Trabalho 3 - MC886

Henrique Parede de Souza - 260497
Mateus de Lima Almeida - 242827

Introdução

O problema escolhido para este trabalho foi a classificação para o dataset **F3 Block Seismic Dataset**. Este dataset possui diversas imagens contendo características geológicas do solo, cada uma contendo um .png de anotação identificando as 6 classes possíveis.

Apesar do F3 ser um dataset voltado para segmentação, podemos utilizá-lo para classificação. Para isso, atribuímos um label para cada imagem a partir da imagem de anotação correspondente, sendo o label um vetor de 6 posições que indica se cada uma das 6 classes está presente na anotação, sendo, portanto, um problema de classificação multiclasse. Consideramos uma classe presente na anotação se ela tiver ocorrência maior que 10%.

O código da implementação encontra-se ao final do Jupyter Notebook anexado à tarefa. Para acessar os modelos implementados, acesse o link: models_assignment_3.

Pré-processamento e *gradient handling*

O pré-processamento aplicado nesta pipeline de classificação *multilabel* consiste em preparar as imagens e os rótulos de forma adequada para o treinamento da rede. Primeiramente, as imagens no formato TIFF são carregadas e normalizadas para o intervalo de 0 a 255, sendo convertidas para o tipo uint8. Caso a imagem seja em tons de cinza (2D), ela é convertida para uma imagem RGB com 3 canais, duplicando o canal original.

No caso dos modelos customizados, as imagens no formato TIFF são convertidas para PNG, isso foi realizado para reduzir a RAM utilizada durante o treinamento dos modelos, que era muito maior (excedendo o permitido pelo Google Colab) quando feito diretamente com as imagens TIFF.

Em seguida, é aplicada uma sequência de transformações nas imagens: redimensionamento para 224x224 pixels (tamanho esperado por modelos como a ResNet), conversão para tensor e normalização com média e desvio padrão padrão do ImageNet. Essas transformações garantem que as imagens estejam em um formato compatível com o modelo pré-treinado utilizado.

Por fim, as máscaras correspondentes às imagens são analisadas para identificar quais classes estão presentes. Para cada imagem, é criado um vetor binário indicando a presença (ou não) de cada uma das seis classes, com base na proporção de pixels pertencentes a cada classe na máscara. Esses vetores servem como rótulos multilabel para o treinamento e são armazenados em um arquivo CSV.

O tratamento dos gradientes no treinamento é feito para garantir estabilidade e evitar acúmulos indesejados. Antes de cada backpropagation, os gradientes são zerados com *optimizer.zero_grad()*, evitando que resíduos de iterações anteriores afetem as atualizações. Em seguida, a função de perda propaga os gradientes com *loss.backward()*, e o otimizador Adam atualiza os pesos via *optimizer.step()*. Esse otimizador é adequado para problemas com múltiplas classes e inclui regularização L2 (*weight_decay*) para reduzir *overfitting*.

Modelo Base (ValDir)

O modelo base foi feito a partir da ResNet18, disponibilizada no Torch, com inicialização aleatória. A camada de saída foi adaptada para ter 6 neurônios, correspondendo ao problema de classificação proposto.

Cada imagem foi convertida de .tiff para uma variável do Pillow, por questões de praticidade. Além disso, as imagens em níveis de cinza foram convertidas para RGB, por meio da cópia do canal original. Por fim, cada uma delas foi redimensionada para 224x224, convertida em tensor e normalizadas com a média e desvio padrão do ImageNet.

