Universidade Estadual de Campinas
Instituto de Computação
MO434 - Deep Learning

Relatório 2 - Redes Neurais Convolucionais Para Detecção de Larvas

Maria Fernanda Bosco

1S/2025

Conteúdo

1 Introdução **1**

2 Desenvolvimento **1**
2.1 Primeira rede: modelo baseline 1
2.2 Adição de outros tipos de blocos 3
2.2.1 Segunda rede: inclusão de bloco residual 3
2.2.2 Terceira rede: inclusão de bloco *inception* 4
2.2.3 Quarta rede: inclusão de bloco de atenção SE 5
2.3 Redes com modelos pré-treinados 7
2.3.1 Enconder VGG16 7
2.3.2 Encoder ResNet50 8

3 Resultados **8**
3.1 Modelo baseline 9
3.2 Modelo Residual+Inception+SE 11
3.3 Modelos Pré-treinados 13

4 Conclusão **15**

5 Anexos - Relatório 1 **17**

1 Introdução

Este relatório apresenta uma análise comparativa entre diferentes arquiteturas de redes neurais aplicadas à classificação de imagens de larvas. Foram avaliados modelos treinados do zero e modelos pré-treinados (VGG16, ResNet50).

Nesse problema existem duas categorias possíveis sobre as quais queremos queremos fazer a classificação das imagens: larvas e restos alimentares. O dataset é composto por 1598 imagens totais, dentre as quais 247 são de larvas e 1351 são de restos alimentares. Ou seja, temos um desbalanceamento entre as classes, onde a classe de interesse (larva) representa $\sim 15.5\%$ de todo o dataset. A imagem 1 apresenta um exemplo de cada classe.

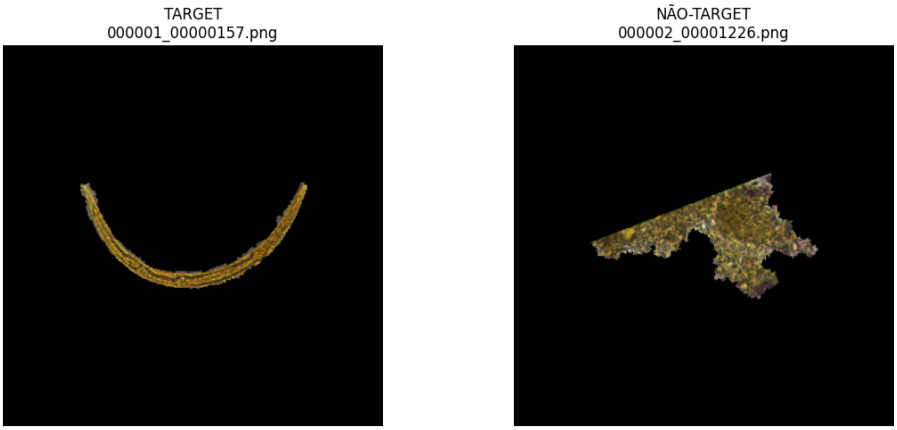

Figura 1: imagem à esquerda: larva, imagem à direita: restos alimentares

Para a realização do treinamento das redes analisadas os dataset foi divido em treino (50%), validação (20%) e teste (30%), mantendo as proporções entre classes e foram criados 3 *jupyter notebooks*:

- `01-baseline-model.ipynb`: contendo a construção de uma rede convolucional baseline mais simples;
- `02-improved-models.ipynb`: contendo a construção de três redes com acréscimos de outros tipos blocos (residual, inception e de atenção);
- `03-pretrained-models.ipynb`: contendo a construção de duas redes usando encoders pré-treinados.

No desenvolvimento do relatório iremos explorar as diferentes redes propostas para a clasificação, avaliando qual o impacto do uso de diferentes blocos.

2 Desenvolvimento

2.1 Primeira rede: modelo baseline

A primeira rede proposta (*LarvaeNet*), definida no notebook `01-baseline-model.ipynb`, é composta por dois blocos principais: extrator de *features* e classificador. O extrator de features possui dois blocos convolucionais com kernel de 5x5, seguidos de *batch normalization*, ativação ReLU e *max pooling*. Já o classficador possui uma camada de *flatten*, que atacha as saídas do extrator em um vetor 1D, uma camada linear seguida por *dropout* de

30% e ativação ReLU e, por fim, uma segunda camada linear com 2 saídas, correspondentes as classes preditas. A figura 2 apresenta uma representação gráfica mais detalhada do esquema da rede.

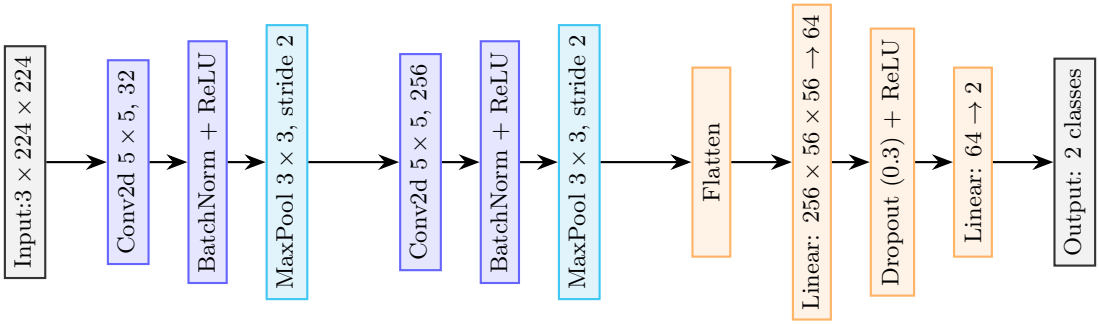

Figura 2: Arquitetura LarvaeNet

Para o treinamento da rede, foram testadas diversas configurações, incluindo:

- presença ou ausência da camada de *dropout* no classificador;
- balanceamento dos batches do *DataLoader* do conjunto de treino por meio do uso de `WeightedRandomSampler`;
- regularização L2 com pesos iguais a 1×10^{-4}, 1×10^{-3} e 1×10^{-2};
- taxas de aprendizado de 1×10^{-5}, 1×10^{-4} e 2×10^{-4};
- uso de *learning rate scheduler* com decaimento exponencial a cada época ou *step decay* a cada 5 épocas.

Os parâmetros foram ajustados conforme a necessidade, com base nos resultados dos testes de treinamento e na análise das curvas de *loss*, acurácia e Kappa de Cohen. Nas configurações iniciais, observou-se um forte indício de *overfitting*, com grande discrepância entre as curvas de treino e validação. Para mitigar esse efeito, foi introduzido um *dropout* de 30% na camada de classificação e adotado um coeficiente de regularização L2 igual a 0.01.

Também foram identificadas dificuldades de convergência, evidenciadas por grande oscilação nas métricas de validação. Para lidar com esse problema, a taxa de aprendizado foi ajustada para 2×10^{-4} e foi utilizado o agendador `StepLR` com `step` igual a 5 e fator de decaimento de 0.5, que se saiu melhor que o *scheduler* com decaimento a cada época.

Além disso, ao treinar a rede com o *dataset* original (sem balanceamento), as curvas de *loss* e acurácia aparentaram certa estabilidade, mas a métrica Kappa apresentou um desnível acentuado entre treino e validação, indicando que o modelo tinha dificuldade em aprender a classe minoritária. Com isso, definiu-se a seguinte configuração final de treinamento para o modelo *baseline*:

- uso de *DataLoader* de treino balanceado por classe;
- *dropout* de 30% na camada do classificador;
- regularização L2 com coeficiente igual a 0.01;
- taxa de aprendizado inicial de 2×10^{-4};

- *learning rate scheduler* do tipo `StepLR`, com `step` = 5 e multiplicador = 0.5.

O treinamento foi realizado por apenas 30 épocas, devido a limitações computacionais. A imagem 3 apresenta as curvas de treinamento da rede final, onde é possível observar que a *loss* cai de maneira consistente, indicando que o modelo está aprendendo. A acurácia e o índice Kappa aumentam de forma consistente, atingindo valores elevados e estáveis após poucas épocas. As curvas do Kappa revelam que, mesmo após ajustes nos parametros de treinamento, o modelo ainda apresenta um *overfitting* leve.

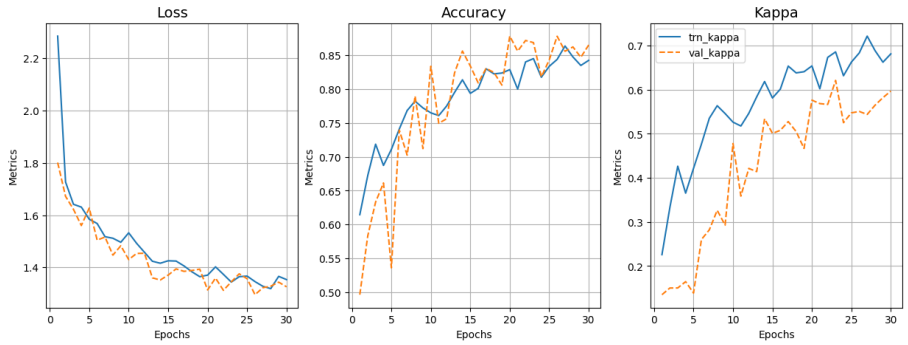

Figura 3: curvas de aprendizagem modelo *LarvaeNet*

2.2 Adição de outros tipos de blocos

2.2.1 Segunda rede: inclusão de bloco residual

Na construção da segunda rede proposta, denominada *LarvaeNetResidual* e implementada no notebook `02-improved-models.ipynb`, foi introduzido um bloco residual inicial em substituição ao primeiro bloco convolucional convencional da *LarvaeNet*. A figura 6 apresenta um detalhamento da arquitetura final da rede.

O novo bloco residual projeta os 3 canais de entrada para 32 canais e é composto por duas camadas convolucionais com kernel 3×3, cada uma seguida por operações de *batch normalization* e ativação ReLU. O bloco inclui ainda uma *skip connection* e, ao final, uma camada de *max pooling*. O segundo bloco convolucional permaneceu com as mesmas configurações da rede anterior.

A introdução do bloco residual teve como objetivo facilitar o fluxo de gradientes e estabilizar o processo de treinamento, buscando melhorar a capacidade de generalização do modelo e permitir a exploração de arquiteturas mais profundas.

