixos valores de precisão para classe “MÉDIA” e “RUIM”, já que pode-se notar que o modelo previu diversas imagens incorretamente para essas classes.

6
Prova de Conceito

Considerando os resultados satisfatórios e o termo de compromisso entre PUC(Grupo LEARN) e Petrobras, neste capítulo apresenta-se o protótipo entregue à Petrobras, englobando tarefas básicas para uso por geofísicos e especialistas. Nesta versão, não foi incorporado ao software as técnicas de aumento de dados descritas na subseção 4.5, portanto, entrega-se um modelo com resultado de 88% de F1-score no conjunto de validação cruzada 10-folds e 77% em um conjunto de teste.

Dados os requisitos de entrega, foi estipulado que o programa deve ser capaz de realizar as seguintes **Funcionalidades** básicas: TREINAMENTO, PREDIÇÃO e REFINO. que serão melhor explicadas ao longo do capítulo.

Além disso, o produto foi projetado para garantir os seguintes **Indicadores de Qualidade**: (1) REPRODUTIBILIDADE dos resultados reportados, (2) CONFIABILIDADE da execução(código livre de bugs e fácil manutenção) e (3) CLAREZA de uso(documentação assertiva e enxuta). Com base nestes indicadores, constrói-se uma série de testes que buscam auferir e controlar a qualidade do programa.

Como último requisito do software, foi definido que todo o programa deve ser acessado por linha de código e passível de ser usado em terminal Linux. Apenas o módulo de refino, que necessita de um marcador de imagens, fará uso de uma interface gráfica.

6.1
Arquitetura e Tecnologias

O projeto está em sua maior parte desenvolvido na linguagem de programação Python, pois esta possui uma facilidade de integração com o TensorFlow, que é uma das principais bibliotecas de *Deep Learning*.

O sistema possui três funcionalidades principais:

– TREINAMENTO: permite que se realize o aprendizado da rede neural desde o início. Por meio desta funcionalidade, a Petrobras pode replicar o procedimento treino e salvar os pesos em um arquivo dentro da pasta *seismogramClassifiers/checkpoints*.

– PREDIÇÃO: recebe um diretório contendo os dados sísmicos e salva os resultados da predição em um arquivo ".txt" ou ".csv".
– REFINO: recebe um diretório contendo os dados sísmicos e um arquivo com as marcações de algumas imagens. Refina a rede com base nas imagens marcadas e realiza a funcionalidade de predição para todo o diretório. Esta funcionalidade é designada para obter bons resultados em aquisições muito distintas entre si. Convencionou-se o número de 100 imagens marcadas para a operação de refino.

Todas as funcionalidade do software são acessados de forma conveniente pela biblioteca "Invoke" do Python, que permite, por meio de simples linhas de comando, acessar as funcionalidades.

Além das três funcionalidade principais, inclui-se a funcionalidade de "métricas", que retorna o desempenho do classificador sobre um dado conjunto de dados, e a funcionalidade de teste, que é utilizada para controle da qualidade do software.

Para correta utilização do software, o usuário precisa primeiramente estar ciente para que serve cada uma das funcionalidades.

Visando este objetivo de CLAREZA(3), foi desenvolvido um arquivo "README", que contém as principais instruções do que faz cada funcionalidade e como operar cada uma delas.

É de suma importância que a Petrobras esteja a par da documentação e, em qualquer momento de dúvida, consulte sem hesitar o arquivo "README", que se encontra na raiz do projeto.

A arquitetura da rede proposta está apresentada descrita propriamente na subseção 4.3.

Nos tópicos a seguir detalha-se a instalação e execução do ambiente, as três funcionalidades principais da ferramenta e a auxiliar de "métricas", e o marcador de imagens. A funcionalidade de testes recebe uma seção especial.

6.1.1 Instalação e Execução

Todo o ambiente de utilização da ferramenta está pensado no uso de Docker. Por meio de um arquivo Dockerfile, um build e um executável é possível instalar e operacionalizar todo o ambiente. Todos estes arquivos estão presentes na raiz do projeto e suas funcionalidades estão descritas à seguir:

– Dockerfile: O arquivo Dockerfile contém as instruções para instalação do ambiente e dos pacotes. Este arquivo é responsável pela virtualização do sistema e a comunicação com o computador

– build.sh: O Arquivo de “BUILD” é responsável apenas montar a instrução de imagem presente no Dockerfile. O arquivo de “BUILD” é utilizado apenas uma vez.

