

Equipe - Segmentação de Vaso Sanguíneo da Retina

Alunos:

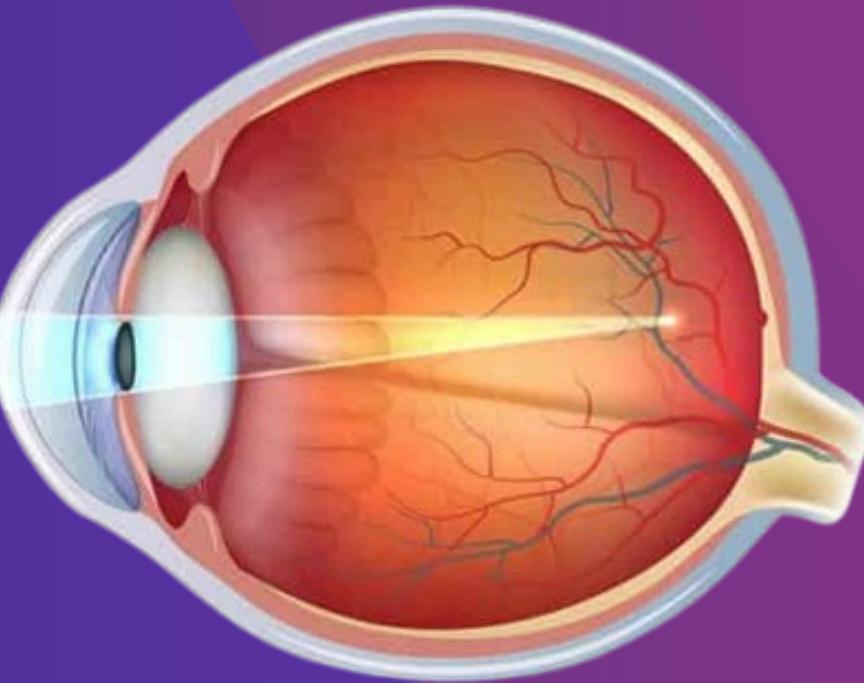
Alan Delon

Anderson

Catharina Carneiro

Gabriela Zerbone

Hanna Câmara da Justa



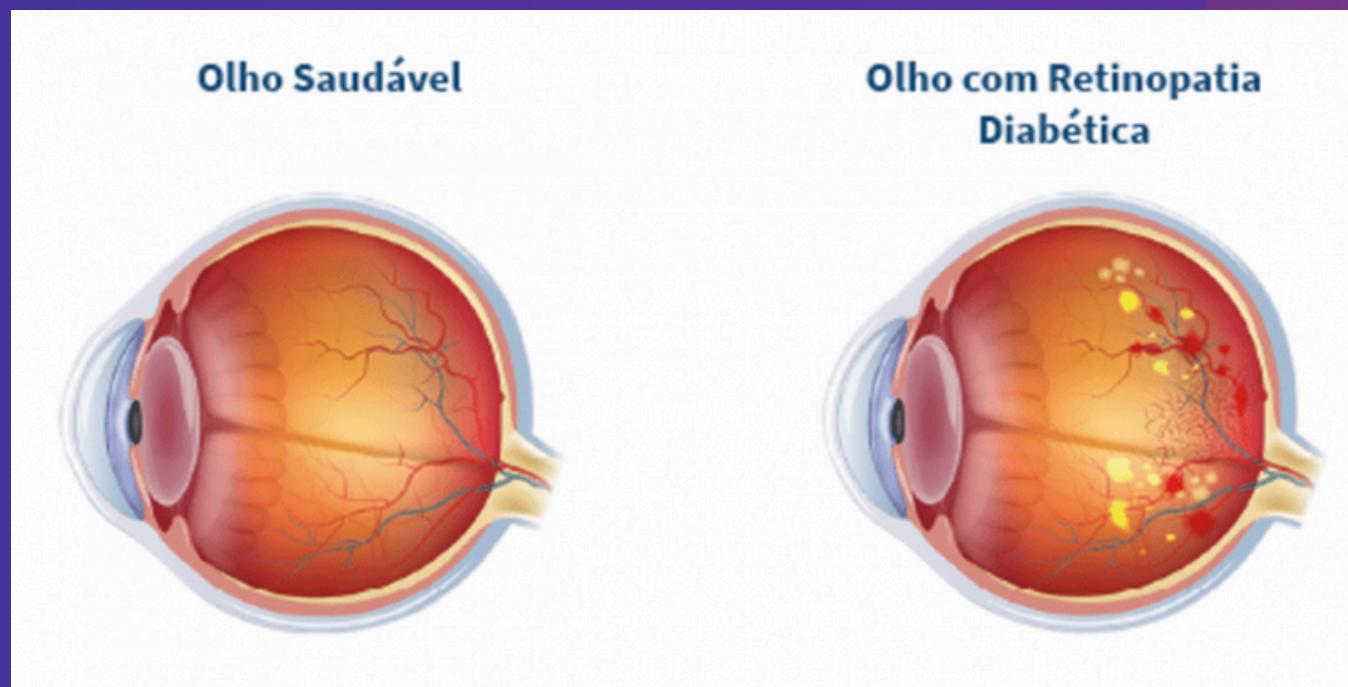
Sumário

- Introdução
- Metodologia
- Resultados
- Perspectivas futuras
- Conclusões

Introdução

O que são retinopatias?

- Patologias que afetam a retina.
- As mais comuns envolvendo os vasos: Diabética e Hipertensiva.
- As doenças aumentam os vasos sanguíneos, que se rompem e leva a hemorragia, diminuindo a visão do paciente.



Introdução

Por que esse problema é importante?

- Doenças oculares são uma das principais causas de cegueira no mundo.
- A identificação precoce essencial para a eficácia do tratamento.
- Artigos que utilizaram Redes neurais e algoritmos como random forest com alta performance na segmentação dos vasos da retina.
- As imagens podem ser integradas em um sistema automatizado para triagem de retinopatias e auxiliar no diagnóstico precoce.

Introdução



- Visto que a ML pode ser um aliado para diagnósticos das retinopatias, procuramos códigos com treinamentos baseados em um dataset do Kaggle.
- Buscamos inserir melhorias nesse código para que ele possa ser utilizado de uma forma mais eficaz.

Código escolhido:

100% TensorFlow after 1 epoch



Introdução

Porquê escolhemos esse dataset?

- Título com 1 epoch;
- 100% de acurácia;
- Código “pequeno”;
- Treinou para classificação, e não segmentação de imagens;
- Sem exploração inicial de dados;
- Sem métricas diferenciadas;
- Sem análises de overfitting.