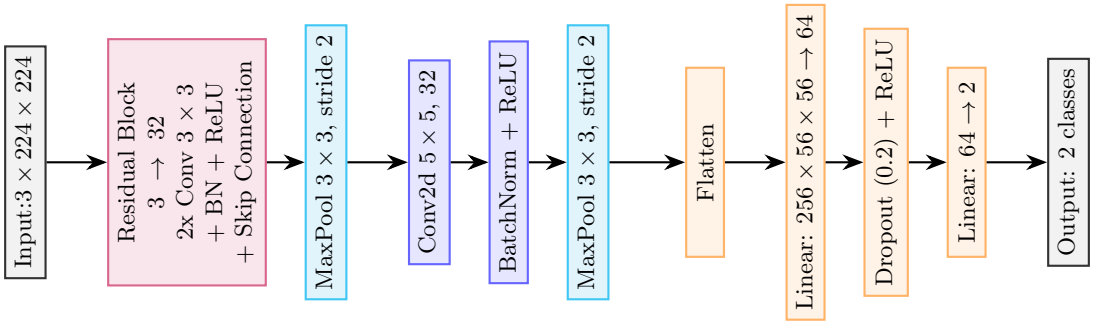

Figura 4: Arquitetura *LarvaeNetResidual*

A figura 5 mostra que, embora as curvas de *loss* e acurácia apresentem comportamento estável, há um pequeno descolamento mais entre as curvas de treino e validação da métrica Kappa, indicando a presença de um *overfitting* leve.

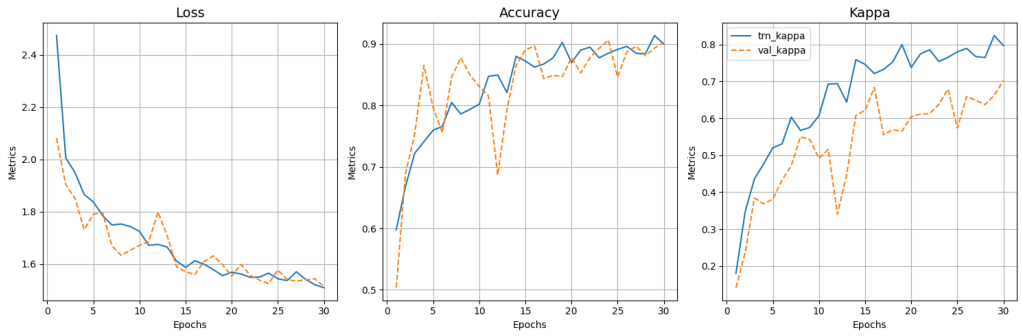

Figura 5: Curvas de aprendizado LarvaeNetResidual

2.2.2 Terceira rede: inclusão de bloco *inception*

Já na terceira rede proposta, *LarvaeNetInception*, foi incluído um bloco do tipo *inception* entre o bloco residual e o convolucional usual. O bloco inception processa a entrada em paralelo por diferentes branches e depois concatena os resultados. O bloco estruturado segue os seguintes passos:

- Redução de dimensionalidade: antes de passar pelos branches, a entrada passa por uma convolução 1x1 para reduzir o número de canais, tornando o bloco mais eficiente.
- Branch 1: convolução 1x1 para capturar padrões locais simples.
- Branch 2: convolução 1x1 seguida de convolução 3x3, capturando padrões de tamanho intermediário.
- Branch 3: convolução 1x1 seguida de convolução 5x5, capturando padrões maiores.
- Branch 4: MaxPooling 3x3 (com padding para manter o tamanho) seguido de convolução 1x1, que ajuda a capturar informações de contexto e suavizar ruídos.
- Concatenação: as saídas dos quatro branches são concatenadas ao longo da dimensão dos canais.
- Normalização e ativação: o resultado concatenado passa por uma camada de BatchNorm seguido por uma ativação ReLU.

A expectativa é de que esse bloco permita que a rede aprenda a extrair informações em diferentes escalas e contextos, tornando-a mais robusta para variações de tamanho e forma dos objetos na imagem.

A figura 7 apresenta as curvas de treinamento, no qual foram utilizadas as mesmas configurações adotadas na *LarvaeNetResidual*. Porém, as curvas mostram uma rápida convergência e estabilização e menos discrepância entre o treino e a validação. A acurácia de treino e validação atinge rapidamente patamares elevados (acima de 0.9), indicando boa capacidade de aprendizado e generalização. O índice Kappa também acompanha esse comportamento, sugerindo que o modelo está aprendendo a distinguir bem entre as classes.

Comparando com a LarvaeNetResidual, observa-se que a LarvaeNetInception tende a convergir mais rápido e alcançar métricas finais ligeiramente superiores, especialmente em

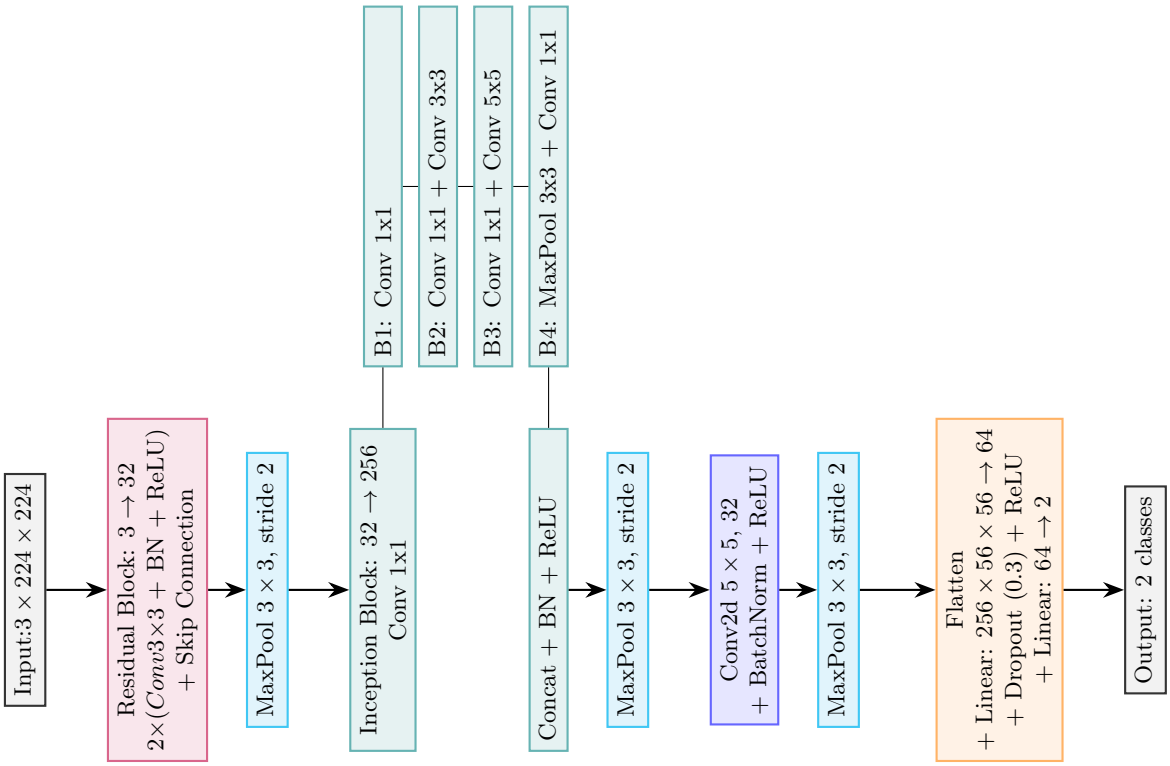

Figura 6: Arquitetura LarvaeNetInception

acurácia e Kappa de validação. Isso indica que a inclusão do bloco Inception, contribuiu para um aprendizado mais robusto e uma melhor generalização.

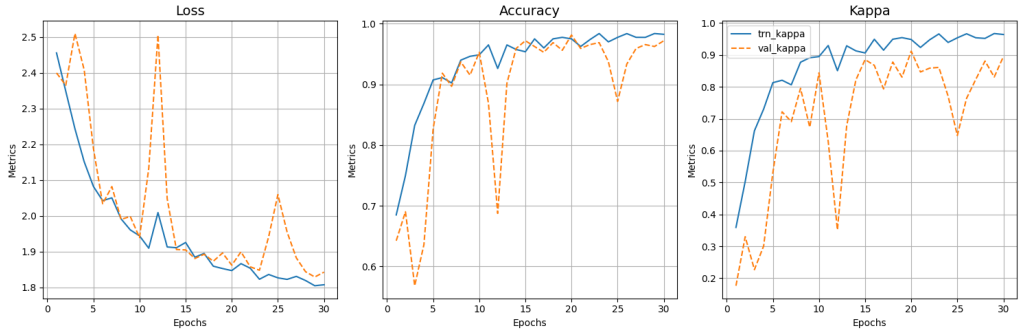

Figura 7: Curvas de aprenzado LarvaeNetInception

2.2.3 Quarta rede: inclusão de bloco de atenção SE

Na quarta rede proposta, denominada *LarvaeNetAttention*, o bloco convolucional tradicional foi substituído por um bloco do tipo *Squeeze-and-Excitation* (SE), um mecanismo de atenção de canais desenvolvido para redes neurais convolucionais. Esse bloco tem como objetivo recalibrar a importância de cada canal nas representações extraídas, permitindo que o modelo enfatize aqueles mais relevantes para a tarefa de classificação.

O bloco SE utilizado possui três etapas principais:

- **Squeeze (compressão)**: por meio de uma operação de *Global Average Pooling*,

cada canal do tensor de entrada é reduzido a um único valor, resultando em um vetor de tamanho igual ao número de canais.

- **Excitation**: esse vetor comprimido é passado por duas camadas *fully connected*. A primeira reduz a dimensionalidade por um fator de 16 e aplica uma ativação ReLU; a segunda restaura a dimensão original e aplica uma função sigmoide, gerando pesos entre 0 e 1 para cada canal.
- **Scale (recalibração)**: os pesos gerados são multiplicados pelos canais originais do tensor de entrada, ajustando a importância de cada canal.

Na implementação desta rede, o bloco convolucional modificado consiste em uma convolução com kernel 5×5, seguida por *batch normalization* e ativação ReLU. Em seguida, aplica-se o bloco SE, finalizando com uma operação de *max pooling*. A figura 8 apresenta um diagrama ilustrativo da arquitetura final da rede.