– run.sh: O arquivo de “RUN” é utilizado para rodar o ambiente. Nele que é especificado para utilizar a GPU, as portas que serão utilizadas e a montagem do volume virtual. Toda vez que a máquina for reiniciada, ou o ambiente fechado, será preciso executar o “RUN” novamente.

Ambos os arquivo build.sh e run.sh são arquivos do tipo bash e podem ser executado pelo comando “sh bashfile.sh”.

Para detalhar mais o projeto, destaca-se alguns pontos importantes presentes no arquivo Dockerfile.

```
1  FROM tensorflow/tensorflow:1.14.0-gpu-py3-jupyter
2  COPY requirements.txt /requirements.txt
3
4  RUN apt-get update
5  RUN apt-get install vim -y
6  RUN ["apt-get", "install", "-y", "libsm6", "libxext6", "
       libxrender-dev"]
7  RUN pip install -r /requirements.txt
8  RUN mkdir /workspace
9
10 EXPOSE 8888
11 WORKDIR /workspace
12
13 ENTRYPOINT ["jupyter","notebook","--ip=0.0.0.0","--allow-root"]
```

Listing 6.1: Instruções do Dockerfile

Percebe-se na primeira linha que o ambiente virtual está configurado para utilizar o tensorflow 1.14, com o acesso a GPU, com o python3 e a ferramenta jupyter. Contudo, caso a GPU do usuário seja mais antiga e não esteja compatível com a versão 1.14 do tensorflow, o usuário deve-se atentar para alterá-la. Isto está corretamente indicado no README do arquivo.

Nas linhas a seguir, existe uma sequência de comandos RUN, instruindo a instalar os pacotes necessários do python e dependências do OpenCV. Estas dependências são importantes, pois permitem que o OpenCV, que é a responsável pela parte de Visão Computacional(leitura e processamento de imagens) opere corretamente.

Por volta das útlimas linhas está descrito a porta de comunicação com o container e o diretório de montagem do volume, que está em “/workspace”. Por meio desta configuração, é possível se comunicar de forma integrada com o ambiente virtual e salvar corretamente os arquivos na máquina local.

6.1.2 Funcionalidades Principais

Dentre as funcionalidade principais, a mais simples é a de predição, que apenas recebe o caminho para os dados de destino e retorna as classificações para os arquivos. Seu processo está diagramado na parte superior da figura 6.1, que apresenta o fluxograma para as três funcionalidades.

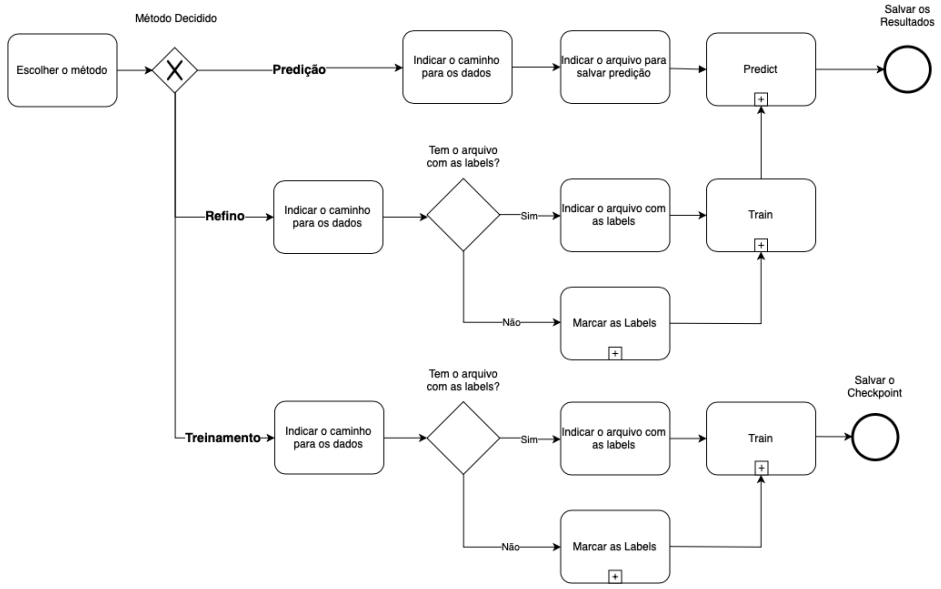

Figura 6.1: Funcionalidade Principais do Software(Treino, Predição e Refino).