Introdução

Qual nosso objetivo?

Propor melhorias como:

- Potencializar o modelo de treinamento;
- Explorar novas métricas;
- Tornar o modelo mais rápido;
- Análises de overfitting.

Metodologia Inicial



Divisão do dataset em 5 partes:

- 1) Exploração inicial dos dados;**
- 2) Pré processamento de imagens: Normalização e redimensionamento;**
- 3) Modelagem;**
- 4) Novas Métricas;**
- 5) Gráficos e análises de loss (overfitting).**

Em cada parte, foi procurado na literatura e no chatGPT pontos de melhoria.

Metodologia

1) Exploração inicial dos dados:

```
2  drive.mount('/content/drive', force_remount=True)
3  path = '/content/drive/MyDrive/dataset_retina'
4  dataset_path = Path(path)
5  # Definição dos caminhos das pastas de treino e teste dentro do dataset
6  pasta_treinamento = os.path.join(path, "Data", "train") # Caminho para as imagens de treino
7  pasta_teste = os.path.join(path, "Data", "test") # Caminho para as imagens de teste

1 def sumario_coluna(df):
2     dados_resumo = []
3
4     for nome_coluna in df.columns:
5         tipo_coluna = df[nome_coluna].dtype
6         valores_distintos = df[nome_coluna].nunique()
7
8         contagem_distintos = None
9         if valores_distintos <= 10:
10            contagem_distintos = df[nome_coluna].value_counts().to_dict()
11
12         dados_resumo.append([
13             nome_coluna,
14             str(tipo_coluna),
15             valores_distintos,
16             contagem_distintos
17         ])
18
19     # Markdown para exibição
20     saida_markdown = "## 📊 Sumário das Colunas\n\n"
21     saida_markdown += "| Coluna | Tipo de Dado | Valores Distintos | Distribuição dos Valores |\n"
22     saida_markdown += "|-----|-----|-----|-----|\n"
23
24     for col in dados_resumo:
25         distribuicao = col[3] if col[3] is not None else "N/A"
26         saida_markdown += f"| `{{col[0]}}` | `{{col[1]}}` | {col[2]} | {distribuicao} |\n"
27
28     display(md(saida_markdown))
29
30     # Exemplo de uso
31     sumario_coluna(df_imagens)
```

```
1 def verificar_imagens_corrompidas(caminho_pasta):
2     imagens_corrompidas = []
3
4     for raiz, _, arquivos in os.walk(caminho_pasta): # Percorre recursivamente o diretório do dataset
5         for arquivo in arquivos:
6             caminho_arquivo = os.path.join(raiz, arquivo)
7             try:
8                 with Image.open(caminho_arquivo) as imagem: # Tenta abrir cada imagem
9                     imagem.verify() # Verifica a integridade da imagem
10                except (IOError, SyntaxError) as erro: # Se ocorrer um erro, a imagem é considerada corrompida
11                    print(f"Imagen corrompida encontrada: {caminho_arquivo} - Erro: {erro}")
12                    imagens_corrompidas.append(caminho_arquivo)
13
14    return imagens_corrompidas
15
16 # Executa a verificação e exibe o número de imagens corrompidas
17 imagens_corrompidas = verificar_imagens_corrompidas(path)
18 print(f"Número de imagens corrompidas: {len(imagens_corrompidas)})
```

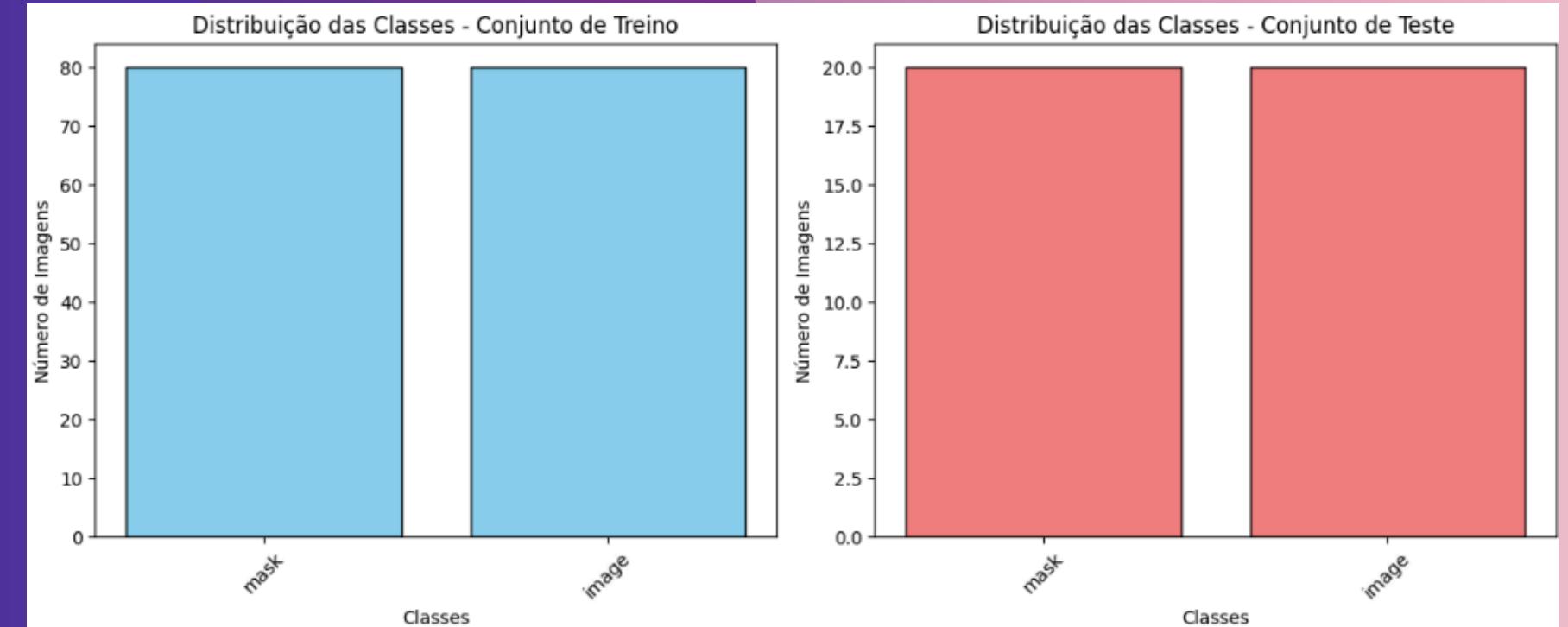
Fizemos funções para explorar os dados iniciais do dataset como: Tamanhos das imagens, quantidade de imagens, imagens duplicadas, corrompidas, etc.