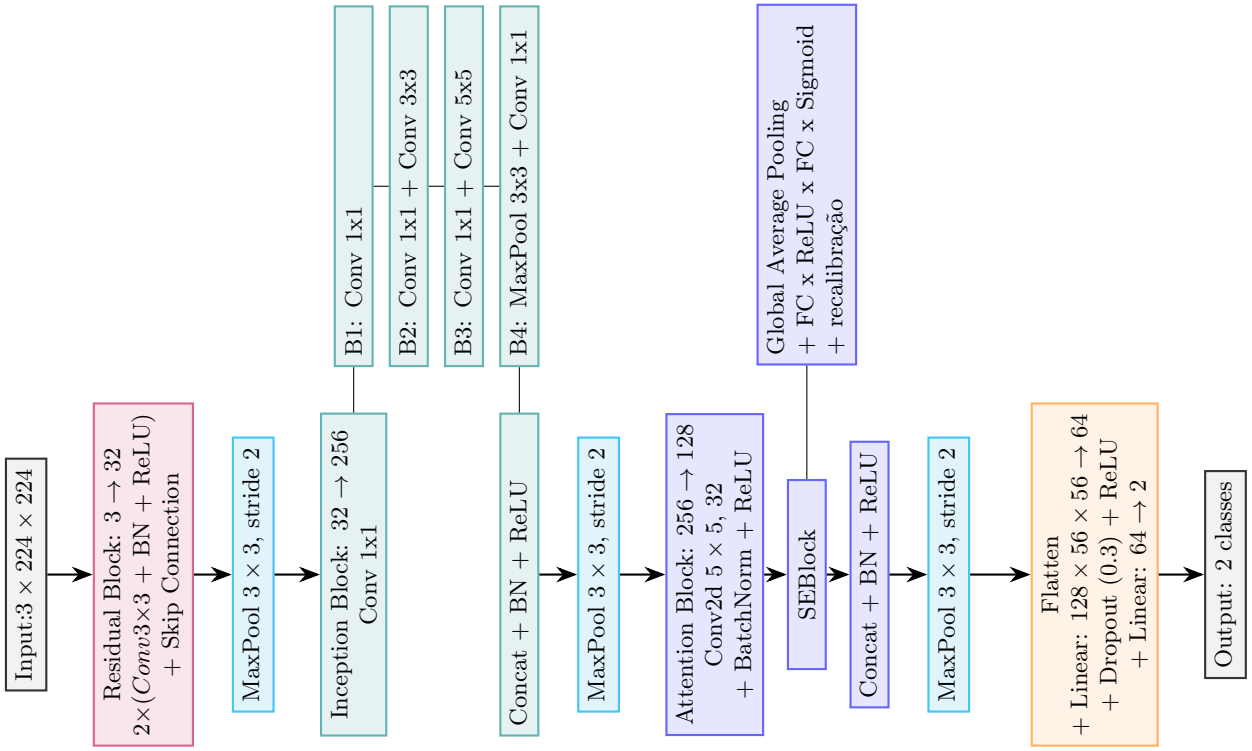

Figura 8: Arquitetura LarvaeNetAttention

Para o treinamento da rede o learning rate foi ajustado para $2e-4$ e o *learning scheduler* para decaimento com multiplicador de 0.5 a cada 5 épocas. Na figura 9 nota-se que a rede apresenta desempenho semelhante ou ligeiramente superior a *LarvarInception*, especialmente em estabilidade das métricas de validação, pois apresenta menos oscilação. Isso sugere que o bloco SE contribui para um aprendizado mais robusto e consistente, ajudando o modelo a focar nos canais mais relevantes e reduzindo possíveis flutuações causadas por ruídos ou padrões menos importantes.

A figura 9 mostra que a *LarvaeNetAttention* apresentou desempenho semelhante ou ligeiramente superior à *LarvaeInception*, especialmente em termos de estabilidade das métricas de validação, com menor oscilação ao longo das épocas. Esses resultados sugerem que o bloco SE contribui para um aprendizado mais robusto e consistente, auxiliando o

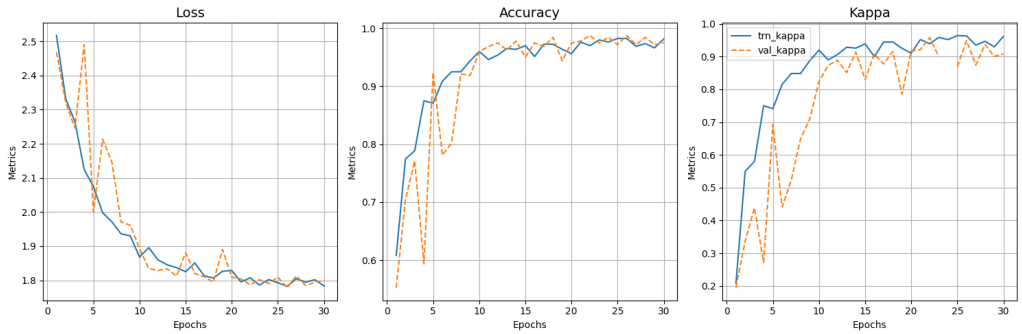

Figura 9: curvas de aprendizagem LarvaeAttention

modelo a concentrar-se nos canais mais informativos e reduzindo o impacto de ruídos ou padrões irrelevantes.

2.3 Redes com modelos pré-treinados

2.3.1 Enconder VGG16

A rede *LarvaeVGG16* foi desenvolvida como uma adaptação da arquitetura clássica *VGG16*, originalmente proposta para tarefas de classificação de imagens no ImageNet. Nesta abordagem, a *VGG16* foi utilizada como extratora de características (*feature extractor*), aproveitando-se de pesos pré-treinados no ImageNet.

A arquitetura faz uso de blocos convolucionais originais da *VGG16*, compostos por convoluções 3×3 seguidas de ativações ReLU e camadas de *max pooling*, organizados em profundidade crescente (de 64 até 512 filtros). Essas camadas foram congeladas durante o treinamento, ou seja, seus pesos não foram atualizados, permitindo que o modelo reutilize o conhecimento prévio aprendido a partir de padrões visuais genéricos.

As camadas finais da *VGG16* original foram substituídas por um classificador adaptado ao problema binário (larva e não-larva), composto por:

- *flatten* das *features* extraídas;
- camada linear com *dropout* de 50% e ativação ReLU;
- camada linear de saída.

Por se tratar de uma rede pré-treinada, o treinamento do classificador foi realizado por apenas 10 épocas. Como ilustrado na figura 10, a rede apresentou alta acurácia e bons valores da métrica Kappa, evidenciando sua capacidade de capturar os padrões relevantes da tarefa, mesmo com poucos ciclos de treinamento. No entanto, observa-se um descolamento expressivo entre as curvas de treino e validação da métrica Kappa, indicando um *overfitting* considerável.

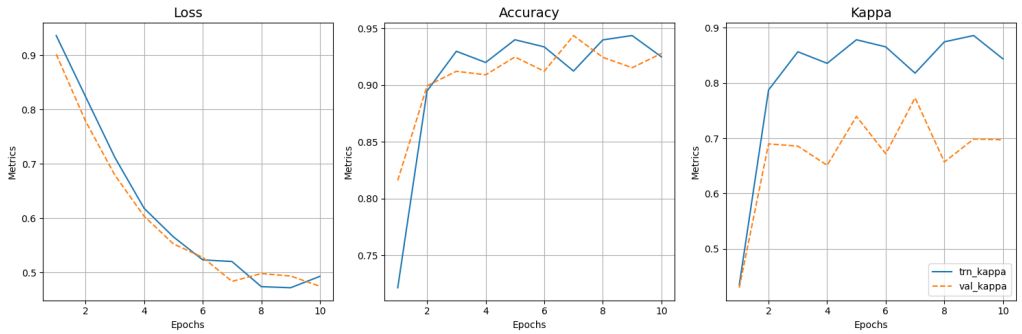

Figura 10: curvas de aprendizagem LarvaeVGG16

2.3.2 Encoder ResNet50

A rede *LarvaeResNet50* é baseada na arquitetura *ResNet-50*, uma das redes convolucionais profundas mais amplamente utilizadas para tarefas de classificação de imagens. Essa arquitetura emprega blocos residuais com *skip connections*, o que permite o treinamento de redes muito profundas, mitigando problemas de degradação do gradiente.

Neste caso, a *ResNet-50* foi utilizada como base, mantendo-se as camadas convolucionais e blocos residuais do modelo original, com pesos pré-treinados no ImageNet. Esses pesos foram congelados durante o treinamento, de modo que apenas as camadas finais — originalmente *fully connected* — foram substituídas por um classificador adaptado à tarefa binária, em configuração semelhante à adotada na rede *LarvaeVGG16*.

O treinamento foi novamente conduzido por apenas 10 épocas. Conforme pode ser observado na figura 11, o modelo apresentou rápida convergência, alta acurácia e bom desempenho de generalização. Embora nas primeiras épocas a métrica Kappa tenha exibido um desnível entre os conjuntos de treino e validação, essa diferença foi reduzida ao longo do treinamento, sugerindo uma diminuição do *overfitting*, que não se mostrou significativo.

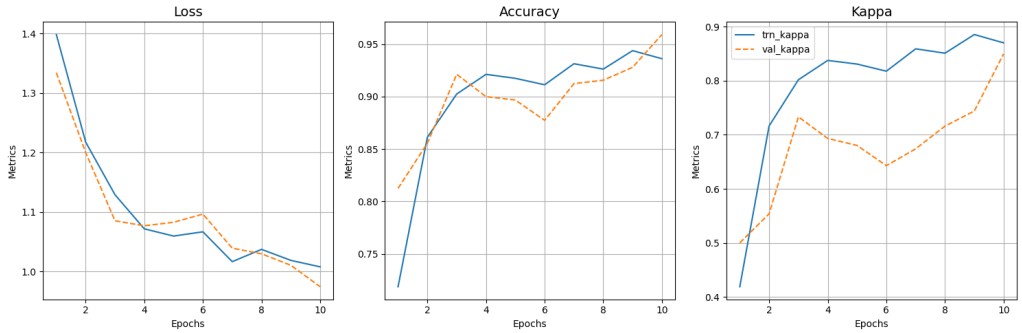

Figura 11: curvas de aprendizagem LarvaeResNet50

3 Resultados

Após o treinamento, todos os modelos foram avaliados no conjunto de teste utilizando as métricas de *loss*, acurácia e Kappa, cujos resultados estão apresentados na Tabela 1. Observa-se que os modelos baseados em arquiteturas pré-treinadas, especialmente a *VGG16*, apresentaram os menores valores de *loss*, seguidos pela *ResNet50*, o que indica uma melhor capacidade de ajuste aos dados de teste sob a ótica da função de perda.

model	test loss	test accuracy	test kappa	total parameters
Residual+Inception+SE	1.8368	0.9688	0.8753	7411202
Residual+Inception	1.8830	0.9625	0.8516	7409154
ResNet50	0.9991	0.9313	0.7557	262530
VGG16	0.4757	0.9250	0.7501	65922
Residual	1.5565	0.8896	0.6338	51596098
Baseline	1.3620	0.8375	0.5064	51588194

Tabela 1: resultados métricas no *dataset* de teste para cada modelo

Entretanto, ao considerar as métricas de acurácia e Kappa, os modelos treinados do zero com a introdução de blocos especializados apresentaram resultados superiores, em especial o modelo com bloco SE. Isso sugere que, embora os modelos pré-treinados possuam boa capacidade de reconstrução do padrão global dos dados, os modelos desenvolvidos especificamente para o problema demonstram maior poder de generalização para as classes-alvo. Ainda assim, possivelmente, se os modelos pré-treinados fossem treinados por mais épocas poderiam alcançar os resultados dos modelos específicos.