Como parâmetro auxiliar para a predição, o usuário pode frequentemente querer escolher onde serão salvo o arquivo. Isso pode ser indicado adicionando o parâmetro -s, seguido pelo caminho do arquivo. A função a seguir apresenta uma instrução simples para predição.

```
1 inv classifier.predict -d "data-dir" -s "save-path"
```

Listing 6.2: Linha de comando da Funcionalidade de Predição

Para avaliar o resultado da predição, o usuário pode utilizar a função de "métricas", que está apresentada abaixo. Pode-se notar que os argumentos que a função recebe é um arquivo com as predições (-p), que é gerada pela função de predict, e o outro arquivo com as golden labels(-t), que indicam a real classe da imagens.

```
1 inv metrics.report -p "prediction-csv" -t "label-csv"}
```

Listing 6.3: Linha de comando da Funcionalidade de Métricas

Como resultado deste comando é apresentado ao usuário a matriz de confusão. Abaixo estão apresentado as métricas de precisão, recall e f1-score

por classe, seguido do resultado de acurácia, média dos f1 e a média dos f1 ponderada pela distribuição de cada classe.

```
root@tabitha:/workspace# inv metrics.report -p temp/dataset/FIGURAS_ML_PUC_2019_Co/results.csv -t temp/results/train_png.txt
Using TensorFlow backend.
[[3285   79    0]
 [  84 1584   77]
 [   0   13  101]]
              precision    recall  f1-score   support

         bad       0.95      0.91      0.93      1745
        good       0.98      0.98      0.98      3364
        ugly       0.57      0.89      0.69       114

    accuracy                           0.95      5223
   macro avg       0.83      0.92      0.86      5223
weighted avg       0.96      0.95      0.95      5223
```

Figura 6.2: Métricas expressas pela ferramenta, dentre elas a Matriz de Confusão, precisão, recall, acurácia e f1-score

A tarefa de refino é dentre elas a mais rebuscada. Tanto o treino quanto o refino necessitam como entrada o arquivo com a marcação manual das imagens. Este arquivo, caso não exista, pode ser gerado pelo marcador de imagens sísmica que está incluído junto ao produto e descrito na próxima seção.

O processo de refino recebe como entrada o caminho para os dados e para o arquivo de marcação. Com isso, realiza um treinamento sobre os dados, aprendendo por apenas uma época sobre a nova aquisição. Após isso, é feito a predição sobre diretório de dados indicado. A linha de código abaixo exemplifica uma execução do refino.

```
1 inv classifier.refine -d "data-dir" -l "label-path" -s "save-
    path"
```

Listing 6.4: Linha de comando da Funcionalidade de Refino

Na figura 6.1, as atividades de “Predict” e “Train” são referentes aos módulos do sistema. Com apenas esses dois módulos já é possível operacionalizar o sistema.

Por fim, a tarefa de treino recebe como entrada o caminho para os dados e o arquivo com a marcação manual das imagens. O código abaixo exemplifica uma execução do treino.

```
1 inv classifier.train -d "data_dir" -l "label_path"
```

Listing 6.5: Linha de comando da Funcionalidade de Treinamento

Ao executar o treinamento, a rede vai inicializar os pesos de forma aleatória e aprender sobre os mesmos por um intervalo de 40 épocas. Os resultados são salvos na pasta padrão dos checkpoints(*seismogramClassifiers/checkpoints*).

6.1.3 Marcador de Imagens Sísmicas

O Marcador de Imagens Sísmicas é uma ferramenta desenvolvida para permitir o usuário fazer a classificação manual das imagens entre “BOA”, “MÉDIA” e “RUIM”. Como saída do marcador é produzido um arquivo com a tupla da informação do nome do arquivo, e sua respectiva classe. Essa informação é passada à rede para o treinamento supervisionado.

Um diferencial do Marcador de imagens desenvolvido é que este é capaz de ler não apenas imagens em formato “.png”, “.jpg”, mas também pode ler e transformar os dados do arquivo no formato “.sgy”. O SEGY(“.sgy”) é um formato comum para armazenamento de dados sísmicos e segue um protocolo bem definido, que possibilita guardar os traços e os metadados geofísicos. O marcador é capaz de segmentar as linhas sísmicas e transformar os traços na imagem que a rede recebe.

A figura 6.3 ilustra o processo do marcador.