Resultados

1) Exploração inicial dos dados:

 **Sumário das Colunas**

Coluna	Tipo de Dado	Valores Distintos	Distribuição dos Valores
arquivo	object	80	N/A
caminho	object	200	N/A
categoria	object	2	{'mask': 100, 'image': 100}
largura	int64	1	{512: 200}
altura	int64	1	{512: 200}
num_pixels	int64	1	{262144: 200}
modo	object	2	{'L': 100, 'RGB': 100}



🔍 Duplicatas em imagens:
Nenhuma duplicata encontrada em /content/drive/MyDrive/dataset_retina/Data/train/image
Nenhuma duplicata encontrada em /content/drive/MyDrive/dataset_retina/Data/test/image

🔍 Duplicatas em máscaras:
Nenhuma duplicata encontrada em /content/drive/MyDrive/dataset_retina/Data/train/mask
Nenhuma duplicata encontrada em /content/drive/MyDrive/dataset_retina/Data/test/mask

Coletamos as informações gerais: 200 imagens totais, 80% selecionadas para treino, 20% para teste, não há imagens duplicadas, todas possuem o mesmo tamanho.

Metodologia

2) Pré processamento de imagens: Normalização e redimensionamento:

```
# Função de pré-processamento para carregar e redimensionar as imagens
def carregar_e_preprocessar_dados(pasta_imagem, pasta_mascara, tamanho_alvo=(128, 128)):

    # Verificar se as pastas existem
    if not os.path.exists(pasta_imagem):
        raise FileNotFoundError(f"Pasta de imagens não encontrada: {pasta_imagem}")
    if not os.path.exists(pasta_mascara):
        raise FileNotFoundError(f"Pasta de máscaras não encontrada: {pasta_mascara}")

    # Carregar e redimensionar as imagens e suas máscaras
    imagens = []
    mascaras = []

    for arquivo_img, arquivo_mask in tqdm(zip(archivos_imagem, arquivos_mascara), total=len(archivos_imagem)):
        # Carregar imagem e redimensionar
        caminho_imagem = os.path.join(pasta_imagem, arquivo_img)
        img = Image.open(caminho_imagem)
        img = img.resize(tamanho_alvo)
        img_array = np.array(img)

        # Carregar máscara e redimensionar
        caminho_mascara = os.path.join(pasta_mascara, arquivo_mask)
        mask = Image.open(caminho_mascara)
        mask = mask.resize(tamanho_alvo)
        mask_array = np.array(mask)

        # Converter máscara para binário (0 ou 1)
        if len(mask_array.shape) == 3 and mask_array.shape[2] > 1:
            # Se a máscara tiver múltiplos canais, converter para escala de cinza
            mask_array = np.mean(mask_array, axis=2)

        # Binarizar a máscara (limiar em 128)
        mask_array = (mask_array > 128).astype(np.float32)

        imagens.append(img_array)
        mascaras.append(mask_array)
```

```
# Processar cada par de imagem e máscara
for arquivo_img, arquivo_mask in tqdm(zip(archivos_imagem, arquivos_mascara), total=len(archivos_imagem)):
    # Carregar imagem e redimensionar
    caminho_imagem = os.path.join(pasta_imagem, arquivo_img)
    img = Image.open(caminho_imagem)
    img = img.resize(tamanho_alvo)
    img_array = np.array(img)

    # Carregar máscara e redimensionar
    caminho_mascara = os.path.join(pasta_mascara, arquivo_mask)
    mask = Image.open(caminho_mascara)
    mask = mask.resize(tamanho_alvo)
    mask_array = np.array(mask)

    # Converter máscara para binário (0 ou 1)
    if len(mask_array.shape) == 3 and mask_array.shape[2] > 1:
        # Se a máscara tiver múltiplos canais, converter para escala de cinza
        mask_array = np.mean(mask_array, axis=2)

    # Binarizar a máscara (limiar em 128)
    mask_array = (mask_array > 128).astype(np.float32)
```

```
# Criar datasets do TensorFlow para treinamento e teste
def criar_tf_dataset(imagens, mascaras, tamanho_batch=32, shuffle=True):

    dataset = tf.data.Dataset.from_tensor_slices((imagens, mascaras))

    if shuffle:
        dataset = dataset.shuffle(buffer_size=len(imagens))

    dataset = dataset.batch(tamanho_batch)
    dataset = dataset.prefetch(tf.data.AUTOTUNE)
```

```
# Criar datasets do TensorFlow para treinamento e teste
def criar_tf_dataset(imagens, mascaras, tamanho_batch=32, shuffle=True):

    dataset = tf.data.Dataset.from_tensor_slices((imagens, mascaras))

    if shuffle:
        dataset = dataset.shuffle(buffer_size=len(imagens))

    dataset = dataset.batch(tamanho_batch)
    dataset = dataset.prefetch(tf.data.AUTOTUNE)
```

Fizemos funções para: validar se as pastas eram lidas, redimensionavam as imagens para um tamanho padrão, binarizavam e normalizavam as máscaras. Também foram feitas funções para criar um dataset do TensorFlow com Batching/Shuffling e Prefetch

Resultados

2) Pré processamento de imagens: Normalização e redimensionamento:

Essa funções são importantes pois precisamos verificar se as pastas de imagem e máscara existem, especialmente em um ambiente colaborativo. A função de redimensionamento de imagens de tamanho padrão é importante pois em ML é exigido entradas com dimensões fixas;

A binarização facilita o aprendizado do modelo;