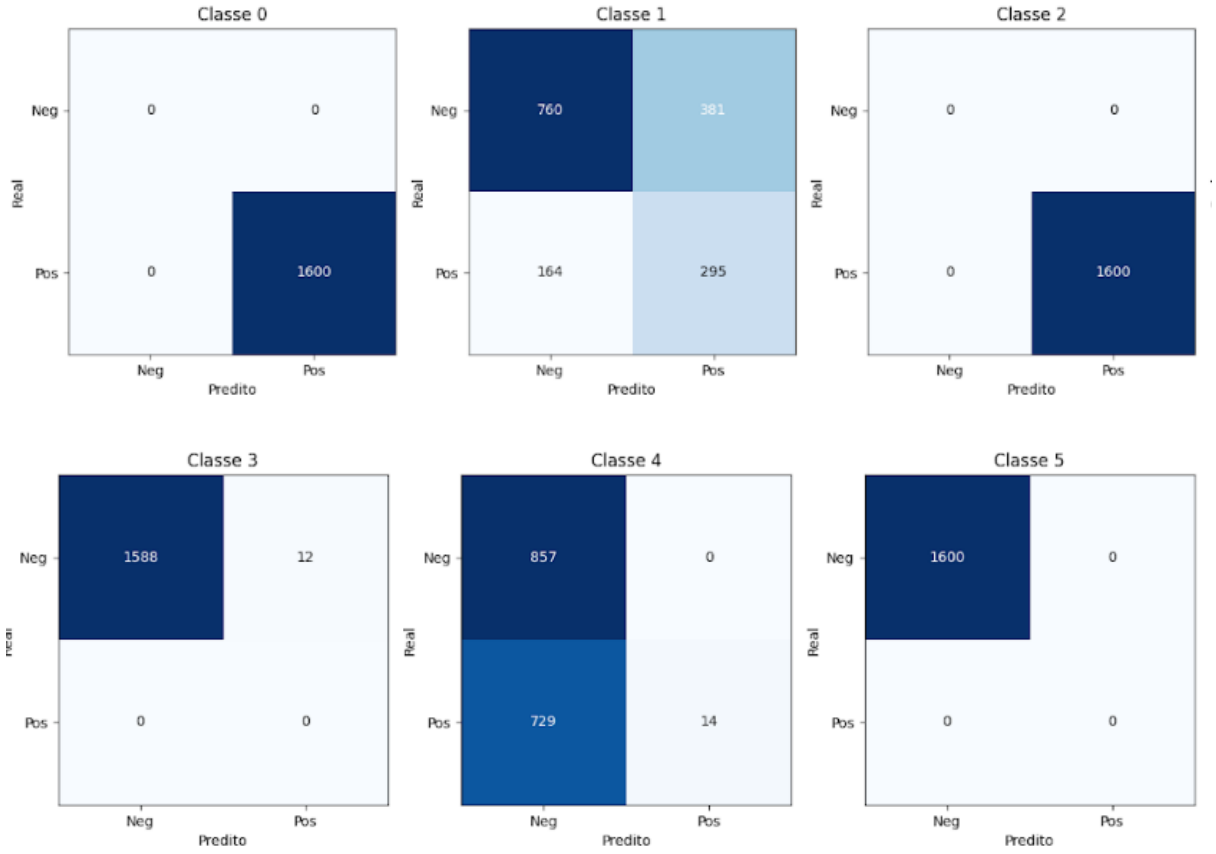
Para cada epoche do treinamento, foi realizado forward pass, backward pass e a atualização dos pesos. Além disso, foi utilizada a loss function BCEWithLogitsLoss, o otimizador Adam (learning rate de 0.001) e o early stop com paciência de 5 epochs.

Durante a validação, as saídas da rede passam por uma sigmoide para obter probabilidades, sendo utilizado o F1Score macro. Por fim, para a fase de teste, utiliza-se um limiar dinâmico.

Os resultados podem ser encontrados no classification report e nas confusion matrix a seguir.

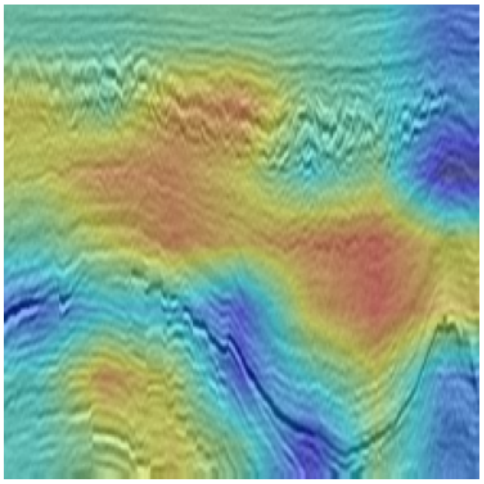
Classification Report:

	precision	recall	f1-score	support
Class 0	1.00	1.00	1.00	1600
Class 1	0.44	0.64	0.52	459
Class 2	1.00	1.00	1.00	1600
Class 3	0.00	0.00	0.00	0
Class 4	1.00	0.02	0.04	743
Class 5	0.00	0.00	0.00	0
micro avg	0.90	0.80	0.85	4402
macro avg	0.57	0.44	0.43	4402
weighted avg	0.94	0.80	0.79	4402
samples avg	0.92	0.81	0.84	4402