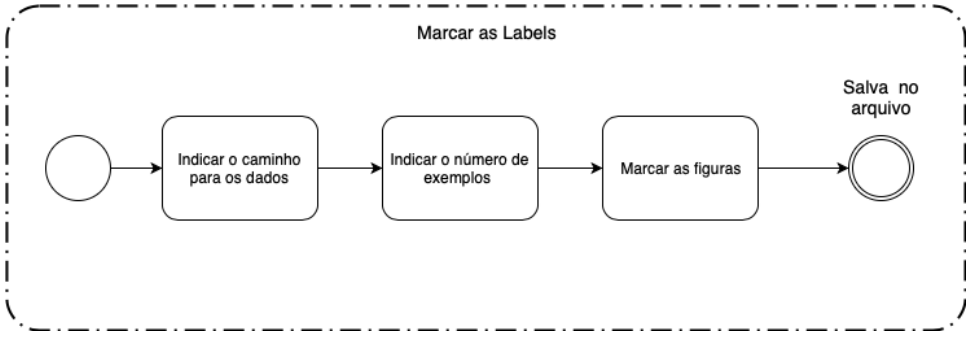

Figura 6.3: Processo do Marcador de Imagens.

O processo inicia com a indicação do caminho para os dados sísmicos e o número de exemplos que se deseja marcar, que é configurado por padrão em 100 imagens.

O número de 100 imagens é uma convenção, porém a ferramenta permite que o usuário marque quantas imagens desejar. A figura 6.4 exemplifica essa parte do processo. O usuário indica o caminho para os dados e seleciona o número de exemplo e depois, ao apertar o botão “OK”, as imagens são coletadas e apresentam-se os repesctivos nome e a quantidade total.

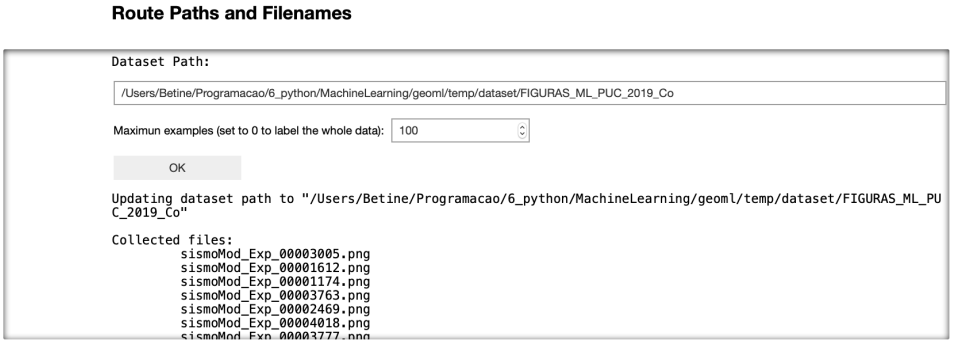

Figura 6.4: Escolha do caminho dos dados e número de exemplos a serem marcados.

Com esta etapa concluída, o usuário pode começar a marcação clicando sobre o botão "Start Labelling". Na figura 6.5 percebe-se que na parte inferior tem os botões com as classes. Ao clicar sobre eles, a ferramenta alterna consecutivamente para a próxima imagem e salva a escolha.

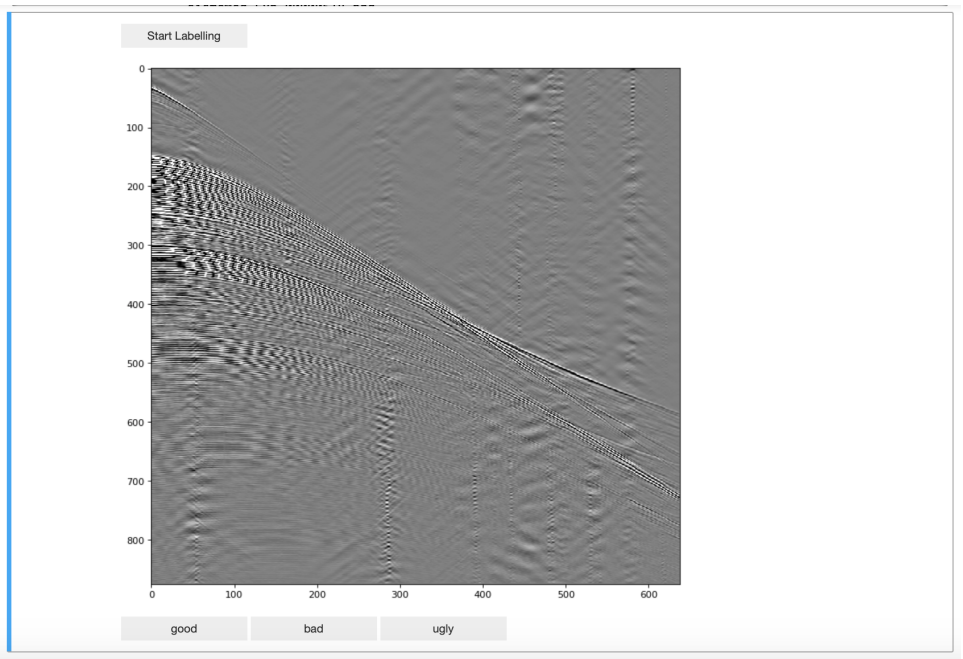

Figura 6.5: Exemplo de Classificação das imagens.