3.1 Modelo baseline

Para uma análise mais aprofundada do desempenho do modelo *baseline*, foram avaliadas as projeções das representações internas por meio das técnicas UMAP e t-SNE. A figura 12 apresenta a projeção das ativações da última camada de *features*, enquanto a figura 13 mostra as projeções após a última camada do classificador.

Observa-se que, ao sair da camada de *features*, as representações das classes ainda se encontram bastante misturadas, com sobreposição considerável. Já após o classificador, as classes passam a se organizar de forma mais distinta no espaço, indicando que o modelo está de fato aprendendo representações progressivamente mais discriminativas. No entanto, a separação ainda não é perfeitamente nítida, o que revela limitações na capacidade de distinção entre as classes — coerente com os resultados de acurácia observados.

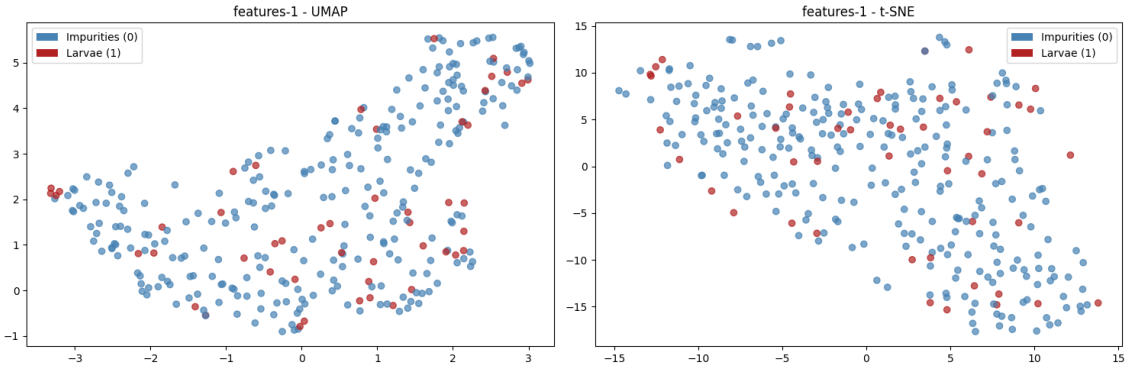

Figura 12: projeções UMAP e t-SNE para última camada do extrator de features da rede LarvaeNet

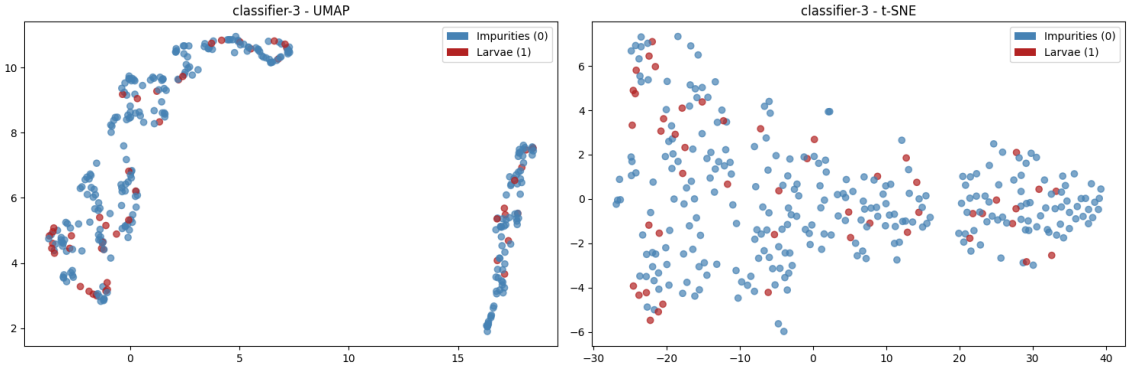

Figura 13: projeções UMAP e t-SNE para última camada do classificador de features da rede LarvaeNet

Para investigar qualitativamente os padrões de acerto e erro do modelo, foram analisadas quatro amostras utilizando mapas de atenção gerados pelo Grad-CAM:

- larva corretamente classificada como larva (`000001_00000039.png`) – imagem 14;
- impureza corretamente classificada como impureza (`000002_00000010.png`) - imagem 15;
- larva incorretamente classificada como impureza (`000001_00000056.png`) - imagem 16;
- impureza incorretamente classificada como larva (`000002_00000302.png`) - imagem 17.

Nos exemplos corretamente classificados, é possível identificar padrões distintos de ativação. No caso da larva corretamente classificada, as regiões de atenção se concentram ao longo do corpo curvado da larva, com destaque central e bordas atenuadas — sugerindo que o modelo está aprendendo a focar nos elementos mais relevantes. Por outro lado, no exemplo da impureza corretamente classificada, observa-se ausência significativa de ativação, indicando que o modelo não identificou padrões similares aos da classe positiva.

Esse mesmo padrão de ausência de atenção aparece no exemplo da larva classificada incorretamente como impureza, sugerindo que o modelo pode ter dificuldades em detectar larvas mais finas ou esticadas. No exemplo de impureza incorretamente classificada como larva, observa-se ativação difusa em regiões do objeto, o que indica possível confusão causada por formas ou texturas semelhantes às larvas curvas. Esses casos reforçam que, apesar do modelo captar representações úteis, ainda há limitações na robustez da detecção em casos ambíguos.

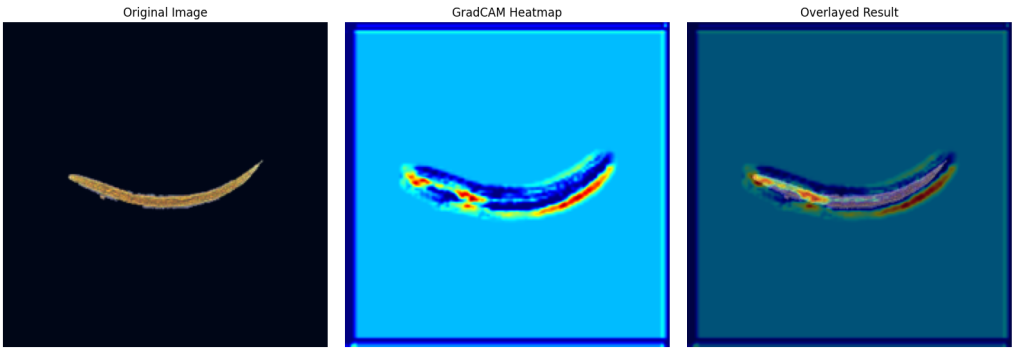

Figura 14: GradCAM Heatmap para exemplo de larva predita corretamento pelo modelo baseline

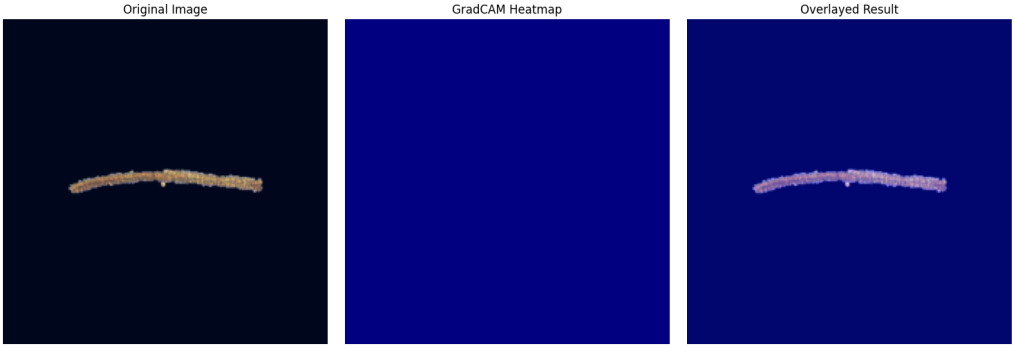

Figura 15: GradCAM Heatmap para exemplo de resto predito corretamente pelo modelo baseline

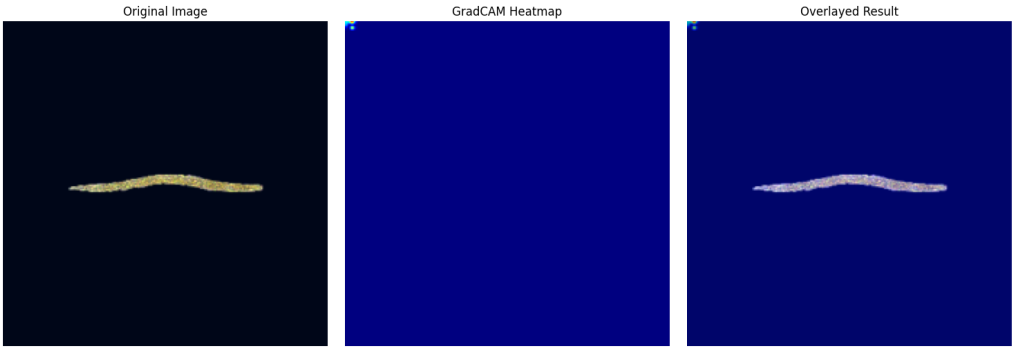

Figura 16: GradCAM Heatmap para exemplo de larva predita incorretamente pelo modelo baseline

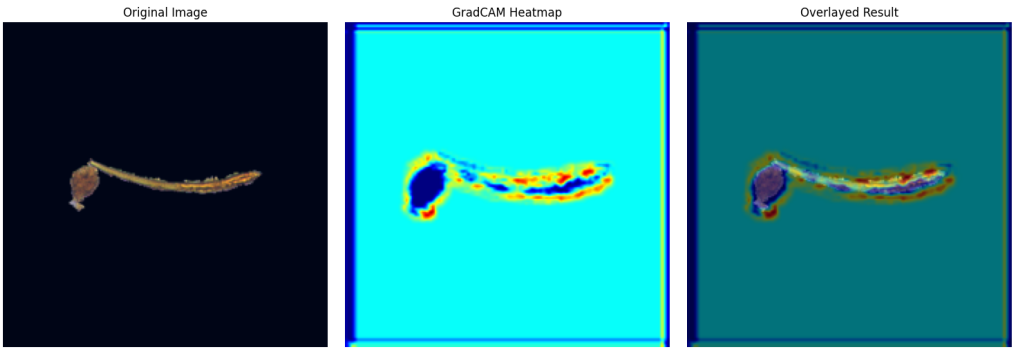

Figura 17: GradCAM Heatmap para exemplo de resto predito incorretamente pelo modelo baseline

3.2 Modelo Residual+Inception+SE

O modelo Residual+Inception+SE foi escolhido dentre os modelos com blocos específicos para ser analisado mais aprofundadamente, pois obteve melhor performance entre os três (Residual, Residual+Inception, Residual+Inception+SE).