Analisando os resultados do modelo, percebemos um desempenho médio satisfatório, apesar de haver um desbalanceamento entre as métricas de cada classe. Uma sugestão de implementação futura seria a utilização de uma ResNet50, mais formas de data augmentation e inicialização com pesos da ImageNet.

Por fim, podemos observar o GradCAM aplicado na imagem *il_0.png* para este modelo. Nela, vemos um destaque de algumas regiões separadas pelo modelo.



Modelos Customizados

Foram desenvolvidos e avaliados três modelos convolucionais distintos para a tarefa de classificação. Cada modelo foi projetado com diferentes níveis de profundidade e complexidade, visando analisar o impacto de alterações estruturais no desempenho final da rede.

Modelo 1 (Lindinha) – Arquitetura Simples e Eficiente

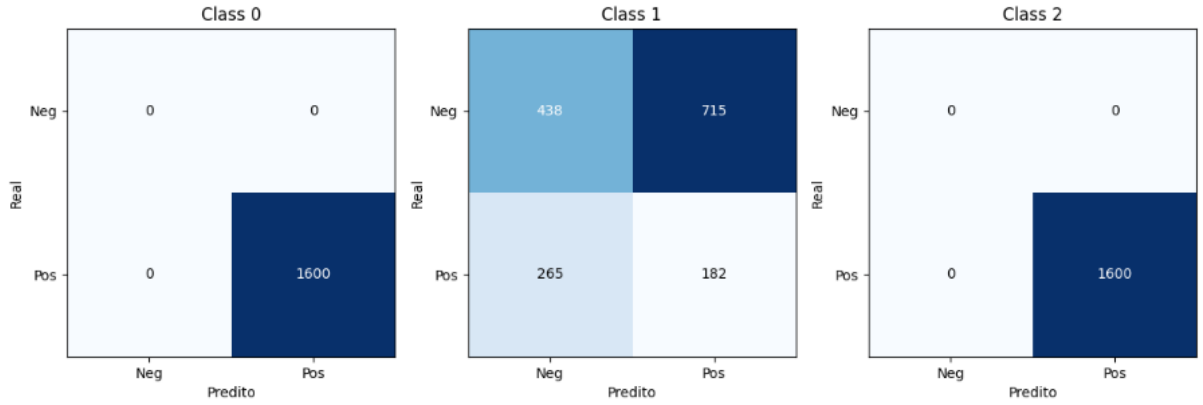
O primeiro modelo, batizado de Lindinha, consiste em uma arquitetura rasa com duas camadas convolucionais (16, 32), utilizando filtros de tamanho 5x5, sem uso de *dropout* nem *batch normalization*. Trata-se de uma rede leve, com poucos parâmetros e baixa profundidade, projetada para capturar padrões visuais mais amplos, o que pode ser benéfico em contextos com pouca variação de textura ou quando o ruído local não é preponderante.

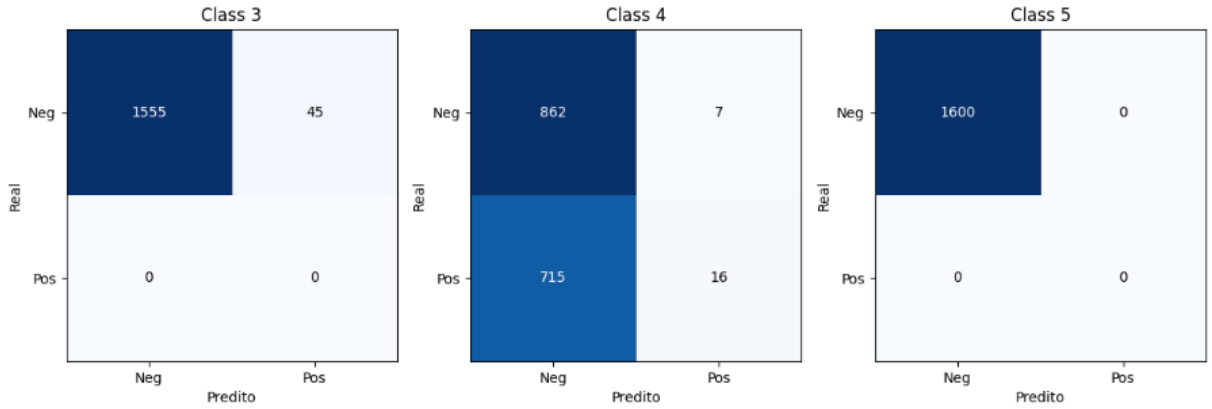
- **Vantagens:** Baixo custo computacional, rápida inferência, desempenho competitivo em termos de acurácia geral.

