Experimentos com Técnicas de SSL nos Campos de Segmentação Sísmica e HAR

Betania Eugenia Rodrigues da Silva	RA: 167508
Darlinne Hubert Palo Soto	RA: 264955
Fernando Gubitoso Marques	RA: 171524
Gabriel Borges Gutierrez	RA: 237300
Gustavo Pessoa Caixeta Pinto da Luz	RA: 271582
Vinicius Leme Soares	RA: 260727

19 de junho de 2024

1 Metodologia

Os experimentos e códigos estão em https://github.com/eborin/2024-MO436-grupo-6.

1.1 Datasets

Para o domínio de sísmica, utilizamos o dataset público Parihaka, disponibilizado públicamente pelo governo neo-zelandês, para o pré-treino dos modelos. Para a tarefa *downstream* de segmentação, utilizamos o dataset F3 fornecido.

No domínio de HAR, foi disponibilizado um arquivo chamado **har.zip** contendo os arquivos **train.csv**, **validation.csv**, and **test.csv** para a tarefa de classificação. As tarefas de pretexto foram treinadas com o dataset público RealWorld of Mannheim, na posição *waist*, compatível com as características dos dados fornecidos: dados obtidos de acelerômetro e giroscópio e um conjunto de atividades similar (*Climbingdown*, *climbingup*, *jumping*, *lying*, *running*, *sitting*, *standing*,*walking*).

1.2 *Backbones* para Sísmica

Utilizamos apenas um *backbone* para os experimentos de sísmica e variamos apenas a cabeça de projeção. Utilizamos a Resnet-50 sem suas camadas finais, tal como ocorre na rede Deeplabv3.

1.3 *Backbones* para HAR

A fim de analisar o impacto dos backbones em cada técnica que utiliza HAR no pré-treinamento e na tarefa de *downstream*, selecionamos dois *backbones*: um baseado em Convolutional Neural Networks (Conv) e outro baseado em Gated Recurrent Units (GRU).

1.3.1 Conv

O *backbone* baseado em *Convolutional Neural Networks*, é composto por uma série de camadas convolucionais seguidas por normalizações e funções de ativação. A primeira camada convolucional recebe como entrada 6 canais e aplica 32 filtros com *kernel* de 7, utilizando um *stride* de 2 e um *padding* de 3. Em seguida, uma *batch normalization* é aplicada, seguida por uma função de ativação ReLU.

Essa sequência de convolução, normalização e ativação é repetida em três blocos adicionais, cada um aumentando o número de filtros para 64, 128 e finalmente 256, enquanto mantém os tamanhos do *kernel* de 3 e *stride* de 1.

1.3.2 *GRU*

O *backbone* baseado em *Gated Recurrent Units* foi utilizado a partir do *backbone* da técnica CPC implementada pelos autores do TNC Tonekaboni et al. [2021], contendo uma arquitetura simples que permite avaliar diversas técnicas. Este modelo possui uma camada GRU bidirecional que processa sequências de entrada com 6 canais, com 100 unidades escondidas.

A saída da camada GRU é então passada por uma camada linear, que projeta a representação para um vetor de tamanho 256, adequado para as tarefas de *downstream*.

1.4 Contrastive Predictive Coding

Derivada do aprendizado contrastivo e proposta pela primeira vez na literatura em 2018 por Oord et al. [2018]. Originalmente desenvolvida para imagens, esse método tem demonstrado resultados promissores em séries temporais, especialmente no contexto de HAR. Neste projeto utilizaremos como referência principal o trabalho de Haresamudram et al. [2021], em que a técnica CPC é reapresentada e estudada no contexto da tarefa de HAR.

No âmbito da técnica CPC, a tarefa de pretexto consiste em prever as codificações de trechos futuros da série temporal a partir das representações latentes que já foram criadas para os dados passados.

O *backbone* da técnica é composto pela junção de duas redes, g_{enc} e g_{ar}. Foram utilizados cada um dos dois *backbones* descritos na subseção 1.3 como g_{enc}, mantendo a mesma rede autoregressiva g_{ar} proposta pelos autores de Haresamudram et al. [2021], que é uma GRU configurada com tamanho de entrada e *hidden* de 256, duas camadas, e dropout de 0,2. A rede prevê codificações futuras a partir das representações latentes geradas pelo g_{enc}.

A aplicação da técnica CPC requer uma pequena modificação em relação a outras técnicas de HAR: ao final de cada *backbone* descrito, é necessário manter três dimensões. Por isso, foi retirada a etapa de *flatten* que reduz as dimensões de 3 para 2, preservando a estrutura tridimensional necessária para o pré-treinamento da técnica CPC.

O pré-treinamento foi realizado com *batch_size* de 64, durante 500 épocas e paciência de 50, para ambos os casos. O tamanho das amostras é de 60 instantes de tempo, por 6 canais, com sobreposição de 50%. O melhor modelo escolhido é aquele que possui menor *loss* no conjunto de validação.

Após o pré-treinamento do *backbone*, avançamos para a tarefa *downstream*. Nesta fase, a cabeça de projeção (camada linear com dimensão 256) é descartada, preservando-se o *backbone* treinado. Em seguida, é utilizada a cabeça de predição descrita na seção **??** para realizar a tarefa alvo.

1.5 Barlow Twins

Originalmente, Zbontar et al. [2021] desenvolveram o Barlow Twins para produzir representações de dados de imagens. Depois Lee and Aune [2021] exploraram o potencial de diferentes técnicas SSL originalmente para imagens (entre elas Barlow Twins), mas aplicadas no contexto de dados de series temporais. É assim que eles adaptaram Barlow Twins usando um backbone simples, mantendo o valor λ original, e diminuindo a dimensionalidade da cabeça de projeção de 8192 a 4096. Baseado neste trabalho anterior, uma pequena exploração de hiperparâmetros foi feita, testando diferentes backbones, cabeças de projeção e factores λ, de acordo com a Tabela 1.

Tabela 1: Hiperparâmetros explorados na avaliação da técnica Barlow Twins.

Hiperparâmetros	Valores de Zbontar et al. [2021]	Valores de Lee and Aune [2021]	Valores explorados
Backbone	Resnet50	lightweight ResNet1D	Conv GRU
Dimensionalidade da cabeça de projeção	8192	4096	256 512 1024
Factor λ	0.005	0.005	0.001 0.005 0.01

A exploração de hiperparâmetros deve ser executada usando apenas os conjuntos de treino e validação. Devido à practicidade e facilidade, os modelos da exploração foram testados também no conjunto de teste meramente com fines ilustrativos, sem influenciar em toma de decisão nenhuma. Os resultados completos se encontram no repositório.

1.6 Learning from Randomness

Esta técnica recente de aprendizado auto-supervisionado não requer aumentações de dados. Ela consiste em treinar um *backbone* com n cabeças de projeção simultâneas cujo objetivo é predizer pseudo-anotações de um dado geradas a partir de n modelos com parâmetros aleatórios. Treinaram-se diferentes *backbones* para diferentes números de cabeças de projeção, 5, 8, 12 e 16, e escolheram-se aqueles que produziram melhor resultado na tarefa *downstream* para comparação com as outras técnicas de SSL.