Quando analisamos as projeções UMAP e t-sne da última camada de features e última camada do classificador nota-se um compartamento semelhante ao do modelo baseline: vemos que ao sair da camada de features as classes ainda se encontram bem misturada e ao sair do classifica- dor já estáo mais separadas (pela projeção UMAP). Neste caso, é possível observar um terceiro conjunto sendo formado, possívelmente algum grupo de

imagens que pode estar confundindo o aprendizado do modelo possa estar forçando o aparecimento de três grupos ao invés de três como esperado.

Para este modelo, dos exemplos de amostras analisados na última sessão, o exemplo de resto predito incorretamente passou a ser classificado de forma correta. Analisando os mapas de atenção para este modelo, nota-se que para a imagem de larva que continou sendo predita erroneamente como resto alimentar o mapa não consegue identificar regiões de atenção, indicando que o modelo ainda pode estar sofrendo com casos de larvas mais finas.

Dentre os modelos com blocos especializados — Residual, Residual+Inception e Residual+Inception+SE —, o último foi selecionado para uma análise mais aprofundada, por ter apresentado o melhor desempenho entre os três.

As projeções UMAP e t-SNE foram novamente analisadas, tanto para a saída da última camada de *features* (do bloco de atenção) quanto para a saída do classificador. Observa-se um comportamento semelhante ao verificado no modelo *baseline*: as representações ainda se apresentam bastante misturadas após a extração de *features*, mas mostram maior separação após o classificador, conforme evidenciado principalmente pela projeção UMAP.

Neste caso, no entanto, é possível notar a formação de um terceiro agrupamento de amostras, o que pode indicar a presença de um subconjunto de imagens com características ambíguas. Essa configuração pode estar dificultando o processo de aprendizado e forçando a separação em três grupos, em vez dos dois esperados para uma tarefa binária.

Entre os exemplos analisados na seção anterior, observa-se que o erro de classificação de uma impureza como larva foi corrigido neste modelo. Já o caso da larva incorretamente classificada como impureza permaneceu sem correção. A análise do mapa de atenção Grad-CAM mostra que, para essa amostra (19), o modelo continua sem ativar regiões significativas, sugerindo que ainda há dificuldades na detecção de larvas mais finas ou com formas menos características. Isso demonstra a hipótese de que a morfologia dessas larvas pode estar fora do padrão predominante no conjunto de treinamento.

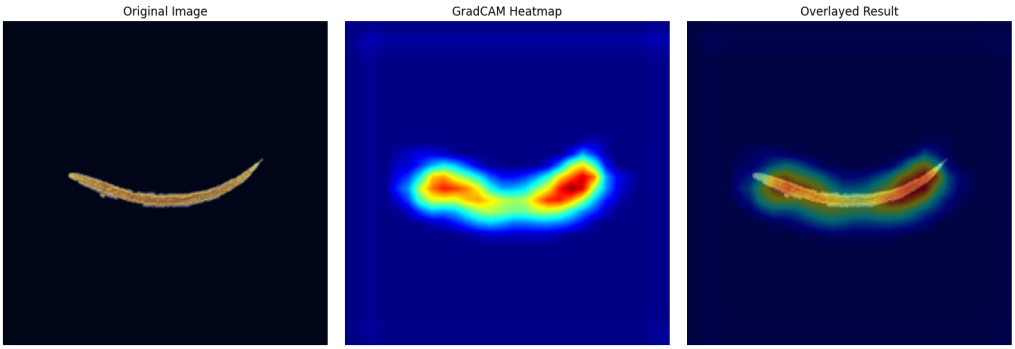

Figura 18: GradCAM Heatmap imagem `000001_00000039.png` para o modelo Residual+Inception+SE

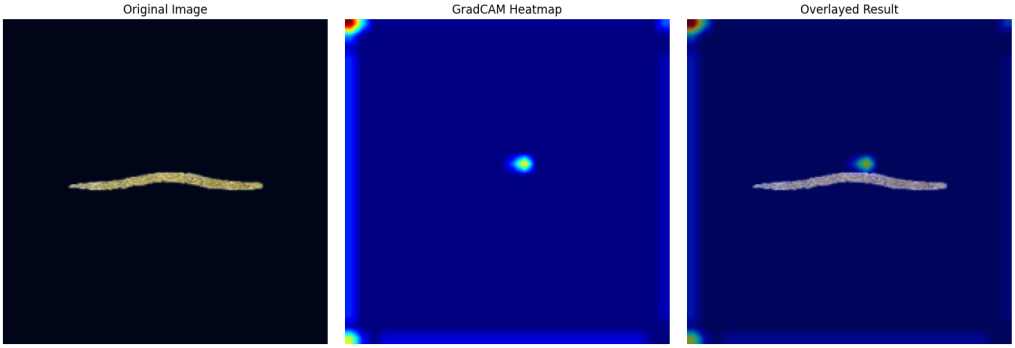

Figura 19: GradCAM Heatmap imagem `000001_00000056.png` para o modelo Residual+Inception+SE

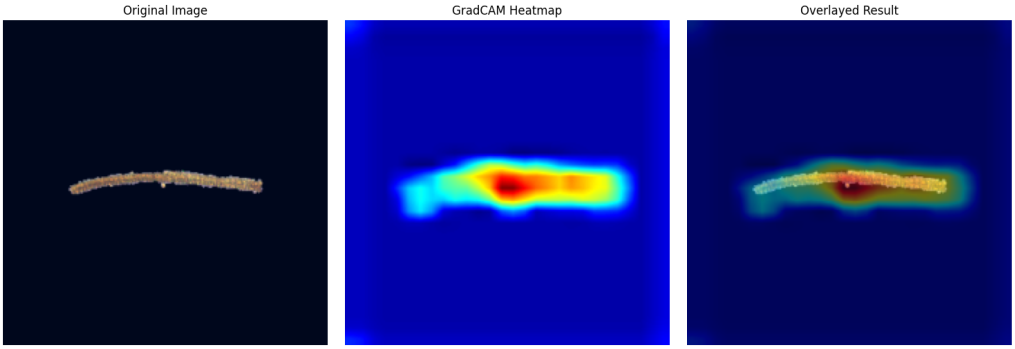

Figura 20: GradCAM Heatmap imagem `000002_00000010.png` para o modelo Residual+Inception+SE

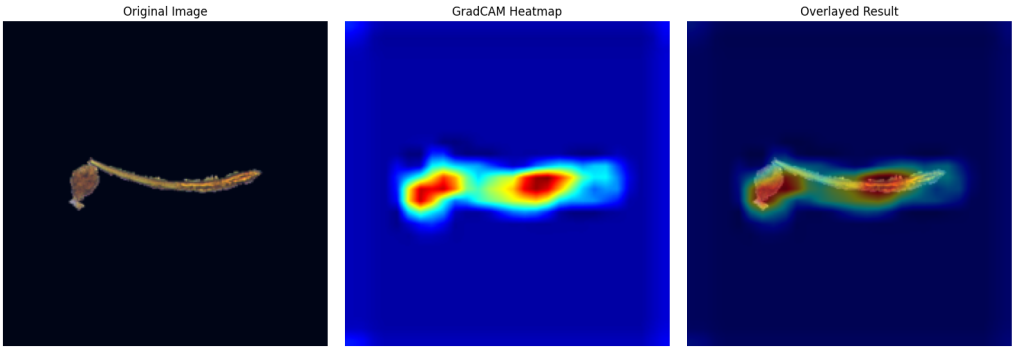

Figura 21: GradCAM Heatmap imagem `000002_00000010.png` para o modelo Residual+Inception+SE

3.3 Modelos Pré-treinados

Para uma análise mais aprofundada dos modelos pré-treinados, também foram examinadas as projeções UMAP e t-SNE , bem como os mapas de atenção gerados pelo Grad-CAM em camadas estratégicas.

As projeções das camadas da rede com *VGG16* pré-treinada podem ser consultadas na pasta `projection/vgg16_pretrained`. Observa-se, ao longo de praticamente todas as camadas, uma separação mais clara entre as classes quando comparadas às projeções do modelo *baseline*. Além disso, essa separação se mostra mais limpa e consistente, sugerindo que as *features* extraídas são mais robustas. A partir da camada 20, essa distinção entre as classes se intensifica. A figura 22 mostra a projeção da primeira camada do classificador, evidenciando uma separação bastante nítida entre as categorias.

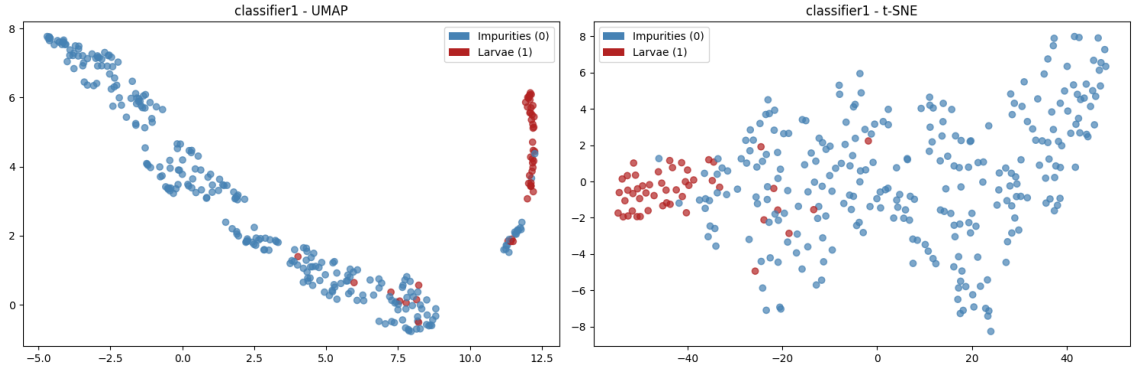

Figura 22: projeções UMAP e t-sne para a primeira camada do classificador da rede pré-treinada com VGG16

Já as projeções da rede com *ResNet50* pré-treinada estão disponíveis na pasta *projection/ResNet50_pretrained*. Nesse caso, a separação entre as classes ocorre ainda mais precocemente do que na VGG16. Já na *layer4* — última camada do extrator de *features* — as classes aparecem bem definidas, como ilustrado na Figura 23.