- **Desvantagens:** Menor capacidade de generalização em cenários complexos.

Classification Report:

	precision	recall	f1-score	support
Class 0	1.00	1.00	1.00	1600
Class 1	0.20	0.41	0.27	447
Class 2	1.00	1.00	1.00	1600
Class 3	0.00	0.00	0.00	0
Class 4	0.70	0.02	0.04	731
Class 5	0.00	0.00	0.00	0
micro avg	0.82	0.78	0.80	4378
macro avg	0.48	0.40	0.39	4378
weighted avg	0.87	0.78	0.77	4378
samples avg	0.84	0.80	0.80	4378





Modelo 2 (Florzinha) – Arquitetura Intermediária com Regularização

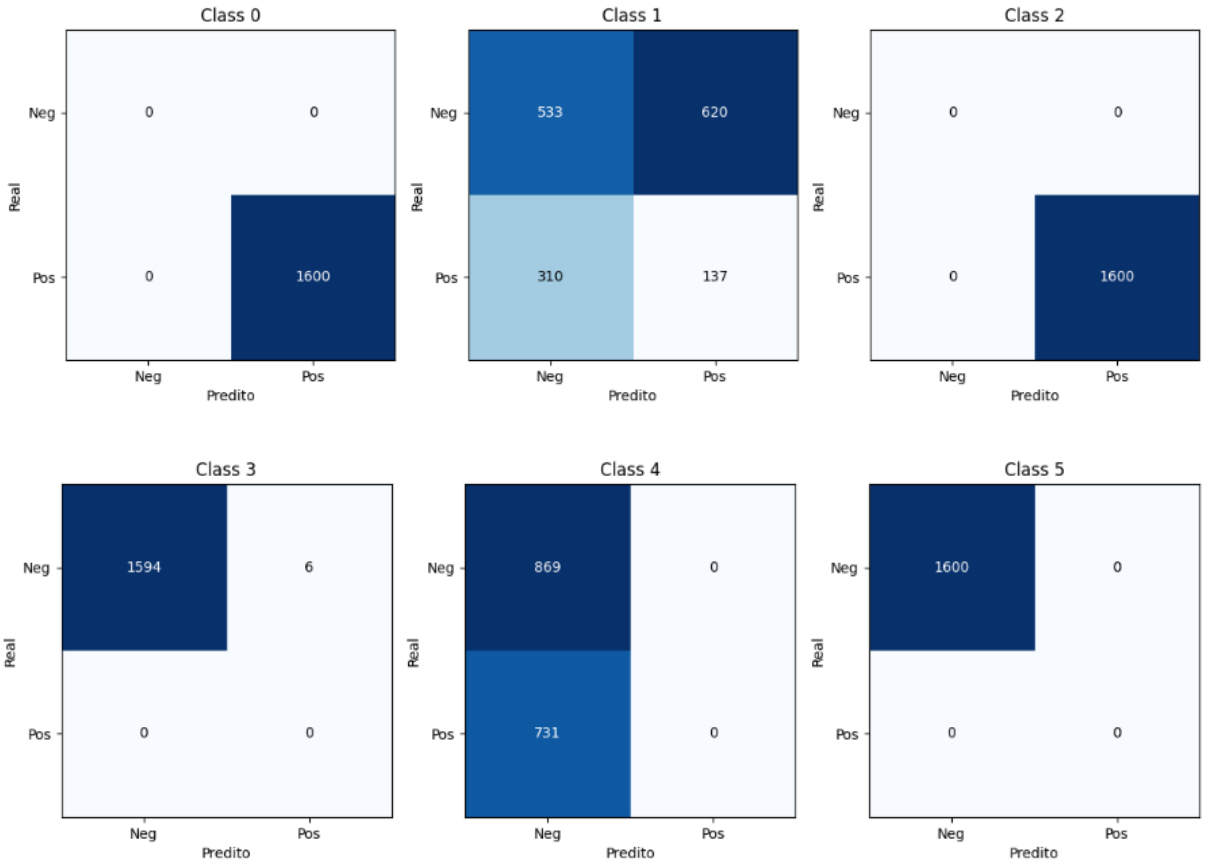
O segundo modelo, batizado de Florzinha, amplia a capacidade da rede ao incorporar três camadas convolucionais (32, 64, 128) com filtros 3x3, juntamente com camadas de *batch normalization* e *dropout* (taxa 0.4). Essa configuração busca um equilíbrio entre capacidade de aprendizado e regularização, reduzindo o risco de *overfitting*.