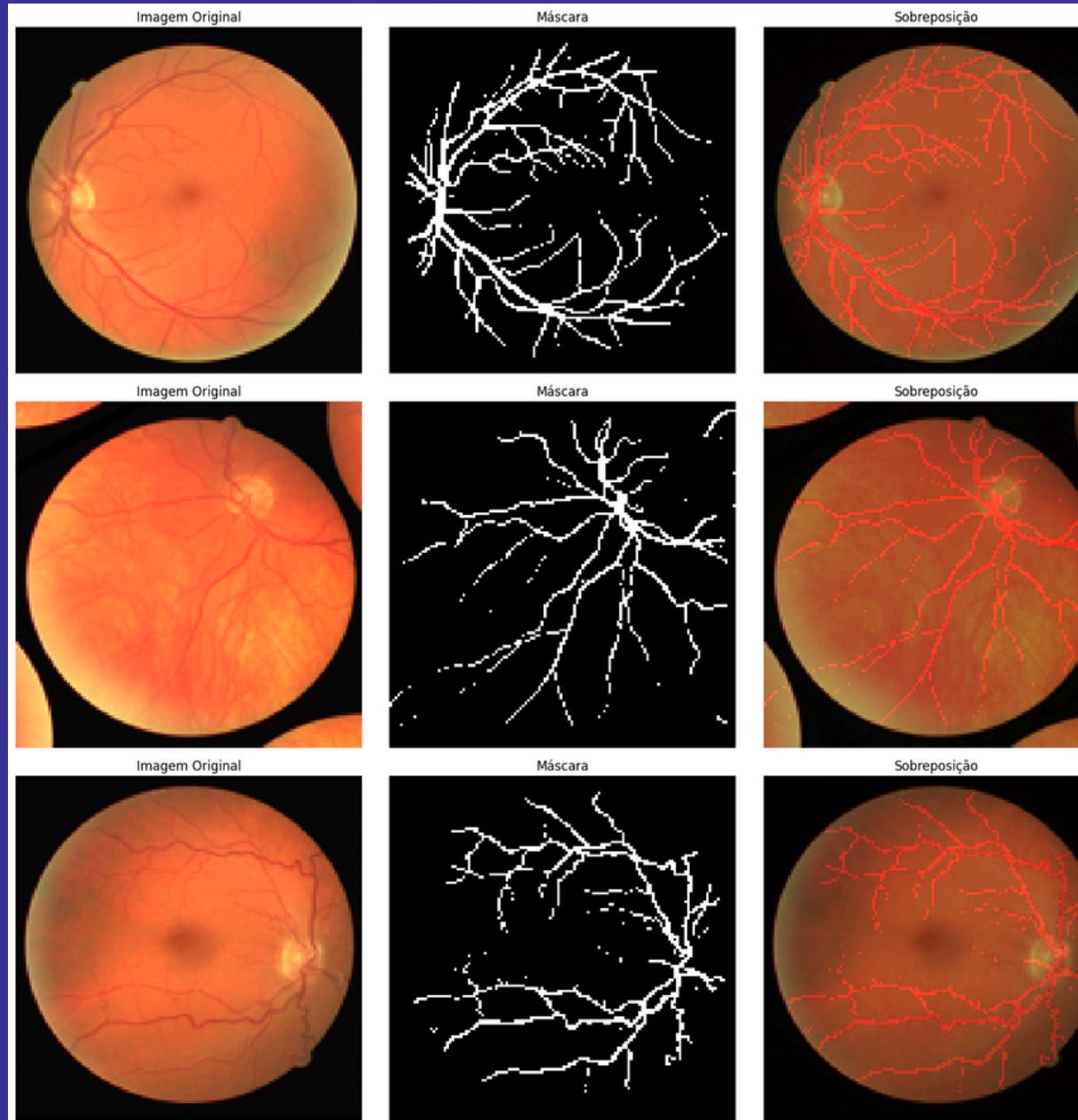
O Batching divide os dados em pequenos blocos, reduz uso de memória e acelera o treinamento;

O Shuffling embaralha os dados a cada época para evitar que o modelo aprenda padrões da ordem dos exemplos;

O Prefetch diminui o tempo de espera e acelera o processo.

Resultados

2) Pré processamento de imagens: Normalização e redimensionamento:



Inicialização:

- Recebe diretórios de imagens e máscaras
- Verifica correspondência entre pares de imagens e máscaras

Carregamento de dados:

- Redimensiona imagens e máscaras para tamanho uniforme
- Normaliza imagens (divide por 255)
- Converte máscaras para formato binário (0 ou 1)
- Gerencia diferentes formatos de imagem (RGB, escala de cinza)

Aumentação de dados:

- Rotação aleatória (± 20 graus, 50% do dataset)
- Espelhamento horizontal (30% do dataset)
- Ajustes de brilho (40% do dataset)
- Zoom aleatório (20% do dataset)

Metodologia

3) Modelagem:

Modelo Sequêncial de Classificação

```
model = Sequential([
    layers.Conv2D(128,5,activation="relu",input_shape=(128,128,3)),
    layers.MaxPool2D(),
    layers.BatchNormalization(),

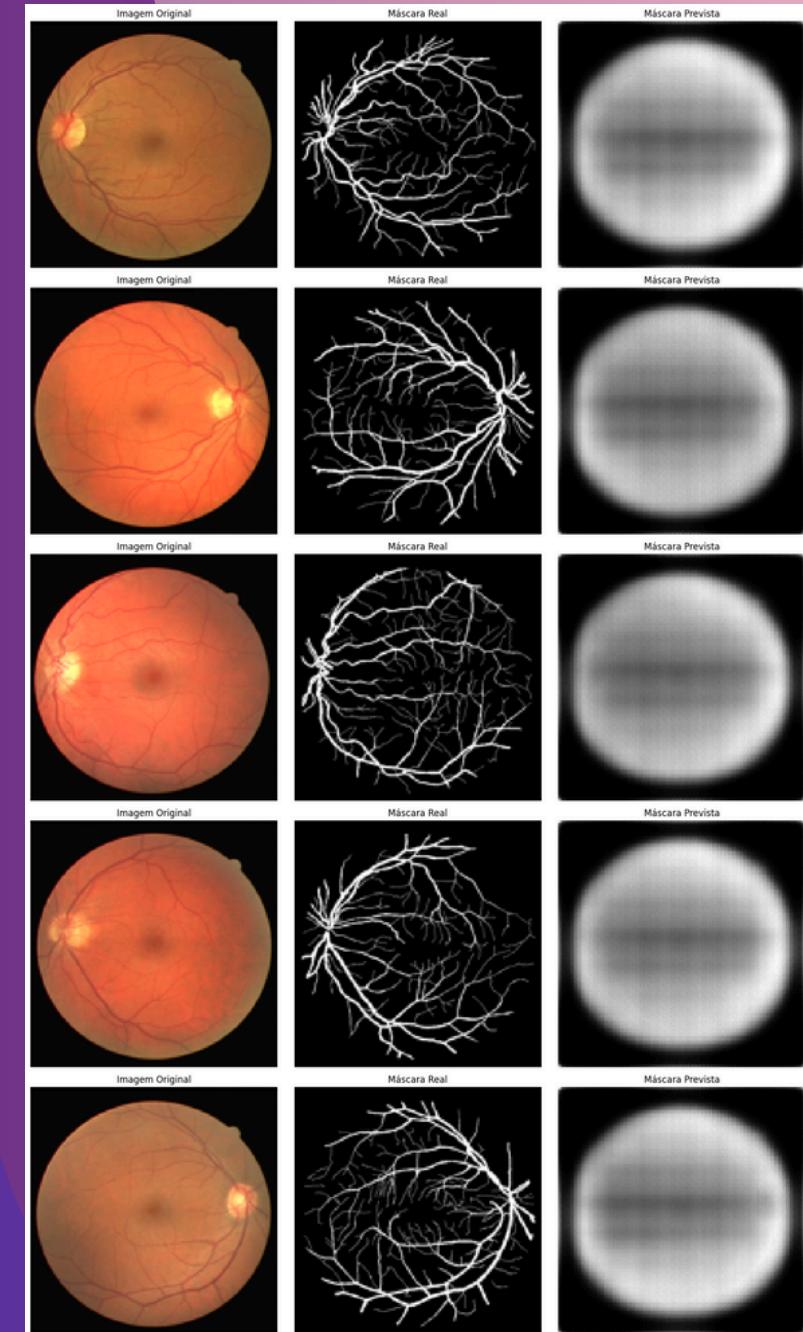
    layers.Conv2D(64,5,activation="relu"),
    layers.MaxPool2D(),
    layers.BatchNormalization(),

    layers.Conv2D(32,5,activation="relu"),
    layers.MaxPool2D(),
    layers.BatchNormalization(),

    layers.Conv2D(16,5,activation="relu"),
    layers.MaxPool2D(),
    layers.BatchNormalization(),

    layers.Flatten(),
    layers.Dense(1,activation="sigmoid")
])
```