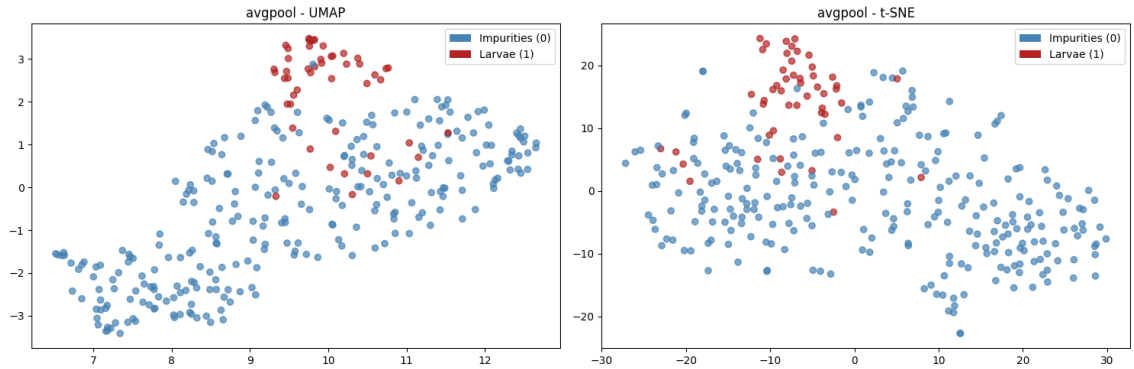

Figura 23: projeções UMAP e t-sne para a última camada do extrator da rede pré-treinada com ResNet50

No caso das redes pré-treinadas, ambas conseguiram classificar corretamente o caso da imagem `000001_00000056.png`. Na imagem 24 é possível visualizar o mapa para a rede com VGG16 e na 25 para a rede com ResNet50. Em ambas é possível notar que os modelos conseguiram capturar regiões importantes também neste caso de larva mais fina e esticada, algo que não foi possível nas demais redes, indicando um aprimoriramento nas features.

Em relação à análise dos heatmaps com GradCAM, ambas as redes pré-treinadas conseguiram classificar corretamente a imagem `000001_00000056.png`, que representa uma larva mais fina e esticada — uma amostra que havia sido mal classificada pelos modelos anteriores. As figuras 24 e 25 mostram os mapas de ativação para os modelos VGG16 e ResNet50, respectivamente. É possível notar que ambos os modelos foram capazes de identificar regiões relevantes da imagem, algo que não acontece nos outros modelos, indicando uma extração de *features* mais refinada e sensível a padrões menos evidentes.

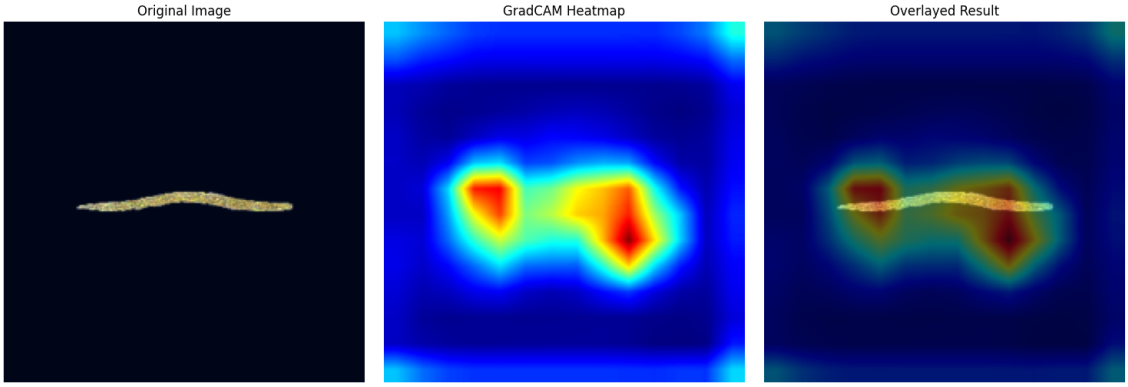

Figura 24: GradCAM Heatmap para imagem `000001_00000056.png` no modelo pré-treinado com VGG16

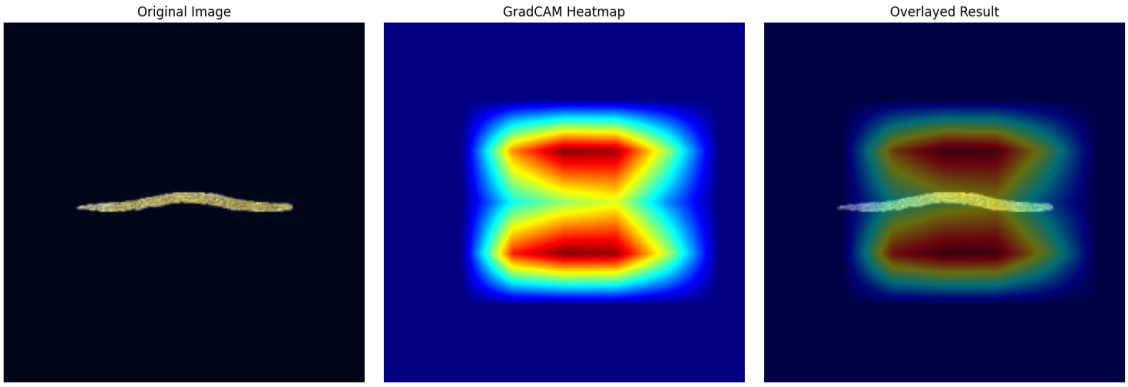

Figura 25: GradCAM Heatmap para imagem `000001_00000056.png` no modelo pré-treinado com ResNet50

4 Conclusão

Este trabalho apresentou uma análise comparativa entre diferentes arquiteturas de redes neurais convolucionais aplicadas à tarefa de detecção de larvas em imagens, com foco na avaliação do impacto de blocos especializados (residual, inception e SE) e do uso de modelos pré-treinados (VGG16 e ResNet50).

Inicialmente, foi implementado um modelo *baseline* simples, que apresentou bom desempenho inicial, mas revelou limitações na capacidade de generalização, especialmente em relação à classe minoritária. A inclusão de blocos especializados resultou em melhorias significativas, com destaque para a arquitetura *Residual+Inception+SE*, que obteve a melhor acurácia e maior índice Kappa no conjunto de teste.

As projeções UMAP e t-SNE revelaram que os modelos com blocos especializados são capazes de aprender representações mais discriminativas ao longo da rede. No entanto, observou-se também a formação de agrupamentos ambíguos em alguns casos, indicando desafios relacionados a subconjuntos de imagens com morfologias menos típicas — como larvas mais finas.

Por outro lado, os modelos com *encoders* pré-treinados demonstraram notável eficiência de treinamento, com rápida convergência e estabilidade nas métricas. Apesar de apresentarem valores inferiores de acurácia e Kappa em comparação aos modelos customizados, foram os únicos capazes de classificar corretamente amostras de maior complexidade morfológica, como evidenciado pela análise dos mapas de atenção via Grad-CAM. Isso sugere

que as *features* aprendidas a partir do ImageNet são mais sensíveis a padrões visuais sutis, mesmo com poucas épocas de treinamento. Além disso, as regiões das bordas, as texturas e os formatos da larvas se mostrarem importantes para identificação das mesmas.

De forma geral, os resultados indicam que a adição de blocos especializados contribui diretamente para a melhoria da performance global da rede, enquanto o uso de arquiteturas pré-treinadas se mostra vantajoso especialmente em contextos com dados limitados ou recursos computacionais restritos. Uma abordagem promissora para trabalhos futuros seria a combinação dessas estratégias, utilizando redes pré-treinadas misturadas com blocos adicionais ajustados para o domínio específico da tarefa.

5 Anexos - Relatório 1

Universidade Estadual de Campinas
Instituto de Computação
MO434 - Deep Learning

Relatório 1 - Simple Classification

Maria Fernanda Bosco

1S/2025

Conteúdo

1 Introdução **1**

2 Desenvolvimento **1**
2.1 Primeira rede: camada escondida inicializada e congelada 1
2.2 Segunda rede: camada escondida inicializada e treinada 3
2.3 Terceira rede: camada escondida e de decisão treinadas 4
2.4 Inicializando camada de decisão 5
2.4.1 Quarta rede: camada oculta e de decisão inicializadas sem treinamento 5
2.4.2 Quinta rede: camada oculta e de decisão inicializadas e treinadas . . 5
2.4.3 Sexta rede: Step como função de ativação 6

3 Conclusão **6**

1 Introdução

O objetivo deste relatório é explorar o uso de uma rede neural para resolver um problema simples de classificação. Nesse problema apresentam-se disponivéis duas variáveis preditoras numéricas (x_1 e x_2) e uma variável resposta categórica Y de 3 classes (c_1, c_2 e c_3). As classes são facilmente separavéis pelas variáveis preditoras, como pode ser visto na imagem 1.

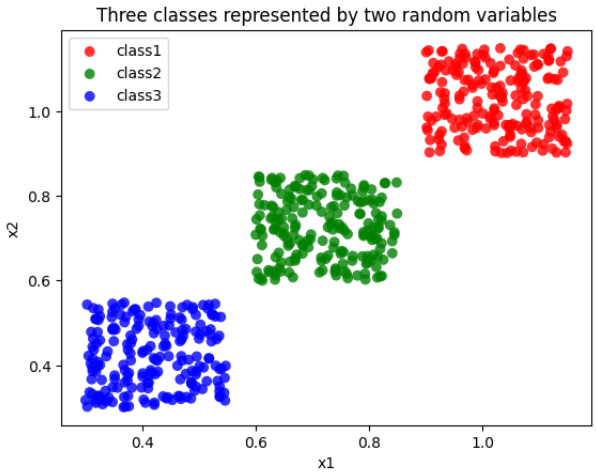

Figura 1: gráfico de dispersão das variáveis x_1 e x_2

Para a realização do treinamento da rede foi utilizado um jupyter notebook (*simple-classification.ipynb*), no qual o dataset contendo as variáveis preditoras e resposta é criado e divido em treino (50%) e teste (50%). No desenvolvimento do relatório iremos explorar diferentes redes propostas para a clasificação, avaliando qual o impacto da inicialização e treinamento de diferentes camadas e uso das funções de ativação ReLU e Step.