1.6.1 Sísmica

No domínio de segmentação semântica, utilizou-se o backbone Resnet-50 presente no modelo DeepLabV3. Os projetores aleatórios consistem em três convoluções de *kernel size* 5 e passo 2 seguidas de um redimensionamento para (100×50) com interpolação bilinear. As cabeças de projeção por sua vez consistem em apenas uma convolução seguida de redimensionamento semelhante ao dos projetores aleatórios. Este último passo é feito apenas para evitar problemas com dados de dimensões levemente diferentes.

Os modelos foram pré-treinados por 50 épocas com *batch size* de 8. O pré-treino pode parar antecipadamente se a perda de validação (segundo as pseudo-*labels* dos projetores) não melhorar por três épocas. Em seguida foram treinados por 50 épocas no *dataset* F3. O modelo obtido ao final do processo de treinamento é aquele com melhor loss de validação (segundo as anotações do F3).

Para testar a eficiência dos *backbones*, utilizou-se uma cabeça de projeção formada por duas camadas de convolução transposta. Além disso, as redes *downstream* foram treinadas diversas vezes reduzindo a quantidade de dados de treinamento – para 50%, 10%, 5% e 1% – para averiguar a utilidade dos *backbones* para tarefas *few-shot*.

O desempenho dos modelos *downstream* foi medido utilizando a métrica mIoU.

1.6.2 HAR

No domínio de HAR, testaram-se dois backbones: um convolucional e outro contendo uma rede recorrente GRU. Os projetores aleatórios consistem em uma rede *multilayer perceptron* com 4 camadas, enquanto que as cabeças de projeção consistem em apenas uma camada linear.

Os modelos foram pré-treinados por 100 épocas com *batch size* de 256. O pré-treino pode parar antecipadamente se a perda de validação (segundo as pseudo-*labels* dos projetores) não melhorar por cinco épocas. O modelo obtido ao final do processo de treinamento é aquele com melhor loss de validação (segundo as anotações do dataset).

1.7 Temporal Neighborhood Coding

TNC (Temporal Neighborhood Coding) é um método de contraste que utiliza amostragem de contraste para comparar pares de amostras distantes. A versão original usa o teste estatístico Augmented Dickey-Fuller (ADF) para identificar a estacionariedade de um sinal, que foi mantida nesta implementação. A técnica foi apresentada por Tonekaboni et al. [2021] como um framework e testada originalmente com um codificador de Rede Neural Recorrente (RNN) bidirecional de camada única.

O treinamento do *backbone* na tarefa de pretexto consiste em estabelecer janelas como sendo vizinhas ou não-vizinhas em relação a cada uma das janelas de consulta da série temporal, em que o método de selecionar as janelas utilizado foi o teste estatístico. A cabeça de projeção é um classificador binário que recebe as amostras do espaço de codificação e prevê se elas pertencem à vizinhança temporal. A função de perda é calculada para janelas próximas e distantes, incluindo um termo de ponderação W, de forma que o modelo diferencie entre amostras temporalmente próximas e distantes. Após treinar o codificador de forma auto-supervisionada, os pesos do aprendizado de representação são armazenados e podem ser usados para a tarefa subsequente.

Foram utilizadas 100 épocas na tarefa de pretexto, com *batch size* de 32. A cabeça de projeção foi mantida como a implementação original, consistindo em uma camada *Fully Connected* seguida por uma ativação ReLU, uma camada de regularização e outra camada *Fully Connected*. Na tarefa *downstream*, foram utilizadas 1000 épocas máximas com critério de parada antecipada com paciência igual a 100.

1.8 SimCLR

O SimCLR é uma técnica de aprendizado auto-supervisionado baseada no aprendizado contrastivo. Essa abordagem envolve a criação de duas views de um mesmo exemplo por meio de transformações, e, em um espaço latente, tenta aproximar a representação de views geradas a partir da mesma imagem enquanto separa as views de imagens diferentes. As representações são construídas por meio de duas redes: a primeira rede (F) projeta os dados em um espaço intermediário, e a segunda rede (G), geralmente mais simples, faz a reprojeção em um segundo espaço, onde a aproximação das views da mesma imagem e o afastamento das views de imagens diferentes são realizados, conforme ilustrado na Figura 1.

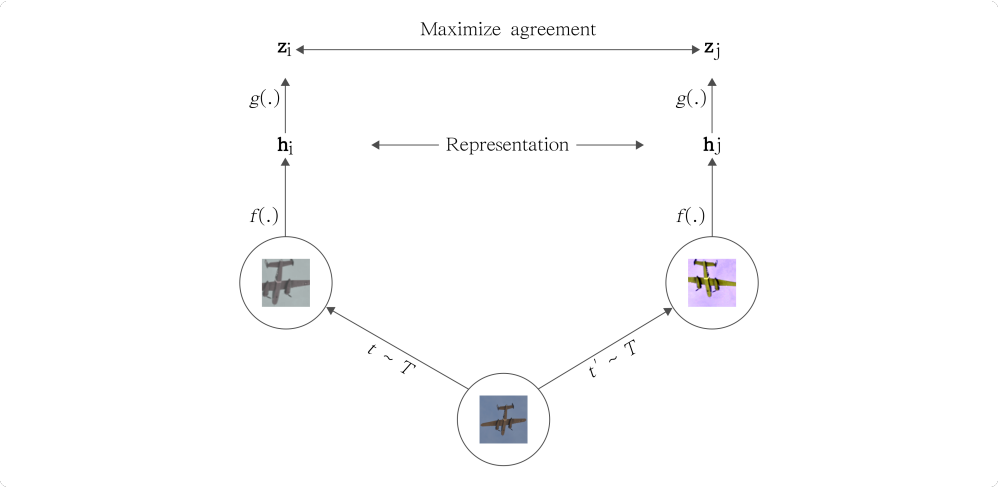


Figura 1: Arquitetura SimCLR

Para este experimento, utilizamos uma ResNet50 como a primeira rede (F) e uma MLP com 4 camadas FC e ativação ReLU. Após o treinamento com a tarefa de pretexto, apenas a ResNet50 (F) foi empregada como o backbone para a tarefa downstream. Realizamos três treinamentos distintos com o backbone, resultando em três backbones a serem avaliados: um treinado com batch size de 4, outro com batch size de 5 e o último com batch size de 10. Esses três backbones foram escolhidos para examinar o impacto do tamanho do batch na performance do modelo, uma vez que Chen et al. [2020] discutem a importância do tamanho do batch para o desempenho em tarefas downstream. A seleção do tamanho do batch foi baseada no hardware disponível: duas GPUs RXT 4090, com menor memória (23 GB) mas maior capacidade de operações por segundo (entre 6 e 7it/s), e uma GPU A6000, com maior memória (40 GB) porém menor velocidade (1.90 it/s).

Para avaliar o desempenho na tarefa downstream, foi empregada uma rede simples com apenas duas camadas de deconvolução como cabeça de predição. A função dessa camada é, de forma mais simples possível, transformar a representação construída pelo backbone no resultado desejado, que, neste caso, é a segmentação semântica de fácies sísmicas. Esse design foi escolhido para permitir uma avaliação clara da efetividade do pré-treinamento. Ao utilizar um segmentador simples, o peso do resultado da segmentação recai sobre o backbone, minimizando a influência da cabeça de predição.