Quando terminarem as imagens especificadas, aparece um botão de salvar. Ao clicar, a marcação é salva para uma pasta padrão.

6.2 Testes

Toda a comunicação entre o código Python e as bibliotecas instaladas, já são bastante testadas e documentadas, portanto, não houve necessidade de realizar testes de integração. Consequentemente, os testes realizados foram

testes unitários, utilizando a biblioteca pytest do Python. O marcador de imagens sísmicas é alvo apenas de testes manuais, pois possui usabilidade simples e não é o objetivo principal do sistema.

Os testes sobre o sistema são divididos pela funcionalidade(Predição, Refino, Treino e Métricas) e o caso de teste.

O caso de Teste descreve qual a característica que busca ser avaliada e para cada caso existe um resultado esperado e um indicador atrelado.

O resultado esperado se refere ao valor de saída, que deve ser assegurado pela ferramenta. Esta saída pode ser uma booleana Verdadeira("OK"), uma booleana Falsa("FAIL") ou um valor numérico. Já o indicador mostra para qual diferencial competitivo o caso de teste contribui. Seu indicador pode contribuir com a REPRODUTIBILIDADE dos resultados reportados ou a CONFIABILIDADE dos comandos.

A Figura 6.6 apresenta todos os casos de testes que estão atualmente implementados.

Funcionalidade	Caso de Teste	Resultado Esperado	Indicador
Predição	Predição idêntica à anterior	OK	REPRODUTIBILIDADE
	Caminhos Incorretos	FAIL	CONFIABILIDADE
	Caminhos Inexistentes	FAIL	CONFIABILIDADE
	Não passar parâmetro Obrigatório	FAIL	CONFIABILIDADE
	Conferir se os resultados estão salvos	OK	CONFIABILIDADE
	Conferir se os resultados estão salvos na pasta indicada	OK	CONFIABILIDADE
	Refinar com os caminhos corretos sobre SEGY	OK	CONFIABILIDADE
	Refinar com os caminhos corretos sobre Image	OK	CONFIABILIDADE
Refino	Predição idêntica à refino anterior	OK	REPRODUTIBILIDADE
	Caminhos Incorretos	FAIL	CONFIABILIDADE
	Caminhos Inexistentes	FAIL	CONFIABILIDADE
	Não passar parâmetro Obrigatório	FAIL	CONFIABILIDADE
	Conferir se os resultados estão salvos	OK	CONFIABILIDADE
	Conferir se os checkpoints estão salvos	OK	CONFIABILIDADE
	Refinar com os caminhos corretos sobre SEGY	OK	CONFIABILIDADE
	Refinar com os caminhos corretos sobre Image	OK	CONFIABILIDADE
Treino	Predição idêntica à treinamento anterior	OK	REPRODUTIBILIDADE
	Caminhos Incorretos	FAIL	CONFIABILIDADE
	Caminhos Inexistentes	FAIL	CONFIABILIDADE
	Não passar parâmetro Obrigatório	FAIL	CONFIABILIDADE
	Conferir se os checkpoints estão salvos	OK	CONFIABILIDADE
	Treinar com os caminhos corretos sobre SEGY	OK	CONFIABILIDADE
	Treinar com os caminhos corretos sobre Image	OK	CONFIABILIDADE
Métricas	Dois arquivos idênticos	F1=1	CONFIABILIDADE
	Dois arquivos completamente distintos	F1=0	CONFIABILIDADE
	Arquivo de Treino Conhecido	F1=0.95	REPRODUTIBILIDADE
	Arquivo Incorreto	FAIL	CONFIABILIDADE

Figura 6.6: Testes realizados por caso e resultado esperado.

Percebe-se pela figura 6.6 que muitos casos de testes foram pensados e implementados, no total tem-se 27 casos de teste e 204 parametrizações, porém ressalva-se que é desafiador pensar em todos os casos de usos possíveis, sendo estes apenas os principais.