2 Desenvolvimento

2.1 Primeira rede: camada escondida inicializada e congelada

A primeira rede proposta para solucionar o problema de classificação apresenta somente uma camada escondida, a qual será inicializada e congelada. A camada decisão da rede será aprendida por retropropagação.

Para construção da rede é utilizado o PyTorch. Primeiramente a classe *NeuralNet* é criada, definindo uma rede neural MLP com uma camada oculta de 4 neurônios e uma camada de saída com 3 neurônios para classificação das 3 classes desejadas. A primeira camada recebe 2 entradas (as duas variáveis preditoras disponíveis) e gera 4 saídas e a segunda camada recebe 4 entradas de gera 3 saídas (uma para cada classe). A imagem 2 representa um diagrama da estrutura rede da proposta.

A função de ativação escolhida entre as camadas é a ReLU. Os pesos das camadas lineares são inicializados com o método Xavier Uniform e os *biases* como zero. O primeiro modelo (*model*) é criado, utilizando como função de perda a entropia cruzada (Cross-Entropy Loss) e o otimizador com o gradiente descendente estocástico (SGD) com taxa de aprendizado $1e^{-1}$.

É proposta uma inicialização dos pesos dos neurônios da camada oculta (*hidden layer*), com valores baseados em vetores ortogonais e distâncias à origem. O valor $v = \sqrt{2}/2 =$

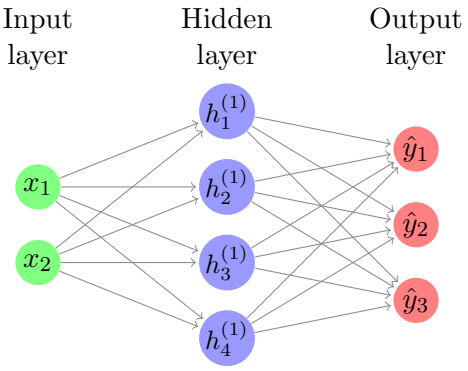

Figura 2: esquematização da rede neural proposta

0.7071 é definido para ser usado nos vetores de pesos, forçando-os a serem unitários (norma = 1) e ortogonais as linhas de separação que deseja-se criar. Ou seja, os vetores defidos são $w_1^1 = (v, v)$, $w_2^1 = (-v, -v)$, $w_3^1 = (v, v)$ e $w_4^1 = (-v, -v)$. Já os valores $b_1 = 1.2021$ e $b_2 = 0.7778$ representam as distâncias da origem para os vetores de separação de classes desejados, esses valores são utilizados para criar o vetor de *biases* da camada: $(-b_1, b_1, -b_2, b_2)$. Os vetores de peso indicam a direção e o de bieses a posição das fronteiras desejadas.

A camada inicial é congelada, o que siginifica que seus pesos não serão atualizados durante o treinamento da rede, mantendo os valores propostos definidos. A figura 3 apresenta a esquematização gráfica da inicialização proposta e a figura 4 os vetores de pesos iniciais da rede, tanto da camada oculta (inicializada), quanto de saída.

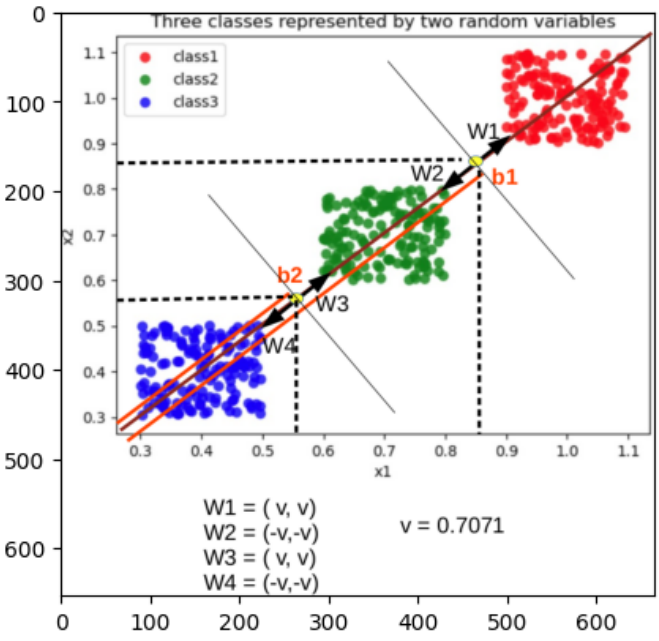

Figura 3

Em seguida a rede neural é treinada com 200 épocas. Em cada época:

- os gradientes são zerados para evitar acúmulo;
- a predição é realizada com base nas entradas (*forward pass*);

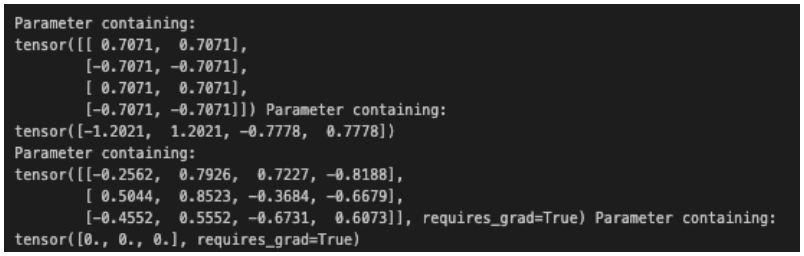

Figura 4: *model1* - vetores de pesos e *biases* iniciais da camada oculta e de saída

- a acurácia é calculada;
- a perda é calculada;
- o passo de *backward* é realizado, calculando os gradidentes dos pesos e *biases* do modelo em relação a perda;
- os valores dos pesos são atualizados.

Na figura 5 são apresentados os valores dos vetores de pesos após o treinamento da rede. Como a primeira camada foi congelada, os pesos são iguais aos definidos inicialmente. A acurácia do treinamento também é computada, sendo $acc \approx 0.98$.

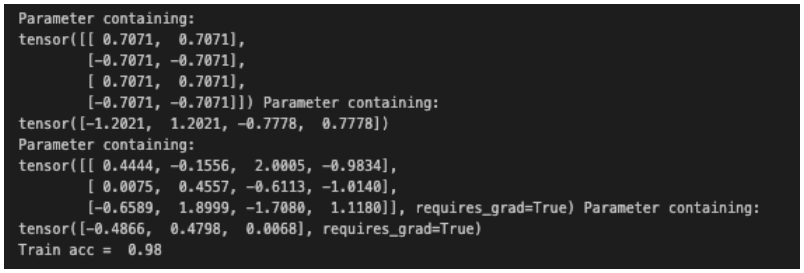

Figura 5: *model1* - vetores de pesos e *biases* da camada oculta de saída após o treinamento

Por fim, o modelo final é avaliado na base de teste, apresentando uma perda de ≈ 0.591 e acurácia $= 0.99$.

2.2 Segunda rede: camada escondida inicializada e treinada

Um segundo modelo é proposto (*model2*), agora descongelando a camada escondida. A mesma é inicializada da mesma forma que anteriormente, porém descongelada, dessa forma ela é refinada e tem seus pesos atualizados durante o treinamento da rede. A figura 6 apresenta os valores dos pesos iniciais de cada camada, agora é possível observar o parâmetro *requires_grad=True* para a primeira camada, o que garante que a mesma não está congelada.

O treinamento é feito da mesma forma apresentada na sessão 2.1, mantendo o número de épocas (200) e taxa de aprendizado ($1e^{-1}$). A figura 7 apresenta os pesos e *biases* finais da camada oculta e de saída e também a acurácia do modelo treinado, que é igual a 1. Ou seja, o modelo acertou todas predidições na base de treino.

Avaliando o modelo na base de teste foi observada $loss \approx 0.296$ e também foi obtida acurácia $= 1$, mostrando que novamente o modelo foi capaz de acertar todas as predições.

```
Parameter containing:
tensor([[ 0.7071,  0.7071],
        [-0.7071, -0.7071],
        [ 0.7071,  0.7071],
        [-0.7071, -0.7071]], requires_grad=True) Parameter containing:
tensor([-1.2021,  1.2021, -0.7778,  0.7778], requires_grad=True)
Parameter containing:
tensor([[-0.2950,  0.1182, -0.0354, -0.8927],
        [-0.3129,  0.5942,  0.3731, -0.2313],
        [-0.6265, -0.0949,  0.8909, -0.2188]], requires_grad=True) Parameter containing:
tensor([0., 0., 0.], requires_grad=True)
```

Figura 6: *model2* - vetores de pesos e *biases* iniciais das camadas escondida e de saída

```
Parameter containing:
tensor([[ 1.7039,  1.7275],
        [-0.8056, -0.8725],
        [ 0.4648,  0.4574],
        [-0.8550, -0.8892]], requires_grad=True) Parameter containing:
tensor([-1.1861,  1.8404, -1.0209,  1.8781], requires_grad=True)
Parameter containing:
tensor([[ 0.9459, -1.0629,  0.0712, -1.8647],
        [-0.0502,  0.5236,  0.3395, -0.3436],
        [-2.1300,  1.1568,  0.8179,  0.8655]], requires_grad=True) Parameter containing:
tensor([-0.3553,  0.1714,  0.1840], requires_grad=True)
Train acc =  1.0
```

Figura 7: *model2* - vetores de pesos e *biases* das camadas escondida e de saída após treinamento

2.3 Terceira rede: camada escondida e de decisão treinadas

No terceiro modelo proposto (*model3*) a rede é construída da mesma forma, mantendo a mesma estrutura e forma de treinamento dos modelos anteriores, porém nenhuma camada é inicializada por cálculo prévio. Ou seja, todos os pesos, de ambas as camadas, são inicializados de forma arbitrária e refinados por retropropagação durante o treinamento.

```
Parameter containing:
tensor([[ 0.9646,  0.1556],
        [-0.1494, -0.7459],
        [ 0.6817, -0.4306],
        [-0.6873,  0.3139]], requires_grad=True) Parameter containing:
tensor([0., 0., 0., 0.], requires_grad=True)
Parameter containing:
tensor([[-0.0639,  0.0074,  0.6332,  0.2417],
        [-0.1434, -0.4135,  0.2587,  0.6992],
        [-0.3242,  0.0611, -0.4392,  0.2545]], requires_grad=True) Parameter containing:
tensor([0., 0., 0.], requires_grad=True)
```