As transformações empregadas também desempenham um papel crucial no SimCLR. Em geral, quanto maior o número de transformações, melhor o resultado do pré-treinamento. No entanto, no domínio da sísmica, é necessário ter cautela com as transformações, pois elas podem descaracterizar os dados, criando cenários que não ocorrem naturalmente. Considerando isso, as transformações aplicadas aos dados são detalhadas na Tabela 2.

Tabela 2: Transformações empregadas no SimCLR

Transformação	Parâmetros
Random Resized Crop	size = 256, scale = (0.2, 1) 100% de probabilidade
Random Horizontal Flip	50% de probabilidade
Random Color Jitter	b = 0.8, c = 0.8, s = 0.8, h = 0.2, 80% de probabilidade
Random Grayscale	20% de probabilidade
Random Gaussian Blur	50% de probabilidade

1.9 Bootstrap Your Own Latent

A técica Boostrap Your Own Latent consiste em treinar duas redes de mesma arquitetura onde uma delas (chamada de *online*) tenta aprender a representação da outra (chamada de *target*). Ambas são alimentadas com uma diferente visão de um mesmo dado, obtidas através de transformações aleatórias de um mesmo conjunto de transformações. Dessa forma, a rede *online*, por meio de uma cabeça de predição, procura aprender a projeção feita pela rede *target*. Os pesos da *target* são atualizados a partir de uma média móvel entre seu estado atual e atualização feita pelo cálculo da perda entre as representações, enquanto a *loss* é propagada normalmente pela rede *online*. Parte da implementação foi baseada no trabalho de Lightly Library [2024].

A implementação no contexto desse projeto se iniciou pela escolha do *backbone*. Foi usada a Resnet50 da mesma forma que no *backbone* da DeepLabV3. Suas últimas três camadas são descartadas, obtendo então uma representação da forma *BatchSize* x 2048 x (H / 32) x (W / 32), onde H e W são altura e largura originais do dado de entrada, respectivamente. Dessa forma, uma camada de *AveragePooling* é necessária na interface entre o *backbone* e a cabeça de predição da técnica (uma mlp com *Batch Normalization* e uma camada de ativação ReLu). A função de perda utilizada foi a Similiaridade entre Cossenos Negativa. Dessa forma, um valor próximo de -1 indica uma similaridade alta entre as representações, enquanto algo próximo de 0 indica pouca ou nenhuma similaridade e próximo de 1 uma similaridade alta, porém, com vetores em sentidos contrários.

As transformações originais descritas na implementação de Grill et al. [2020] e baseadas em Lightly Library [2024] consistem em: *Crop*, *Rotation*, *Horizontal* e *Vertical Flip*, *ColorJitter*, *GrayScale*, *GaussianBlur* e *Solarize*. Todas as transformações foram implementadas com probabilidades independentes de serem aplicadas. Os demais hiperparâmetros foram mantidos como no artigo original.

Uma cabeça de predição foi implementada pensando em uma equivalência com a avaliação linear, porém, para a tarefa de segmentação semântica. A cabeça de predição utilizada consiste em duas camadas de 2D *Transposed Convolution*, que levam o dado de um espaço de 2048 canais para 512 e de 512 para o número de classes de saída. Esse processo multiplica cada dimensão por 16 através do *kernel size*, portanto, uma última camada de interpolação no modo *bilinear* leva o dado ao tamanho original.

Com a implementação descrita, foi realizada uma busca pelos hiper parâmetros da técnica. Os principais testados foram combinações entre *Batch Size* e *Input Size* (limitados pela memória do sistema). Além dessas, algumas variações de *learning rate*, épocas de treinamento e passos de atualização também foram exploradas. No mesmo processo de busca por hiper parâmetros, as transformações sugeridas pelo artigo para o dado original foram selecionadas de acordo com a adequação ao dado e convergência do pré-treino.

Em um primeiro momento, as melhores configurações foram escolhidas a partir da *loss* reportada pelo pré-treinamento (quanto mais próxima de -1). Dessa forma, uma avaliação da representação obtida através do *backbone* foi realizada para definir qual das configurações seria utilizada para a tarefa *downstream*. A avaliação consistiu em congelar os pesos do *backbone* e, com 30 épocas de treino e 100% dos dados, treinar a cabeça de predição, reportando o IoU do conjunto de teste. Dessa forma é possível avaliar o espaço latente gerado pelo *backbone* e qual deles produz um conjunto de *features* mais relevante e com melhor relação com a saída.

Para uma métrica de comparação, a tarefa de *downstream* foi realizada tanto com os *backbones* pré-treinados como com os pesos inicializados aleatoriamente, junto da cabeça de predição da avaliação anterior. Com uma configuração de *Batch Size* de 8, *learning rate* de 0,007 e *CrossEntropyLoss* como função de perda, o modelo supervisionado convencional e pré-treinado foram treinados com as seguintes porcentagens do conjunto de treino: 1%, 5%, 10%, 25%, 50% e 100%. Dessa forma é possível analisar o ganho obtido através dos pesos inicializados da tarefa de pretexto. Para definição do melhor modelo, outras configurações de *Batch Size* e *learning rate* foram utilizadas na tarefa de *downstream* com o *backbone* pré-treinado.

2 Resultados

A avaliação da tarefa *downstream* para HAR consistiu em aplicar o protocolo *linear readout*, aplicando uma cabeça de predição em cima das representações congeladas aprendidas por cada *backbone*, medindo as métricas acurácia e F1-score. A cabeça de predição consistiu em três camadas lineares com ativações ReLU e uma de *Batch Normalization*. Além disso, foram realizadas avaliações do treinamento supervisionado e no regime de poucos dados. A Tabela 3 mostra os resultados

gerais da avaliação de cada técnica. Cada backbone foi treinado na sua correspondente tarefa de pretexto, foi inserido no classificador linear e treinado e testado com os dados da tarefa de pretexto.

Tabela 3: Resultados para HAR de acordo com técnica de SSL e Backbone no teste da tarefa downstream. Valores em negrito indicam a acurácia e F1-score máximos encontrados.

Técnica	Backbone	Acurácia	F1-Score
TNC	GRU	0,542	0,481
	Conv	0,542	0,469
BT	GRU	0,542	0,476
	Conv	0,583	0,517
CPC	GRU	0,412	0,383
	Conv	0,542	0,478
LFR	GRU	0,583	0,533
	Conv	**0,701**	**0,626**

A Tabela 3 mostra que a técnica LFR apresentou os melhores resultados para os dois *backbones*, atingindo métricas superiores ao máximo encontrado por outras técnicas. Isso se confirma também no regime de poucos dados apresentado na Tabela 4, em que a técnica superou o aprendizado supervisionado em até 37.4 pontos percentuais.

Tabela 4: Comparação de treinamentos entre as técnicas de HAR. Valores em negrito indicam a acurácia e F1-score máximos encontrados.