- **Vantagens:** Estrutura mais robusta, maior profundidade permite extração de características hierárquicas mais refinadas, mitigando ruídos locais.

- **Desvantagens:** Maior custo computacional em relação ao modelo anterior, sem ganhos expressivos em classes minoritárias.

Classification Report:

	precision	recall	f1-score	support
Class 0	1.00	1.00	1.00	1600
Class 1	0.18	0.31	0.23	447
Class 2	1.00	1.00	1.00	1600
Class 3	0.00	0.00	0.00	0
Class 4	0.00	0.00	0.00	731
Class 5	0.00	0.00	0.00	0
micro avg	0.84	0.76	0.80	4378
macro avg	0.36	0.38	0.37	4378
weighted avg	0.75	0.76	0.75	4378
samples avg	0.87	0.78	0.81	4378



Modelo 3 (Docinho) – Arquitetura Profunda e Potente

O terceiro modelo, batizado de Docinho, adota uma arquitetura mais profunda, composta por quatro camadas convolucionais com número crescente de canais (`64, 128, 256, 512`), filtros 3×3, *dropout* (taxa 0.2) e *batch normalization*. Essa configração proporciona alta capacidade de aprendizado, permitindo que a rede capture padrões mais complexos e sutilezas presentes nas imagens.

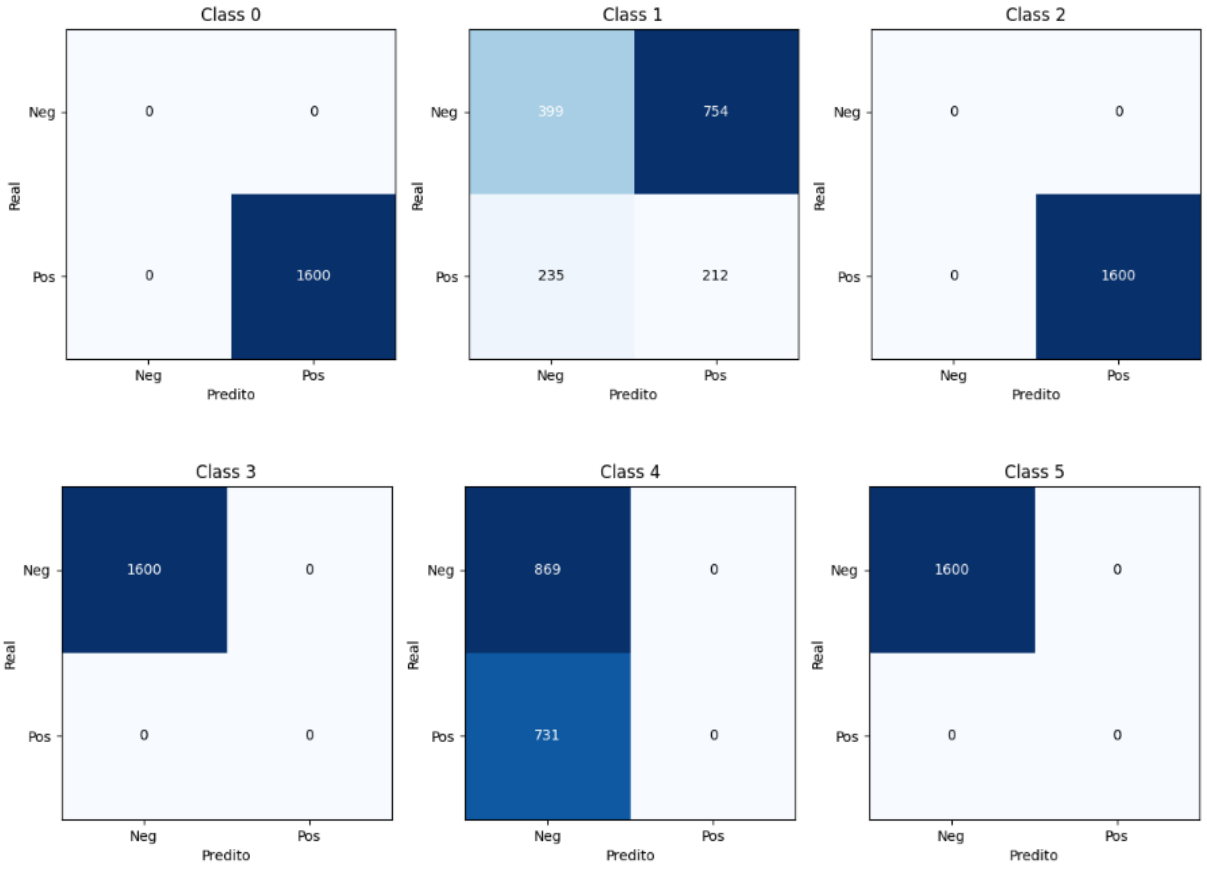
- **Vantagens:** Melhor desempenho em classes com poucos exemplos, maior potencial de generalização, especialmente útil em dados com alta variabilidade estrutural.

- **Desvantagens:** Elevado custo computacional, maior tempo de treinamento e inferência.

```
Classification Report:
                precision    recall  f1-score   support

     Class 0       1.00      1.00      1.00      1600
     Class 1       0.22      0.47      0.30       447
     Class 2       1.00      1.00      1.00      1600
     Class 3       0.00      0.00      0.00         0
     Class 4       0.00      0.00      0.00       731
     Class 5       0.00      0.00      0.00         0