Figura 8: *model3* - vetores de pesos e *biases* iniciais das camadas escondida e de saída

A figura 8 mostra os pesos iniciais da rede e a figura 9 os pesos após do treinamento junto com a acurácia obtida de ≈ 0.957. Avaliando o modelo final na base de teste foi obtida uma perda de ≈ 0.518 e acurácia ≈ 0.967.

```
Parameter containing:
tensor([[ 1.4146,  0.6884],
        [-0.1494, -0.7459],
        [ 1.5631,  0.6317],
        [-0.6873,  0.3139]], requires_grad=True) Parameter containing:
tensor([-0.8797,  0.0000, -0.9131,  0.0000], requires_grad=True)
Parameter containing:
tensor([[ 0.7992,  0.0074,  1.3571,  0.2417],
        [ 0.0063, -0.4135,  0.3930,  0.6992],
        [-1.3370,  0.0611, -1.2974,  0.2545]], requires_grad=True) Parameter containing:
tensor([-1.2726,  0.1388,  1.1338], requires_grad=True)
Train acc =  0.9566666666666667
```

Figura 9: *model3* - vetores de pesos e *biases* após treinamento das camadas escondida e de saída

2.4 Inicializando camada de decisão

Nos próximos modelos propostos a camada de decisão será inicializada com os seguintes vetores de pesos: $w_1^2 = (1, 0, 0, 0)$, $w_2^2 = (0, v, v, 0)$, $w_3^2 = (0, 0, 0, 1)$. A ideia é induzir a camada de decisão para que para a classe 1 seja ativado somente o neurônio 1 da camada oculta, para a classe 2, os neurônios 2 e 3 e para a 4 somente o neurônio 4. Os *biases* são inicializados zerados.

2.4.1 Quarta rede: camada oculta e de decisão inicializadas sem treinamento

Na construção da quarta rede (*model4*), a camada de decisão é inicializada conforme definido anteriormente e a oculta conforme especificado na sessão 2.1. Dessa forma, os vetores de pesos tem os valores definidor conforme expresso na figura 10.

```
Pesos da camada oculta: Parameter containing:
tensor([[ 0.7071,  0.7071],
        [-0.7071, -0.7071],
        [ 0.7071,  0.7071],
        [-0.7071, -0.7071]])
Vieses da camada oculta: Parameter containing:
tensor([-1.2021,  1.2021, -0.7778,  0.7778])
Pesos da camada de decisão: Parameter containing:
tensor([[1.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.7071, 0.7071, 0.0000],
        [0.0000, 0.0000, 0.0000, 1.0000]])
Vieses da camada de decisão: Parameter containing:
tensor([0., 0., 0.])
```

Figura 10: *model4* - vetores de pesos e *biases* inicias das camadas escondida e de decisão

Nessa rede ambas as camada são congeladas, ou seja, nenhuma será treinada e refinada. Avaliando a rede dessa forma obtemos no dataset de treino $loss \approx 1.047$ e acurácia ≈ 0.363 e, no de teste, $loss \approx 1.059$ e acurácia de ≈ 0.303.

O modelo acaba não performando muito bem. Ao observar a matriz de confusão das classificações (figura 11) é possível notar que o modelo acaba classificando todas as observações como pertencentes a classe 2.

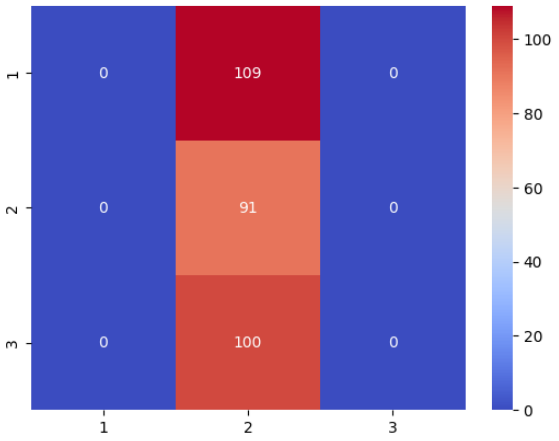

Figura 11: *model4* - matriz de confusão das classificações (eixo x = valores reais, eixo y = preditos)

2.4.2 Quinta rede: camada oculta e de decisão inicializadas e treinadas

Na construção da quinta rede (*model5*), temos a mesma inicialização da sessão 2.4.1, porém agora a rede é treinada com 200 épocas, com ambas as camadas liberadas para

treinamento. Os pesos finais obtidos estão na figura 12 juntamente com a acurácia no treinamento, que foi igual a 1.

```
Pesos da camada oculta: Parameter containing:
tensor([[ 1.5080,  1.6117],
        [-0.6795, -0.7665],
        [ 1.2558,  1.1987],
        [-0.9127, -0.8827]], requires_grad=True)
Vieses da camada oculta: Parameter containing:
tensor([-1.2905,  1.9288, -0.7001,  1.8869], requires_grad=True)
Pesos da camada de decisão: Parameter containing:
tensor([[ 1.9885, -1.0941,  0.7359, -0.8998],
        [ 0.0747,  0.8166,  0.9506, -0.0380],
        [-1.0632,  0.9846, -0.9794,  1.9377]], requires_grad=True)
Vieses da camada de decisão: Parameter containing:
tensor([-0.5177,  0.2410,  0.2768], requires_grad=True)
Train acc =  1.0
```

Figura 12: *model4* - vetores de pesos e *biases* inicias das camadas escondida e de decisão

Avaliando o modelo no conjunto de teste foi observada $loss \approx 0.256$ e acurácia $= 1$. Ou seja, com o treinamento a rede conseguiu ser refinada de forma que acertasse a totalidade das predições.

2.4.3 Sexta rede: Step como função de ativação

Como o resultado da rede com camadas de decisão inicializadas e não treinadas (*model5*) não obteve o resultado esperado, também foi testado alterar a função de ativação para a função Step, pois ela garante valor 0 ou 1.

Com essa mudança e os memos vetores de pesos inicializados conforme a figura 10 e sem treinar a rede, o modelo consegue acertar todas as classificações, atingindo acurácia $= 1$ tanto no treino quanto no teste.

3 Conclusão

Apesar de ser um problema simples de classificação proposto, o exercício de testar várias possibilidades de inicialização e treinamento da rede mostra como é possível impactá-la com as escolhas.

O primeiro modelo, com a camada oculta inicializada, já obtém uma boa performance e durante o treinamento a redução da função de perda não é tão grande, como pode ser observado na figura 13. O que é observado ao liberar a camada oculta para treinamento no segundo modelo é que, apesar de haver melhora na loss e acurácia, ela não é tão significativa. E, ao não inicializar nenhuma das camadas e aprender ambas por retropropagação obtemos a pior performance entre as 3, apesar de novamente não ser tão grande a diferença. Provavelmente se fosse um problema mais complexo de separação de classes seria observado um ganho maior no *model2* em relação ao primeiro e terceiro modelos.

No modelo 4, quando é proposta a inicialização da camada de saída o modelo não performa conforme o esperado, classificando todas as observações como pertencentes a classe 2. Porém, quando liberamos a camada para treinamento no quinto modelo, a rede consegue aprender os pesos ideais e acerta todas as classificações corretamente, obtendo a menor perda de todos os modelos propostos, conforme pode ser observado na tabela 1.

Já no modelo 6, alterando a função de ativação para a Step o modelo também consegue acertar todas classificações, mesmo sem treinamento. Isso evidência que, ao utilizar a função ReLU no modelo 4, acaba-se levando em consideração a intensidade da ativação do neurônio (distância da classe a linha de separação desejada). Ou seja, as classes 1 e 3 acabam sendo classificadas como 2, porque, apesar de terem os neurônios corretos ativados,

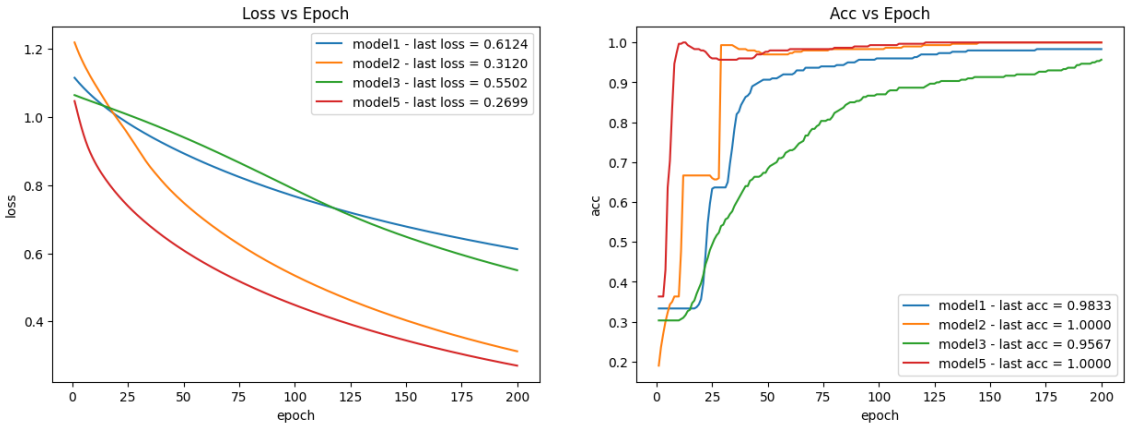

Figura 13: gráfico das perdas e acurácias dos modelos treinados em cada época

modelo	loss	acc
model1	0.591	0.99
model2	0.295	1
model3	0.517	0.967
model4	1.059	0.303
model5	0.256	1
model6	0.642	1

Tabela 1: perdas e acurácias no conjunto de teste para cada rede avaliada

o neurônio da classe dois acaba "pesando"mais, por estar mais distante das observações. Ao mudar para a função Step, todos os neurônios são ativados com a mesma intensidade, fazendo com que as rede entenda as classificações corretamente, sem necessidade de treinamento.

As diferentes construções e resultados das redes propostas, evidenciam como inicializações adequadas e escolhas ideais das funções de ativação podem impactar na performance da rede.