Forma de treinamento	**Backbone**	**Avaliação com n% de amostras**					
		n=1%		**n=50%**		**n=100%**	
		Acc	F1	Acc	F1	Acc	F1
Supervisionado	GRU	0,458	0,412	0,500	0,463	0,542	0,532
	GRU CPC	0,583	0,521	0,630	0,562	0,542	0,474
	Conv	0,453	0,552	0,458	0,405	0,500	0,435
	Conv CPC	0,458	0,369	0,625	0,563	0,417	0,337
SSL	GRU TNC	0,458	0,409	0,458	0,388	0,542	0,481
	Conv TNC	0,500	0,391	0,500	0,434	0,542	0,469
	GRU BT	0,417	0,393	0,458	0,429	0,542	0,476
	Conv BT	0,542	0,534	0,583	0,505	0,583	0,517
	GRU CPC	0,417	0,391	0,380	0,375	0,412	0,383
	Conv CPC	0,458	0,343	0,583	0,567	0,542	0,478
	GRU LFR	0,542	0,498	0,542	0,506	0,583	0,533
	Conv LFR	0,458	0,402	0,667	0,626	**0,701**	**0,626**

2.1 Análise qualitativa das representações geradas - HAR

Para realizar uma análise qualitativa dos modelos explorados, podemos tentar visualizar as representações geradas por deles. Para isso, usamos a técnica de visualização ***t*-SNE**. Com ela, projetamos as representações obtidas por cada modelo (Figura 2, Figura 3, Figura 6 e Figura 7) com o objetivo de reconhecer intuitivamente alguns padrões que podem não ser detectados por métodos quantitativos.

O **conjunto de teste** possui apenas 24 amostras, 4 por classe, e tentar observar padrões a partir de um gráfico com tão poucos pontos foi uma tarefa muito complicada. Por isso, também agregamos ao gráfico as visualizações das representações dos conjuntos de validação e treino. Para maior clareza, os pontos da mesma cor pertencem à mesma classe, enquanto aqueles que pertencem à mesma partição possuem a mesma forma. Um **círculo** representa o conjunto de treino, um **asterisco** representa o conjunto de validação, e um **losango** representa o conjunto de teste. Para evitar que os pontos do conjunto de teste passem despercebidos entre os pontos de treino e validação, eles possuem um contorno preto ao redor.

2.2 Contrastive Predictive Coding

Para realizar uma análise qualitativa, utilizamos o t-SNE Maaten and Hinton [2008] para visualizar os dados mapeados no espaço latente produzido pelos *backbones* pré-treinados com a técnica CPC.

Na figura à esquerda, em 2, que apresenta como codificador da série temporal bruta, uma convolução como g_{enc}, observamos uma melhor separação das classes em comparação com a GRU. Podemos identificar dois *clusters* bem definidos: atividades de baixa energia no canto superior esquerdo e atividades de média e alta energia no canto inferior direito. No *cluster* inferior, a atividade de alta energia *run* se agrupa mais abaixo, com pouca mistura com o *cluster* de média energia, que inclui atividades como *walk*, *stair-up* e *stair-down*, que não são claramente separáveis.

Por outro lado, na figura à direita, onde a série temporal é codificada por uma GRU, as amostras não são separadas, conforme esperado devido ao desempenho inferior da GRU em comparação à convolução, conforme evidenciado na tabela de resultados com treinamento com poucos dados 4.

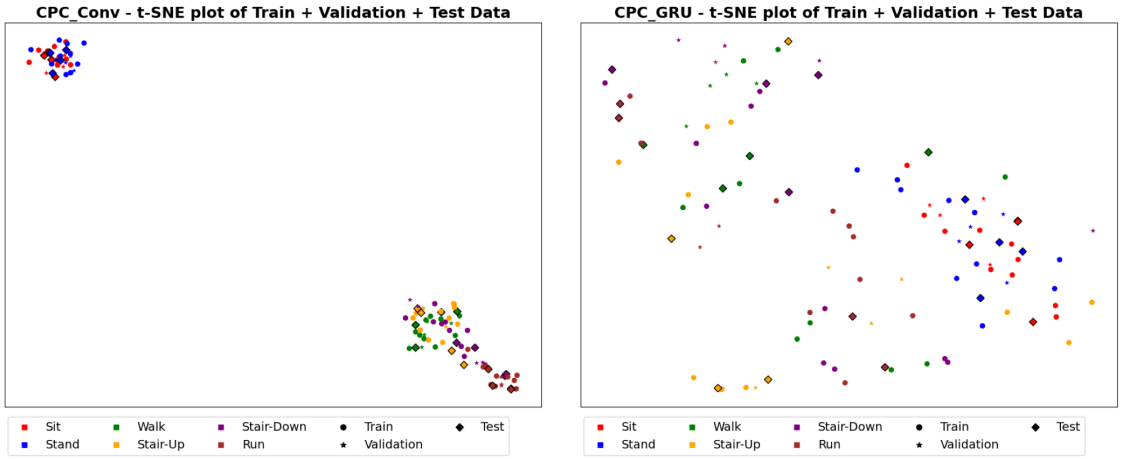


Figura 2: t-SNE plot pro dataset da tarefa *downstream* analisando a junção de treino, validação e teste, indicando o teste com bordas em negrito. Na figura a) g_{enc} é uma rede convolucional, e na figura b) g_{enc} é uma GRU

2.3 Barlow Twins

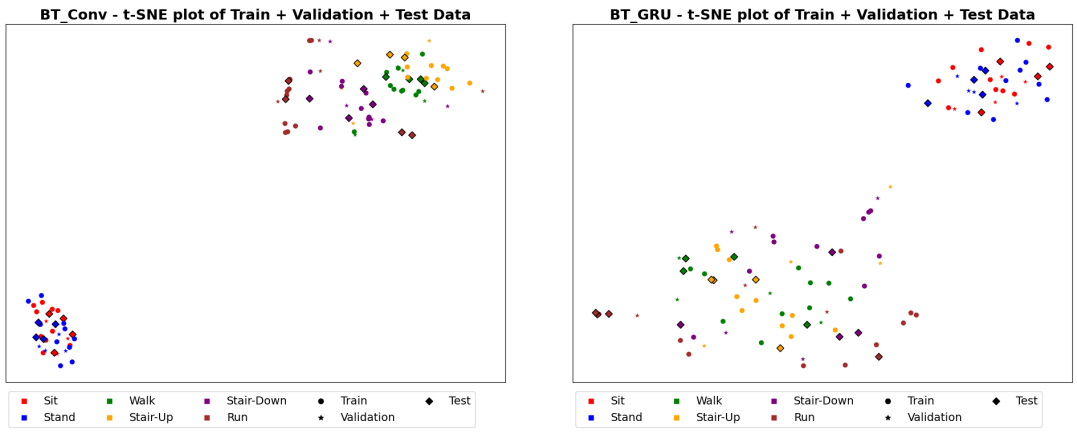
A busca de hiperparámetros realizada para Barlow Twins foi feita com os dados do conjunto de validação. Esta busca incluiu os backbones GRU e Conv, dimensionalidades de cabeça de projeção de 256, 512, e 1024, e valores de factor lambda λ de 0.001, 0.005, e 0.01. A Tabela 5 mostra os resultados desta busca. O máximo valor de **acurácia** obtido usando o backbone convolucional foi **0.583**, enquanto usar o backbone GRU não proporcionou resultados similares, atingindo uma acurácia máxima de **0.542**.

Tabela 5: Resultados da exploração de hiperparâmetros do modelo Barlow Twins. Valores em negrito indicam a acurácia e F1-score máximos encontrados.