   micro avg       0.82      0.78      0.80      4378
   macro avg       0.37      0.41      0.38      4378
weighted avg       0.75      0.78      0.76      4378
 samples avg       0.84      0.80      0.80      4378
```



Desempenho Comparativo

Os três modelos obtiveram acurácias globais semelhantes (~98.4%), reflexo da predominância de poucas classes nas amostras. Contudo, ao avaliar o recall por classe,

observa-se que o **Modelo 3 (Docinho)** apresentou o melhor desempenho para a Classe 1, uma das menos representadas no conjunto de dados. Ainda assim, nenhuma das arquiteturas foi capaz de classificar corretamente as classes 4 e 5, indicando a necessidade de estratégias adicionais, como balanceamento de dados ou funções de perda adaptativas.

Considerações sobre os modelos customizados

A escolha do modelo ideal depende diretamente do contexto da aplicação. Em cenários com restrições de hardware ou foco em classes frequentes, o **Modelo 1 (Lindinha)** é a escolha mais eficiente. Já o **Modelo 2 (Florzinha)** representa um meio-termo entre desempenho e custo, enquanto o **Modelo 3 (Docinho)** é mais indicado para aplicações críticas que demandam sensibilidade a padrões raros, desde que haja suporte computacional compatível.

Considerações finais

Com base nos resultados vistos, é possível observar que o **ValDir** apresentou resultados melhores, o que já era de se esperar, visto que no modelo base foram implementadas técnicas de regularização mais robustas em conjunto com uma arquitetura consolidada. Além disso, no modelo base também foi utilizada *transforms.Normalize()* e redimensionamento uniforme, garantindo mais consistência entre os dados de entrada. Enquanto isso, os modelos customizados não utilizam os artifícios citados, fato que pode limitar sua capacidade de generalização.