Nos itens a seguir explica-se brevemente o significado de cada caso de teste:

- predição idêntica: Executa-se a funcionalidade duas vezes com os mesmo parâmetros, o resultado deve ser o mesmo. Deve ser a mesma predição.
- caminhos incorretos: Passa um dos parâmetros de forma incorreta. Deve dar erro.
- caminhos inexistentes: Indica que vai passar um parâmetro, mas omite seu valor. Deve dar erro.
- parâmetro obrigatório: Omite-se um parâmetro mandatório. Deve dar erro.
- conferir se os resultados estão salvos: Executa-se a função e busca o resultado salvo na pasta. Deve estar presente o arquivo.
- conferir se os resultados estão salvos na pasta correta: Executa-se a função e busca o resultado salvo na pasta na pasta indicada. Deve estar presente o arquivo.
- funcionalidade funciona com SEGY: Executa-se a função com dados no formato segy e o parâmetro de imagem desativado. Deve conseguir operacionalizar perfeitamente.
- funcionalidade funciona com imagem: Executa-se a função com dados no formato de imagem e o parâmetro de imagem ativado. Deve conseguir operacionalizar perfeitamente.

Todos os testes são realizados e devem passar sem nenhuma falha, desta forma obtém-se 100% de cobertura sobre o código. Nota-se que a maioria dos indicadores garante a Confiabilidade do sistema e poucos são para garantir a reprodutibilidade. Isto se deve ao fato de que todos os resultados estão comparados com o melhor modelo entregue à Petrobras, que deve performar na primeira versão com de 88% de F1-score no conjunto de validação cruzada 10-folds e 77% no conjunto de teste.

7
Conclusão

As CNNs têm mostrado grande evolução na última década. Com resultados expressivos em tarefas de visão computacional, as CNN têm dominado o estado-da-arte em diversas aplicações. Em competições como a ILSRVC a SE ResNet já conseguiu superar com certa folga o resultado feito por um humano, obtendo 2.36% de erro top-5. No dataset do MNIST, as CNNs já reportaram o surpreendente resultado de 99.3% de acurácia, tornando a tarefa trivial.

Seguindo esta direção de avanço das CNNs, este trabalho mostra que é possível ter bons resultado em tarefas de classificação sísmica ao tratar as entradas como imagens. A aplicação das técnicas de deep learning provaram se úteis na classificação de ruído e consequentemente têm forte impacto na racionalização dos processos geofísicos, os quais podem ter mais agilidade e principalmente maior qualidade processos futuros de interpretação.

7.1
Proposta

Nesta dissertação foi proposta a rede Miniception para a tarefa de classificação de ruído em reunião de tiros. Outras arquiteturas de Rede Neural foram aplicadas, como por exemplo, a VGG 16, VGG 19 e Inception V3 e seus resultados comparados. Além disso, foram testadas diversas técnicas de aprendizado de máquina, como exemplo, o caminho residual (5), o bloco de atenção (6) e a Compressão e Excitação (*squeeze and excitation* (7)), os quais tiveram seus resultados expostos e comparados. Por fim, foi proposto um avaliação entre diferentes aquisições e apresentado um aumento de dados para a tarefa.

7.2
Contribuições

Com um F1 Score de 95,58 % em uma validação cruzada de 10 folds e 93,56 % em um conjunto de holdout de teste, este projeto demonstrou ter resultados satisfatórios para a tarefa de classificação de ruído em *common shot gather*. Todos os resultados superaram os baselines, que empregaram arquiteturas importantes, como a VGG 16, VGG 19 e Inception V3.

Uma das principais contribuições deste trabalho foi o aumento de dados. Por sua simplicidade matemática, esta operação pode ser replicada em novos trabalhos e tarefas que incluam dados pré-stack. Tendo em vista que a escassez de dados anotados é um grande limitante para desenvolvimento de modelos no setor de campos produtores, o *data augmentation* consegue amenizar essa deficiência, pois gera novos dados anotados com base em um dataset prévio. Tal contribuição torna esta transformação matemática importante para melhoria de resultados de trabalhos relacionados.

Além disto, uma materialização do esforço de projeto está na presente entrega do protótipo, onde observa-se que todo o projeto foi implementado dentro das especificações descritas, no tempo planejado e com casos de testes criados e testados. A Documentação elaborada contribui para dar clareza e entendimento ao sistema. Desta forma, garante-se um ambiente promissor e seguro para fazer a classificação de ruído.

7.3 Trabalhos Futuros

Como forma de melhorar ainda mais os resultados desta pesquisa, é sugerido que se utilize a metodologia e a arquitetura Miniception em maiores volumes de dados. Com mais informação disponível, é previsto que o resultado da validação cruzada seja ainda maior. Para o Holdout, sugere-se que sejam misturadas diferentes aquisições na etapa de treinamento. Com maior variabilidade de exemplos, a rede neural provavelmente irá aprender melhor como generalizar de uma aquisição para outra aquisição distinta.