Backbone	**Dimensionalidade da cabeça de projeção**	**Factor lambda λ**	**Métricas na validação** Acurácia	F1-score
Conv	256	0.001	0.500	0.424
		0.005	0.542	**0.516**
		0.01	0.500	0.495
	512	0.001	0.417	0.347
		0.005	0.458	0.412
		0.01	0.542	0.503
	1024	0.001	**0.583**	0.514
		0.005	0.500	0.462
		0.01	0.417	0.344
GRU	256	0.001	0.458	0.396
		0.005	**0.542**	**0.497**
		0.01	0.458	0.402
	512	0.001	0.500	0.446
		0.005	0.417	0.362
		0.01	0.333	0.280
	1024	0.001	0.500	0.420
		0.005	0.333	0.314
		0.01	0.333	0.305

Após a busca, as melhores configurações encontradas foram usadas para treinar dois modelos. Estas avaliações incluem a acurácia e F1-score usando o conjunto de teste, e os valores podem ser encontrados em Tabela 3.

Para avaliar qualitativamente o desempenho do modelo Barlow Twins, plots das projeções dos dados foram criados. Os pontos de todos os conjuntos (treino, validação, e teste) foram projetados a través do backbone convolucional (Figura 3a) e do backbone GRU (Figura 3b). A acurácia menor obtida pelo backbone GRU no conjunto de validação (Tabela 5) e teste (Tabela 3) se reflete nos pontos menos agrupados e mais espalhados e misturados da Figura 3b. Enquanto isso, em Figura 3a os pontos estão mais agrupados e é possível ver clusters mais definidamente. É notorio também como ambos modelos são capaces de separar fácilmente as atividades de baixo consumo de energía (sit e stand) das de alto consumo de energia (walk, run, stair-up e stair-down).



(a) Usando o backbone convolucional.

(b) Usando o backbone GRU.

Figura 3: *t*-SNE plots dos dados projetados usando modelos Barlow Twins com diferentes backbones. Para uma visão geral do espaço de projeção, foram plotados os pontos do conjunto de treino, validação, e teste. Em (a) foi usado um backbone convolucional, enquanto em (b) foi usado um backbone GRU. Os pontos pertencentes ao conjunto de teste são losangos com bordas em negrito.

2.4 Learning from Randomness

2.4.1 Sísmica

As métricas dos modelos de segmentação, exibidas na figura 4 mostram que a aplicação de uma tarefa de pretexto tende a produzir melhores resultados. O modelo com 12 cabeças obteve a melhor performance para o dataset completo, alcançando 71.47% de mIoU. Os modelos com 16 cabeças tende a ter melhor desempenho em *datasets* mais restritos.

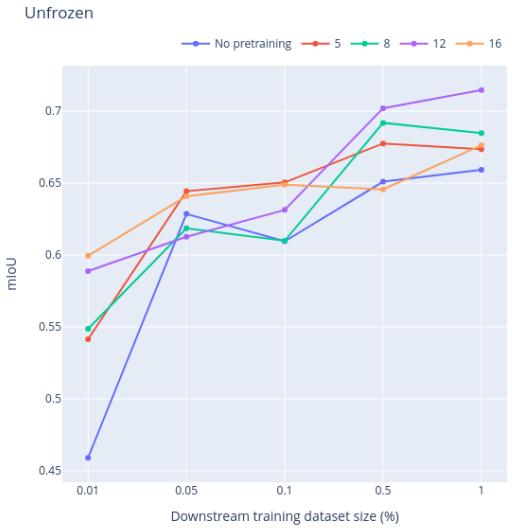


Figura 4: mIoU de modelos de segmentação. A legenda mostra o número de cabeças de projeção usadas durante o pré-treino. Note o eixo horizontal logarítmico.

2.4.2 HAR

Para a tarefa de classificação, nota-se que o *backbone* convolucional consistentemente obtém os melhores desempenhos, enquanto que o modelo com GRU não excede a acurácia de 60%, conforme mostram os gráficos da figura 5. Além disso, notou-se que o *backbone* convolucional tende a melhorar conforme se adicionam mais projetores aleatórios, com seu melhor modelo atingindo 70.1% de acurácia e 62.6% de F1-score.

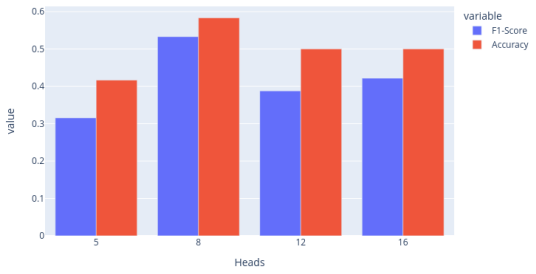


Figura 5: Métricas de modelos de classificação com GRU

Aplicando a técnica t-SNE para os melhores modelos que utilizaram cada uma das arquiteturas de *backbone*, produziram-se dois gráficos (figura 6) que demonstram a capacidade de classificação dos modelos. Nota-se que o gráfico t-SNE do modelo convolucional possui agrupamentos bem definidos de pontos – salvo as categorias *sit* e *stand* – e que pontos tanto dos conjuntos de treino e validação quanto do conjunto de teste ficam próximos segundo a sua classe. Por outro lado, o modelo com GRU possui pontos bem mais espalhados e misturados.

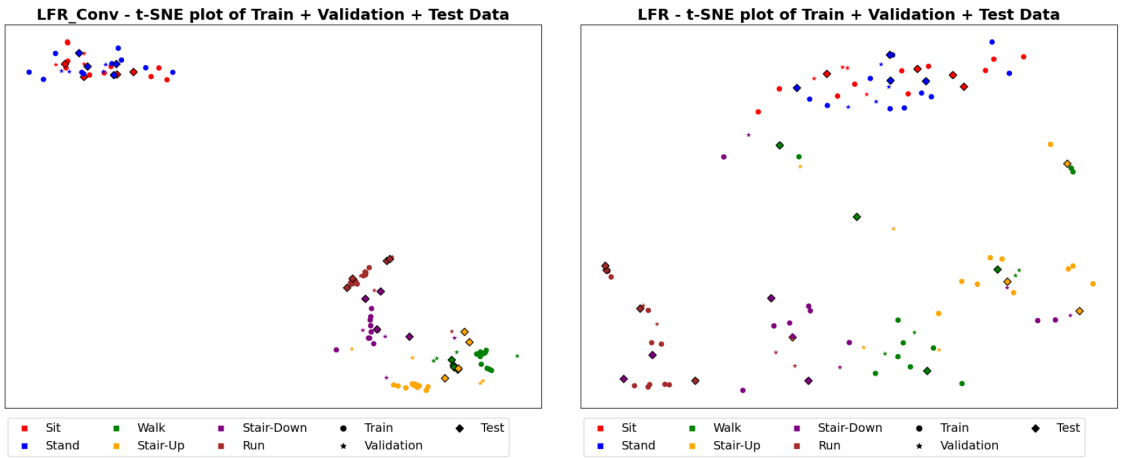


Figura 6: À esquerda, o melhor modelo com *backbone* convolucional. À direita, o melhor modelo com arquitetura GRU.