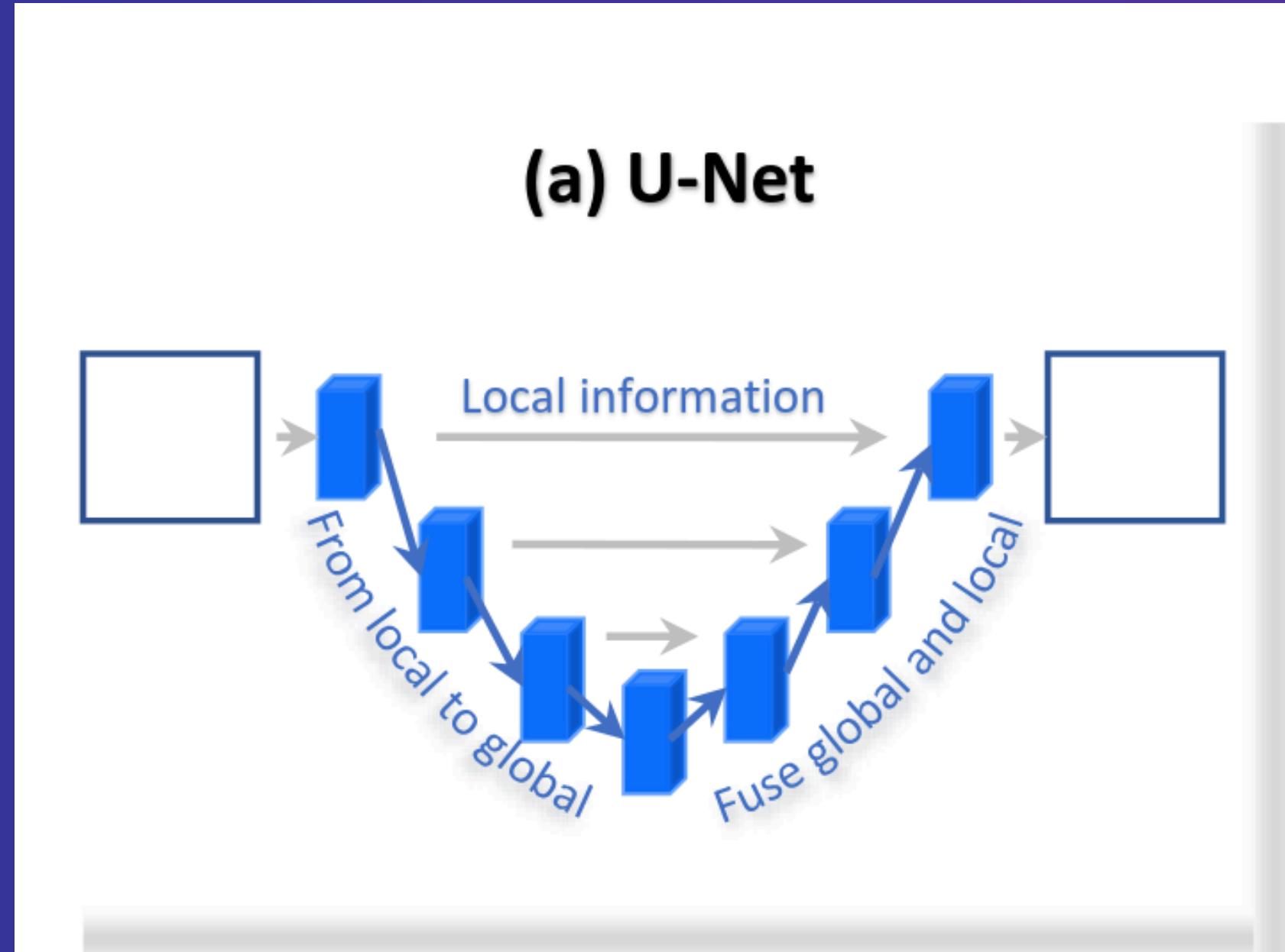
UNet Convencional



Metodologia

3) Modelagem:

Unet MER(Mobile Encoder and Refinement)