O bloco convolucional utilizado é composto de convoluções 3 x 3 e 5 x 5. Incetiva-se o uso de kernels maiores que não foram utilizados nessa dissertação.

Também é sugerido utilizar o processo de aumento de dados em outras aquisições, já que é uma técnica computacionalmente barata e que melhora os resultados de classificadores. Não foi testado no âmbito deste trabalho, mas o procedimento de aumento de dados poderia ser ainda melhorado, pois o arredondamento para o inteiro mais próximo distorce o espectro de frequências dos dados. Sugere-se que os cálculos sejam feitos considerando os números reais oriundos do SEGy e depois aplicar uma interpolação nos valores fracionários para um grid regular.

No contexto deste trabalho foram selecionadas 100 imagens aleatórias como alvo da transformação de aumento de dados, contudo, sugere-se que em trabalhos futuros realizem o efetivo balanceamento das classes, aumentando apenas as classes “MÉDIA” e “RUIM”.

Outra sugestão, é a de utilizar o aprendizado não supervisionado para

atacar a tarefa. Com uma grande abundância de dados e uma metodologia que não precisa de marcação das imagens, espera-se que o resultado fique superior ao apresentado. Dentro do contexto de aprendizado não supervisionado, pode-se fazer uma abordagem considerando a similaridade entre as imagens e trocar a função de perda para a Triplet Loss.

Por fim, outra abordagem seria tratar o problema em uma classificação baseada nos traços. Pode-se utilizar redes recorrentes, que consideram a sequencias temporais, e aplicar uma classificação *many-to-one*, onde teriam múltipos traços como entrada e apenas um token de saída, que seria a resposta de classificação.

7 Referências bibliográficas

[1] DIAS, B.; BULCÃO, A. ; EVSUKOFF, A.. **Transfer learning with deep convolutional neural networks for seismic shot-gather quality classification**. p. 1–5, 05 2018.

[2] ALMEIDA, I.; VARGAS, M.; NOBRE, L.; SILVA, B.; BULCÃO, A.; LANDAU, L. ; EVSUKOFF, A.. **Machine learning applied in swell noise classification**. p. 1–5, 01 2019.

[3] SIMONYAN, K.; ZISSERMAN, A.. **Very deep convolutional networks for large-scale image recognition**. arXiv preprint arXiv:1409.1556, 2014.

[4] SZEGEDY, C.; VANHOUCKE, V.; IOFFE, S.; SHLENS, J. ; WOJNA, Z.. **Rethinking the inception architecture for computer vision**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 2818–2826, 2016.

[5] HE, K.; ZHANG, X.; REN, S. ; SUN, J.. **Deep residual learning for image recognition**. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), p. 770–778, 2016.

[6] WOO, S.; PARK, J.; LEE, J.-Y. ; KWEON, I.-S.. **Cbam: Convolutional block attention module**. In: ECCV, 2018.

[7] HU, J.; SHEN, L. ; SUN, G.. **Squeeze-and-excitation networks**. In: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2018.

[8] ELBOTH, T.; GEOTEAM, F. ; HERMANSEN, D.. **Attenuation of noise in marine seismic data**. In: SEG TECHNICAL PROGRAM EXPANDED ABSTRACTS 2009, p. 3312–3316. Society of Exploration Geophysicists, 2009.

[9] CHAKRABORTY, A.; OKAYA, D.. **Frequency-time decomposition of seismic data using wavelet-based methods**. GEOPHYSICS, 60(6):1906–1916, 1995.

[10] TAMUNOBERETON-ARI, I.; NGERI, A. ; AMAKIRI, A.. **Time-frequency attenuation of swell noise on seismic data from offshore central**

niger-delta, nigeria. IOSR Journal of Applied Geology and Geophysics (IOSR-JAGG), 3:30–35, 10 2015.

[11] ZHOU, H.-W.. **Practical Seismic Data Analysis**. Cambridge University Press, 2014.

[12] MEIER, M. A.; ROSS, Z. E.; HAUKSSON, E.; HEATON, T.; LI, Z.; RAMACHANDRAN, A.; BALAKRISHNA, A.; NAIR, S. ; KUNDZICZ, P.. **Harnessing the Power of Machine Learning Algorithms for Real-Time Signal/Noise Classification**. AGU Fall Meeting Abstracts, Dec. 2018.