2.5 Temporal Neighborhood Coding

As tabelas 3 e 4 mostram que a aplicação da técnica TNC não alterou de forma significativa de acordo com o uso de diferentes *backbones*, atingindo o limite de 54,2% de acurácia. Uma hipótese pode se dever ao uso de *backbones* simples, já que a técnica possui um aumento de desempenho ao escolher de forma mais assertiva as arquiteturas utilizadas, como mostra o trabalho de Xu et al. [2023]. A Figura 7 mostra os gráficos *t*-SNE para a representação aprendida pelas variações dos *backbones* com a técnica TNC. É possível notar uma falta de melhor separação clara entre classes, independente do *backbone*. No entanto, nota-se em ambos os gráficos um agrupamento entre as classes *Sit* e *Stand*, as classes de baixa energia, das demais.

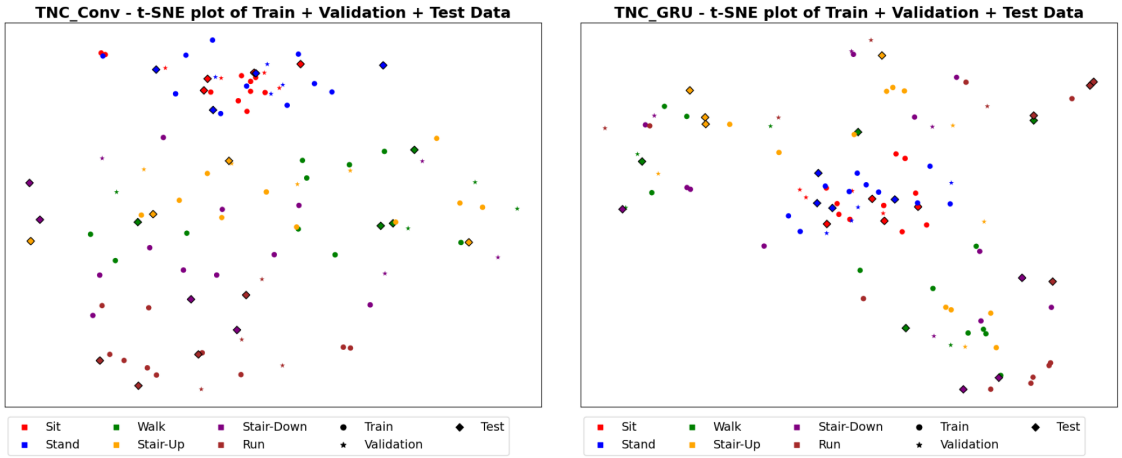


Figura 7: *t*-SNE plot pro dataset da tarefa *downstream* analisando a junção de treino, validação e teste, indicando o teste com bordas em negrito

2.6 SimCLR

Para o SimCLR, conduzimos dois conjuntos de experimentos: um avaliando o impacto do número de épocas de pré-treinamento e outro avaliando o impacto da quantidade de dados disponíveis para o *fine-tuning*.

Começando com a análise do impacto do número de épocas, Tabelas 6 e 7, observamos uma pequena melhora entre 50 e 100 épocas. No entanto, essa melhora é marginal e pode ser atribuída a diferenças nos pesos iniciais dos modelos. É interessante notar que as métricas de validação e treinamento estão muito próximas e significativamente mais altas que as de teste, possivelmente indicando um *overfitting*. Essa situação pode ser causada pelo fato de que as *inlines* de validação estão muito próximas às *inlines* de teste, comprometendo a escolha do modelo através das métricas de validação.

Tabela 6: Resultados experimentos SimCLR 50 Épocas

	Batch Size	mIoU	mF1
Test	4	0.6376	0.8691
	5	0.6358	0.8733
	10	0.6369	0.9281
Val	4	0.9541	0.9866
	5	0.9553	0.9879
	10	0.9549	0.9862
Train	4	0.9711	0.9886
	5	0.9714	0.9919
	10	0.9717	0.9918

Tabela 7: Resultados experimentos SimCLR 100 Épocas

	Batch Size	mIoU	mF1
Test	4	0.6442	0.9039
	5	0.6370	0.8983
	10	0.6301	0.9010
Val	4	0.9547	0.9849
	5	0,9545	0,9877
	10	0.9552	0.9884
Train	4	0.9717	0.9918
	5	0,9714	0,9917
	100	0.9710	0.9886

Olhando para os resultados do experimento com diferentes partes do dataset de fine-tuning, conforme mostrado na Tabela 8 e Figura 8, observamos um claro ganho de desempenho ao aumentar a quantidade de dados de 1% para 25%. No entanto, esse ganho de desempenho diminui em magnitude à medida que mais dados são acrescentados ao dataset.

Tabela 8: mIoU por porcentagem do dataset disponível para fine-tuning

Batch Size	01%	25%	50%	100%
4	0.5223	0.6270	0.6274	0.6442
5	0.5202	0.6340	0.6165	0.6370
100	0.5183	0.6104	0.6335	0.6301

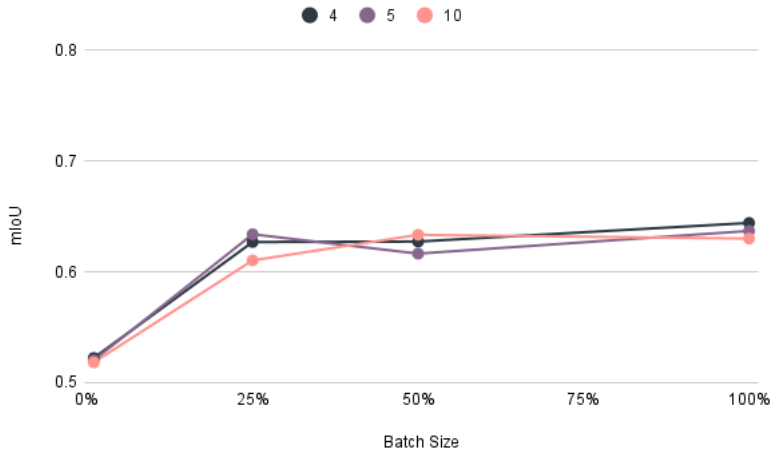


Figura 8: Batch Size Vs % do Dataset de Fine-tuning

É interessante notar também que, em ambos os experimentos, todas as quantidades de elementos por batch apresentaram desempenhos muito próximos. No artigo de Chen et al. [2020], discute-se a importância do batch size para a efetividade da técnica. Devido à falta de recursos computacionais, não foi possível atingir nem 1/10 do valor mínimo de batch size utilizado no artigo original (256 exemplos por batch). É altamente provável que o desempenho da técnica esteja sendo significativamente limitado pelo tamanho reduzido do batch.

2.7 Bootstrap Your Own Latent

A tabela 9 demonstra os *backbones* pré-treinados e suas repectivas *loss* na tarefa de pretexto. O *backbone* que melhor performou na avaliação com *freeze* foi o número 07, entretanto, o número 03 alcançou uma menor métrica na avaliação do pré-treino.

Tabela 9: Configurações de diferentes *backbones*

backbone	épocas	steps	batch	size	loss
1	300	10500	32	256	-0,914
2	1000	8000	128	128	-0,864
3	400	1400	32	256	**-0,927**
4	4000	8000	256	128	-0,834
5	4000	8000	512	64	-0,837
6	1000	8000	64	256	-0,847
7	650	10500	64	256	-0,879

A figura 9 mostra a comparação entre a métrica mIoU entre um modelo treinado no método supervisionado em comparação com o pré-treinado pela tarefa de pretexto. Pode-se observar o crescimento constante e capacidade de generalização maior do modelo pré-treinado. A figura 10 representa a comparação entre os melhores resultados encontrados treinando o mesmo modelo, nas mesmas configurações, com e sem pré-treino.

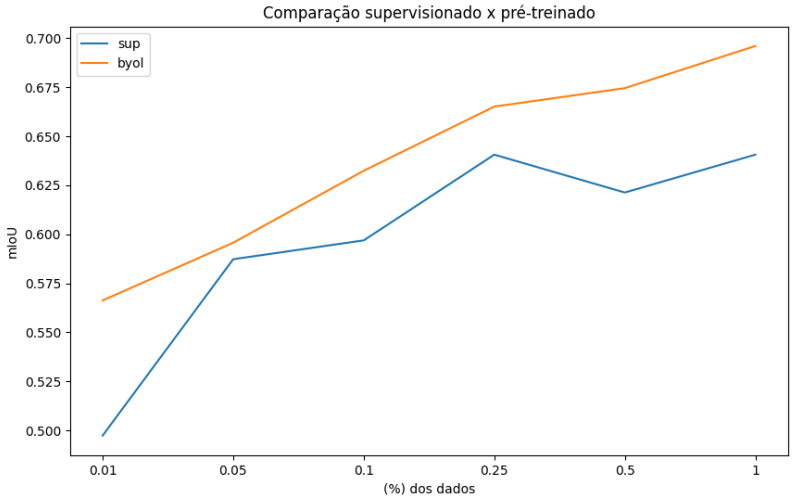


Figura 9: Regime poucos dados

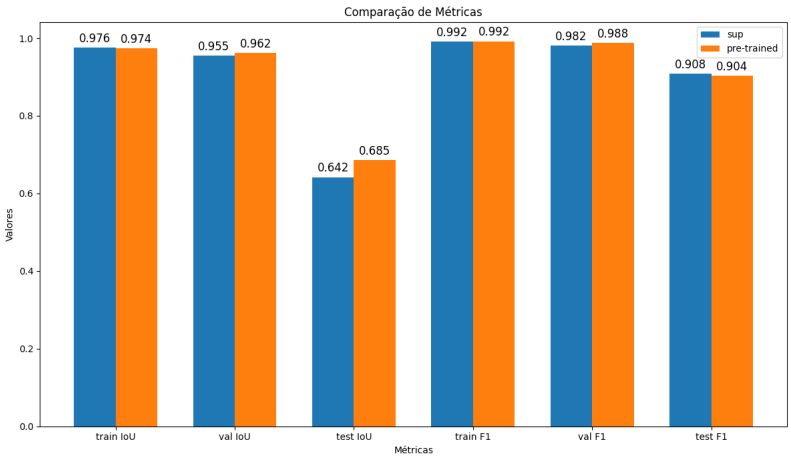


Figura 10: Comparação melhores modelos

3 Discussão

3.1 Contrastive Predictive Coding

Neste trabalho, foi explorado duas redes distintas para compor o *backbone* da técnica, ocupando o lugar de g_{enc}. Analisamos qualitativamente o impacto dos diferentes *backbones* utilizando *t*-

SNE e quantitativamente, através do regime de poucos dados com pré-treinamento e aprendizado supervisionado.

Observamos que o pré-treinamento com o *dataset* RealWorld não foi suficiente para capturar traços essenciais para a tarefa *downstream*, resultando em uma acurácia de 0,542 quando o g_{enc} é uma rede convolucional.

A comparação entre os diferentes g_{enc} sugere que a escolha do *backbone* é crucial para o desempenho. As redes convolucionais demonstraram uma melhor capacidade de separar as classes no espaço latente, em comparação com a GRU, o que destaca a importância da arquitetura do *backbone* na eficácia da técnica de pré-treinamento e na tarefa *downstream*.

3.2 Barlow Twins

A busca de hiperparámetros mostrou dois insights referentes à compatibilidade entre backbone e cabeça de projeção. Os dados em Tabela 5 sugerem que quando o backbone é convolucional, como os usados por Zbontar et al. [2021] e Lee and Aune [2021], o modelo se beneficia de uma cabeça de projeção de alta dimensionalidade. Já o oposto parece acontecer ao usar um backbone GRU, obtindo uma melhor acurácia enquanto menor seja a dimensionalidade da cabeça de projeção.

O factor λ, por outra parte, parece atuar em sinergia com o valor da dimensionalidade da cabeça de projeção. Assim, quando a cabeça de projeção não parece ser suficiente, um valor específico de λ pode fazer o modelo entregar uma melhor acurácia.

Trabalhos futuros poderiam explorar outros valores diferentes destes hiperparâmetros. Valores maiores para a dimensionalidade da cabeça de projeção no backbone convolucional e valores menores com o backbone GRU. Inclusive, identificar se um transformer simples como backbone requeriria também de uma dimensionalidade baixa de cabeça de projeção como o GRU. Algo que não foi explorado foi usar valores de λ maiores, ou multiplicar o factor $1-\lambda$ ao termo de invariancia e limitar a exploração apenas a valores entre 0 e 1 e identificar o tradeoff.

3.3 Learning from Randomness

Como esta técnica é muito flexível, e possui muitos hiperparâmetros que não puderam ser completamente explorados durante a execução deste estudo. O número de projetores aleatórios utilizado, o hiperparâmetro explorado aqui, é uma das mudanças mais simples que podemos fazer ao processo de aprendizado auto-supervisionado proposto pela técnica. O número de parâmetros e a complexidade de projetores aleatórios, por exemplo, podem ter forte influência sobre a qualidade do *backbone* gerado. De fato, como os projetores aleatórios utilizados nos experimentos descritos são mais complexos que as cabeças de projeção, é possível que parte do *backbone* não tenha sido ajustada de forma a produzir *features* úteis para a tarefa *downstream*, mas apenas para se adequar aos projetores aleatórios.

Quanto aos resultados relatados neste estudo, seria reducionista afirmar que uma maior quantidade de projetores aleatórios sempre leva a melhor desempenho. Embora isto se apresente nos resultados da maioria dos experimentos, fazemos a hipótese de que há um número ótimo de projetores para cada tarefa, e que aumentá-los levaria a queda de desempenho. Algo neste sentido ocorreu no experimento de HAR com *backbone* GRU. Nele, o melhor resultado surgiu do uso de 8 projetores. Mesmo ao dobrar este número, não obtivemos resultado semelhante.

3.4 Temporal Neighborhood Coding

Foi possível explorar novas variações de *backbones* para o TNC diferentes das tradicionalmente apresentadas na literatura, além de conjuntos de dados diferentes. Essa tarefa se mostrou mais desafiadora com poucos dados na tarefa de *downstream* permitindo uma avaliação diferente da implementada em outros trabalhos, no entanto não obtendo uma acurácia maior que outras técnicas como o LFR que conseguiu ter sucesso em separar os dados melhor ainda utilizando escolhas similares.