[13] MEIER, M.-A.; E. ROSS, Z.; RAMACHANDRAN, A.; BALAKRISHNA, A.; NAIR, S.; KUNDZICZ, P.; LI, Z.; ANDREWS, J.; HAUKSSON, E. ; YUE, Y.. **Reliable real-time seismic signal/noise discrimination with machine learning**. Journal of Geophysical Research: Solid Earth, 12 2018.

[14] WISZNIOWSKI, J.; PLESIEWICZ, B. M. ; TROJANOWSKI, J.. **Application of real time recurrent neural network for detection of small natural earthquakes in poland**. Acta Geophysica, 62(3):469–485, Jun 2014.

[15] KUYUK, H. S.; YILDIRIM, E.; DOGAN, E. ; HORASAN, G.. **An unsupervised learning algorithm: application to the discrimination of seismic events and quarry blasts in the vicinity of istanbul**. Natural Hazards and Earth System Sciences, 11(1):93–100, 2011.

[16] MOUSAVI, S. M.; ZHU, W.; ELLSWORTH, W. ; BEROZA, G.. **Unsupervised clustering of seismic signals using deep convolutional autoencoders**. IEEE Geoscience and Remote Sensing Letters, p. 1–5, 2019.

[17] TITOS, M.; BUENO, A.; GARCÍA, L. ; BENÍTEZ, C.. **A deep neural networks approach to automatic recognition systems for volcano-seismic events**. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 11(5):1533–1544, May 2018.

[18] JAIN, P.; KULIS, B.; V. DAVIS, J. ; S. DHILLON, I.. **Metric and kernel learning using a linear transformation**. Journal of Machine Learning Research, 13, 10 2009.

[19] MCBREARTY, I.; DELOREY, A. ; A. JOHNSON, P.. **Pairwise association of seismic arrivals with convolutional neural networks**. Seismological Research Letters, 90, 01 2019.

[20] HOFFER, E.; AILON, N.. **Deep metric learning using triplet network**, 2014.

[21] VAN DER MAATEN, L.; HINTON, G.. **Visualizing data using t-SNE**. Journal of Machine Learning Research, 9:2579–2605, 2008.

[22] PAITZ, P.; GOKHBERG, A. ; FICHTNER, A.. **A neural network for noise correlation classification**. Geophysical Journal International, 212(2):1468–1474, 2017.

[23] WANG, Z.; WU, S.; LIU, C.; WU, S. ; XIAO, K.. **The regression of mnist dataset based on convolutional neural network**. In: Hassanien, A. E.; Azar, A. T.; Gaber, T.; Bhatnagar, R. ; F. Tolba, M., editors, THE INTERNATIONAL CONFERENCE ON ADVANCED MACHINE LEARNING TECHNOLOGIES AND APPLICATIONS (AMLTA2019), p. 59–68, Cham, 2020. Springer International Publishing.

[24] RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M.; BERG, A. C. ; FEI-FEI, L.. **ImageNet Large Scale Visual Recognition Challenge**. International Journal of Computer Vision (IJCV), 115(3):211–252, 2015.

[25] DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K. ; FEI-FEI, L.. **Imagenet: A large-scale hierarchical image database**. 2009.

[26] KRIZHEVSKY, A.; SUTSKEVER, I. ; HINTON, G. E.. **Imagenet classification with deep convolutional neural networks**. In: PROCEEDINGS OF THE 25TH INTERNATIONAL CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS - VOLUME 1, NIPS'12, p. 1097–1105, USA, 2012. Curran Associates Inc.

[27] SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S. E.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V. ; RABINOVICH, A.. **Going deeper with convolutions**. CoRR, abs/1409.4842, 2014.

[28] CHOLLET, F.; OTHERS. **Keras**. `https://keras.io`, 2015.

[29] CORTES, C.; VAPNIK, V.. **Support-vector networks**. Mach. Learn., 20(3):273–297, Sept. 1995.

[30] BOSER, B. E.; GUYON, I. M. ; VAPNIK, V. N.. **A training algorithm for optimal margin classifiers**. In: PROCEEDINGS OF THE FIFTH

ANNUAL WORKSHOP ON COMPUTATIONAL LEARNING THEORY, p. 144–152. ACM, 1992.

[31] CRAMMER, K.; SINGER, Y.. **Online ranking by projecting**. Neural Computation, 17(1):145–175, 2005.

[32] CRAMMER, K.; SINGER, Y.. **Pranking with ranking**. In: Dietterich, T. G.; Becker, S. ; Ghahramani, Z., editors, ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 14, p. 641–647. MIT Press, 2002.