3.5 SimCLR

É possível perceber que o SimCLR não apresentou o desempenho esperado quando aplicado a dados sísmicos. Podemos especular sobre os motivos analisando os dois aspectos considerados importantes para a técnica: tamanho do batch e, de forma mais geral, os dados e transformações.

Como discutido anteriormente, os tamanhos de batch possíveis com o hardware disponível não chegam perto do mínimo discutido no artigo, o que prejudica significativamente o desempenho do

modelo. Outro ponto interessante a ser considerado é a grande falta de dados e a baixa diversidade dentro do dataset. Cada exemplo é muito parecido com outros em sua proximidade, resultando em uma diversidade limitada entre as múltiplas classes do dataset.

No artigo original, o dataset utilizado é o ImageNet, que contém 20 mil classes distintas, além de ter 1300 exemplos para cada classe no conjunto de treinamento. Em contrapartida, os datasets F3 e Seam Aí contêm aproximadamente 1500 e 1300 exemplos, respectivamente, no total, abrangendo apenas 6 classes, algumas das quais com pouquíssima área representada no dataset. Essa limitação pode dificultar a separação entre as classes durante a tarefa de pretexto.

Outro ponto a considerar são as transformações. Embora não sejam o foco principal no artigo original, outros estudos destacam claramente a importância e a quantidade de transformações aplicadas. Em Grill et al. [2020], observa-se que a performance do modelo cai consideravelmente quando certas transformações são removidas.

O problema é que a maioria das transformações aplicadas a imagens fotográficas convencionais não pode ser aplicada a imagens sísmicas ou simplesmente não altera a imagem de maneira significativa. Por exemplo, mudanças em saturação e hue, ou a aplicação de random grayscale, não afetam muito as imagens sísmicas, uma vez que elas já estão em preto e branco.

Essa combinação de fatores reduz a quantidade de transformações efetivas disponíveis, diminuindo assim a efetividade da técnica. Portanto, a limitação na variedade e na aplicação das transformações apropriadas para dados sísmicos contribui para o desempenho insatisfatório do SimCLR nesse contexto.

Esses três fatores combinados—baixo batch size, escassez de dados e poucas transformações—já representam um problema significativo para o desempenho da técnica. Quando combinados, esses fatores dificultam ainda mais a obtenção de bons resultados. Para uma melhor avaliação da técnica quando aplicada a dados sísmicos, é necessário que cada uma dessas deficiências seja endereçada individualmente. Somente após isso, uma análise mais minuciosa deve ser conduzida para determinar a real efetividade da técnica.

3.6 Bootstrap Your Own Latent

Ao longo do desenvolvimento do projeto, implementação e experimentação das diferentes combinações de hiper-parâmetros e configurações de testes, alguns pontos podem ser discutidos e algumas conclusões podem ser inferidas.

Com relação às transformações utilizadas por Grill et al. [2020], uma grande parte delas não se encaixa no contexto dos dados sísmicos. A técnica inicialmente foi desenvolvida para classificação de imagens (por isso entenda fotos/imagens com três canais RGB). Ao comparar a natureza dessas imagens com os dados símicos, vemos que alterações de cores não tem relação com um dado onde essas não tem nenhum valor semântico. O *Crop* realizado na imagem também tem uma influência considerável na convergência da representação. Como os dados sísmicos tem uma relação muito grande com a posição onde ele está inserido na imagem, principalmente na profundidade do dado, um *Crop* que retira a informação de contexto carrega o risco de descaracterizar a imagem. Efeitos de *Blur* também impedem que a cabeça de projeção retire informações relevantes da imagem. Dessa forma, baseado nessas análises e algumas configurações de pré-treino onde a *loss* não convergia para um valor consideravelmente baixo, as transformações utilizadas por fim foram *RandomCrop* com um *size* de no mínimo 64 e uma probabilidade de *Horizontal Flip*.

O artigo reporta um ganho significativo na performance da técnica ao utilizar de *Batches* maiores. Por conta da limitação de memória, processamento e de dados disponíveis para o pré-treinamento, as configurações de *Batch Size*, *Input Size* e épocas de treinamento foram limitadas. Foi possível observar o ganho de performance na avaliação com *freeze* no *backbone* ao utilizar *Input Sizes* maiores, mais *steps* de atualização e *batch* maior. Mais representativo do que a quantidade de épocas foi a quantidade de passos que o Gradiente Descendente realizou. Isso se da por conta do limite de dados no conjunto de pré-treino. Vale ressaltar também que quanto maior o tamanho do *Crop* da imagem, mais informação de contexto ela carrega, o que melhora a representação obtida ao final do treinamento. Conclui-se então que, quanto mais atualizações, maior tamanho da imagem e maior tamanho do *batch*, melhor a convergência do pré-treinamento e melhor performance na avaliação com os pesos do *backbone* congelados.

Pode-se observar também a baixa representatividade do conjunto de validação. Seu objetivo deveria ser de indicar o quão especializado nos dados de treinamento seu modelo se tornou e, a partir das métricas reportadas, definir qual época utilizar para avaliação no conjunto de teste. Entretanto, observa-se que os dados de validação foram retirados de uma maneira que são muito parecidos com os dados de treino. Isso se dá pela natureza do dado sísmico, onde duas imagens próximas serão quase idênticas. Por conta disso, por mais que não sejam usados para atualização

dos pesos do modelo, o conjunto de validação não desempenha um bom trabalho ao indicar um possível *overfit* do modelo. Pode-se observar isso pela distância considerável entre as métricas de validação e teste.

Por conta da baixa representatividade do conjunto de validação, os modelos treinados na tarefa de *downstream* provavelmente estão sofrendo de *overfit*, porém, pode-se observar que o pré-treino com a tarefa de pretexto permite um modelo com capacidade maior de generalização independente da porcentagem de dados utilizados quando comparado o aprendizado supervisionado com o modelo *from scratch*.

Referências

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In International conference on machine learning, pages 1597–1607. PMLR, 2020.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. Advances in neural information processing systems, 33:21271–21284, 2020.

Harish Haresamudram, Irfan Essa, and Thomas Plötz. Contrastive predictive coding for human activity recognition. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 5(2):1–26, 2021.

Daesoo Lee and Erlend Aune. Vibcreg: Variance-invariance-better-covariance regularization for self-supervised learning on time series. CoRR, abs/2109.00783, 2021. URL https://arxiv.org/abs/2109.00783.

Lightly Library. Self-supervised learning examples: Byol, 2024. URL https://docs.lightly.ai/self-supervised-learning/examples/byol.html. Accessed: 2024-06-18.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. Journal of Machine Learning Research, 9(Nov):2579–2605, 2008.

University of Mannheim. DataSet - RealWorld | Universität Mannheim. URL https://www.uni-mannheim.de/dws/research/projects/activity-recognition/dataset/dataset-realworld/.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. arXiv preprint arXiv:1807.03748, 2018.

Sana Tonekaboni, Danny Eytan, and Anna Goldenberg. Unsupervised representation learning for time series with temporal neighborhood coding. arXiv preprint arXiv:2106.00750, 2021.

Maxwell Xu, Alexander Moreno, Hui Wei, Benjamin Marlin, and James Matthew Rehg. Retrieval-based reconstruction for time-series contrastive learning. In The Twelfth International Conference on Learning Representations, 2023.

Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. CoRR, abs/2103.03230, 2021.