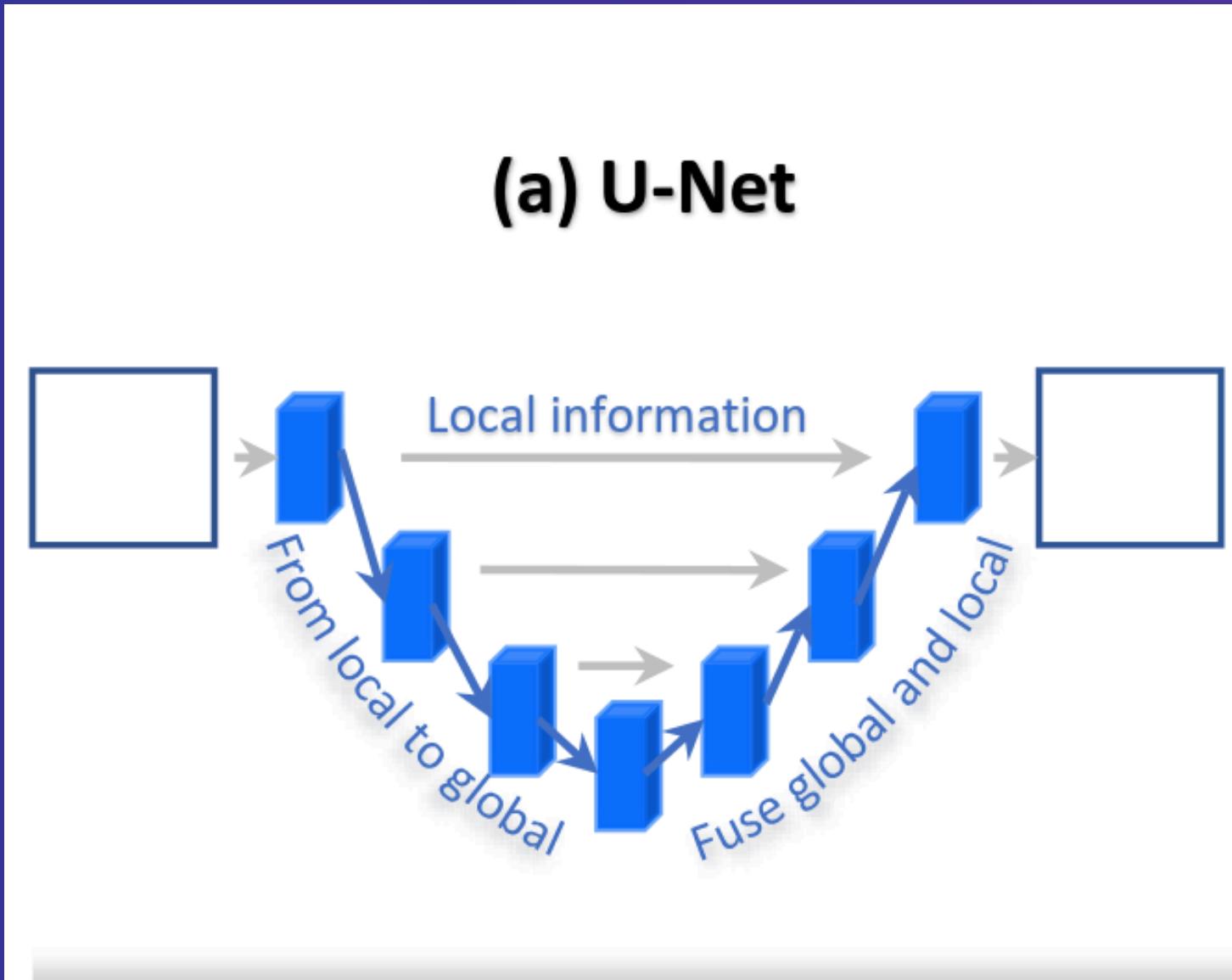
U-Netmer: U-Net meets Transformer for medical image segmentation

Sheng He, Rina Bao, P. Ellen Grant, Yangming Ou

- Base U-Net: Segue o padrão Encoder-Decoder com Skip Connections para segmentação.
- Encoder Eficiente: Usa blocos SeparableConv2D (mais leves) e adiciona spatial_attention nas camadas profundas para focar em áreas importantes.
- Decoder Refinado: Também usa SeparableConv2D e inclui um refinement_module para melhorar a fusão das skip connections e a definição de bordas.
- O objetivo é alcançar boa performance em segmentação com maior eficiência computacional em comparação com uma U-Net padrão.

Metodologia

3) Modelagem:



U-Netmer: U-Net meets Transformer for medical image segmentation

Sheng He, Rina Bao, P. Ellen Grant, Yangming Ou

- Arquitetura U-Net

- Encoder: Características complexas, reduz o tamanho e aumentam o número de canais
- Bridge: Conecta o final do Encoder ao início do Decoder.
- Decoder: Permite a localização precisa dos elementos na imagem
- Saída: 1 classe por ativação sigmoide, comprimindo a saída entre 0 e 1.

- (MER - Mobile Encoder and Refinement):

- Aplica no encoder e decoder o SeparableConv2d que usa menos parâmetros e operações deixando a arquitetura mais leve.
- Refinement Module (`refinement_module`): Aplicado no Decode com o objetivo de refinar as características mais o separableConv2 para refinar as características como bordas.

Metodologia

4) Métricas:

No código do Kaggle:

```
1 es = tf.keras.callbacks.EarlyStopping(  
2     monitor='Accuracy',  
3     min_delta=0,  
4     patience=0,  
5     verbose=0,  
6     mode='auto',  
7     baseline=None,  
8     restore_best_weights=True,  
9     start_from_epoch=10  
10 )  
11
```

Apenas a acurácia, sem gráficos.

Avaliação Quantitativa:

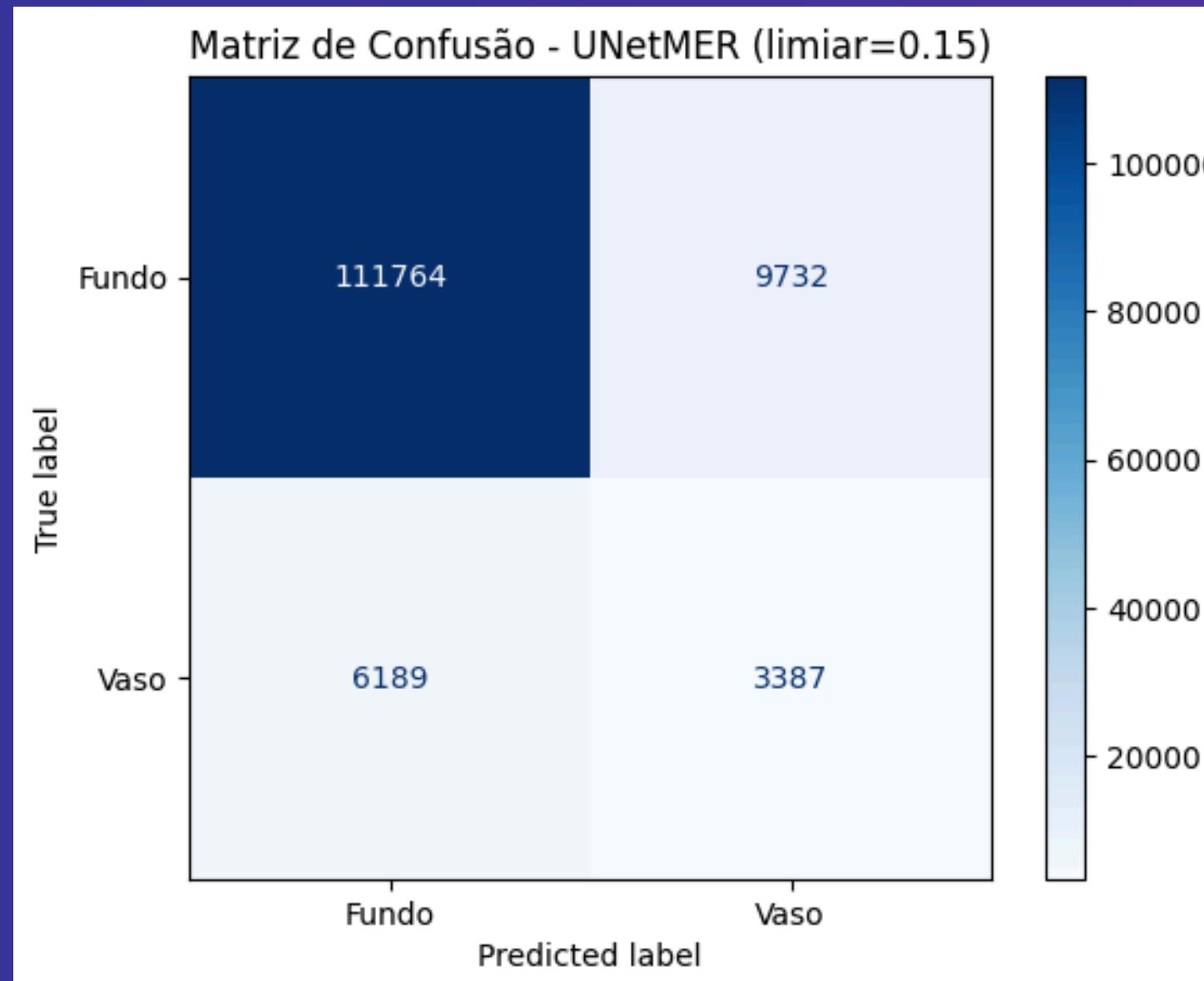
```
# Plotar cada métrica  
for i, (metric, title) in enumerate(zip(metrics_to_plot, titles_to_plot)):  
    if i < len(axes):  
        axes[i].plot(history.history[metric], label=f'Treino')  
        val_metric = f'val_{metric}'  
        # We already checked val_metric exists  
        axes[i].plot(history.history[val_metric], label=f'Validação')  
        axes[i].set_title(title)  
        axes[i].set_xlabel('Época')  
        axes[i].set_ylabel(title)  
        axes[i].legend()  
  
    # Desativar eixos extras não utilizados  
    for i in range(n_metrics, len(axes)):  
        fig.delaxes(axes[i])  
  
plt.tight_layout()  
plt.show()
```

- Dice Coefficient (dice_coef) - mede similaridade entre a segmentação prevista e a real
- Acurácia Binária (binary_accuracy) - porcentagem de pixels classificados corretamente
- Recall/Sensibilidade (recall) - capacidade de detectar vasos sanguíneos
- Precisão (precision) - confiabilidade das detecções positivas

Resultados

4 e 5) Modelagem e Métricas:

Matriz da Confusão:



- **Métricas de avaliação detalhadas:**

- Acurácia: 0.8785
- Precisão (Vaso): 0.2582
- Recall (Sensibilidade Vaso): 0.3537
- F1-Score (Vaso): 0.2985

Desbalanceamento de Classes:

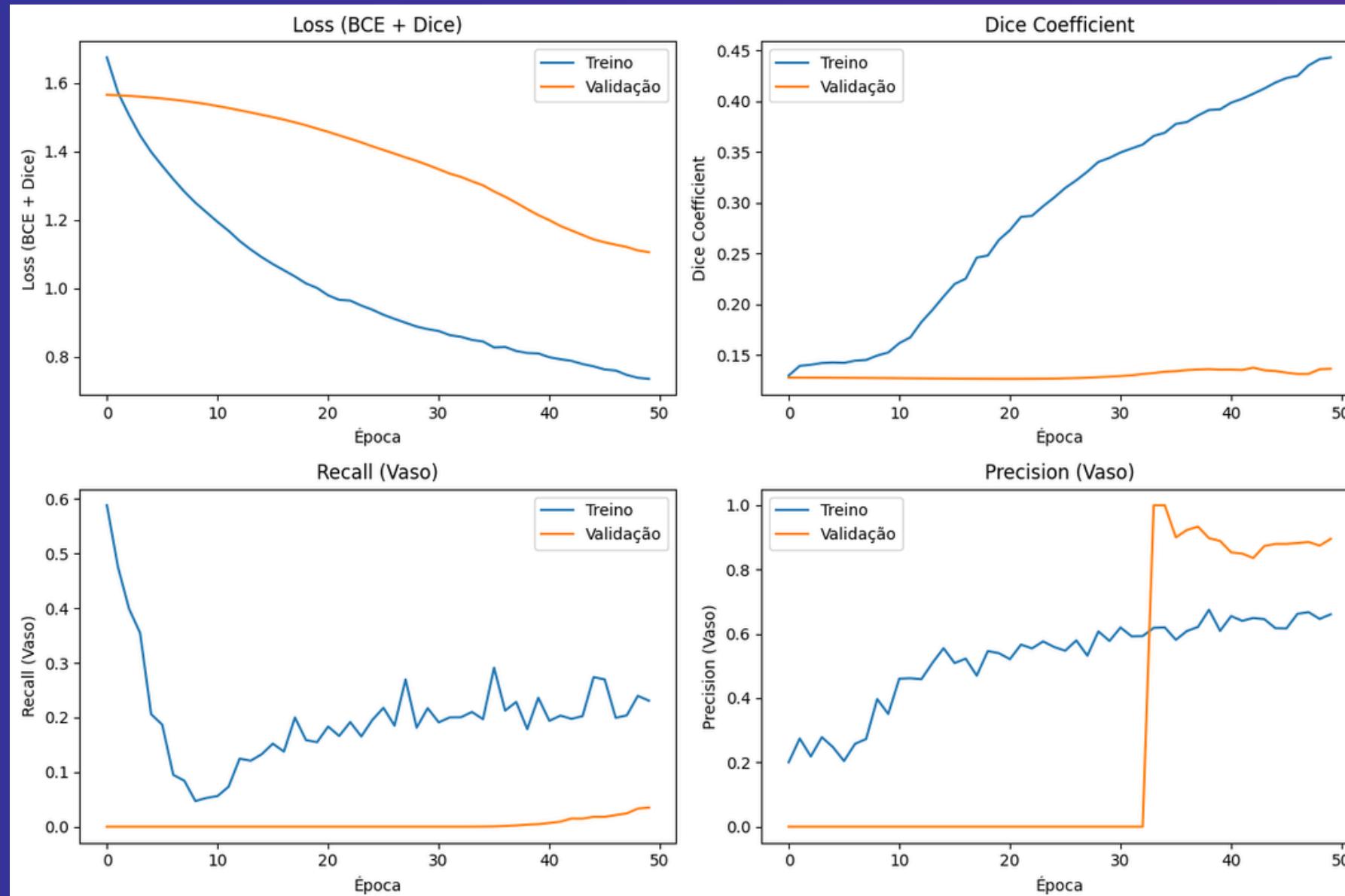
Na segmentação de retina, há naturalmente muito mais pixels de fundo (não-vaso) do que pixels de vaso. Isso pode resultar em:

Alta acurácia mesmo com baixo recall (modelo ignora vasos pequenos)

TN muito maior que as outras células

Resultados

4 e 5) Modelagem e Métricas:

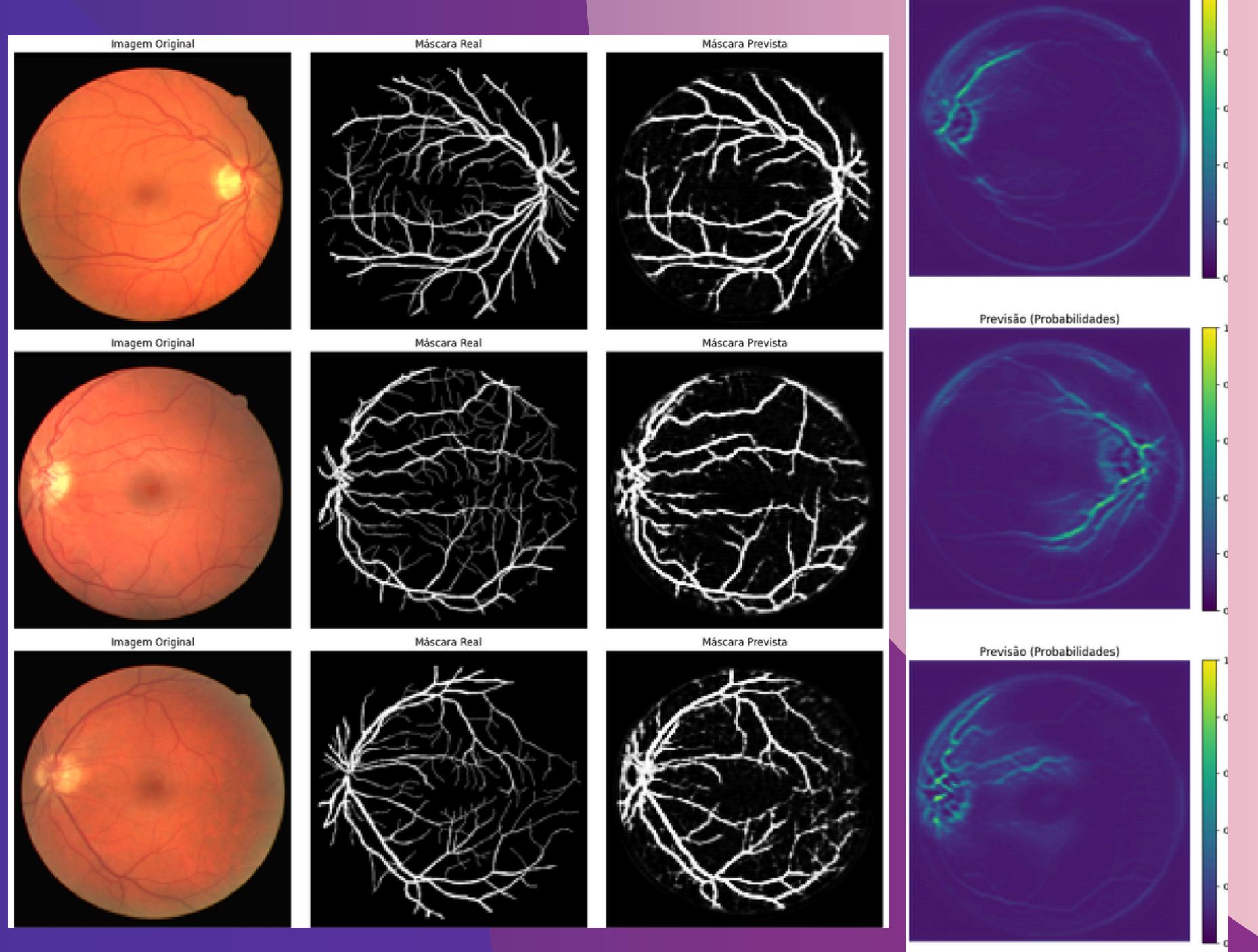


- Gráficos de evolução das métricas ao longo das épocas
- Comparação entre desempenho no conjunto de treino e validação
- Visualização dinâmica baseada nas métricas disponíveis
- 1. Overfitting significativo: Métricas de treino melhoram constantemente enquanto validação permanece estagnada (especialmente o Dice)
- 2. Desbalanceamento de classes: Alta acurácia de validação mas baixo recall sugere que o modelo está classificando a maioria dos pixels como fundo
- 3. Problema de generalização: O modelo aprende padrões nos dados de treino mas não consegue aplicá-los efetivamente nos dados de validação
- 4. Ponto de anomalia: A mudança súbita na precisão de validação na época 30 merece investigação - pode estar relacionada a uma alteração no learning rate, ativação de EarlyStopping ou outro callback

Resultados

4 e 5) Modelagem e Métricas:

- Detecção das estruturas principais;
- Mapa de probabilidades informativo;
- Reconhecimento do disco óptico;
- Falha em vasos finos;
- Inconsistência entre amostras;
- Artefatos circulares.



5) Gráficos e análises de loss (overfitting):

Reaproveitamento do código do Kaggle:

Não tem gráfico no modelo do Kaggle, mas utilizamos o código criado para o nosso modelo para gerar ambos gráficos

```
acc = history.history['Accuracy']
val_acc = history.history['val_Accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.figure(figsize=(12, 5))

# Gráfico de Acurácia
plt.subplot(1, 2, 1)
plt.plot(epochs, acc, 'b-', label='Treinamento')
plt.plot(epochs, val_acc, 'r-', label='Validação')
plt.title('Acurácia por Época')
plt.xlabel('Época')
plt.ylabel('Acurácia')
plt.legend()

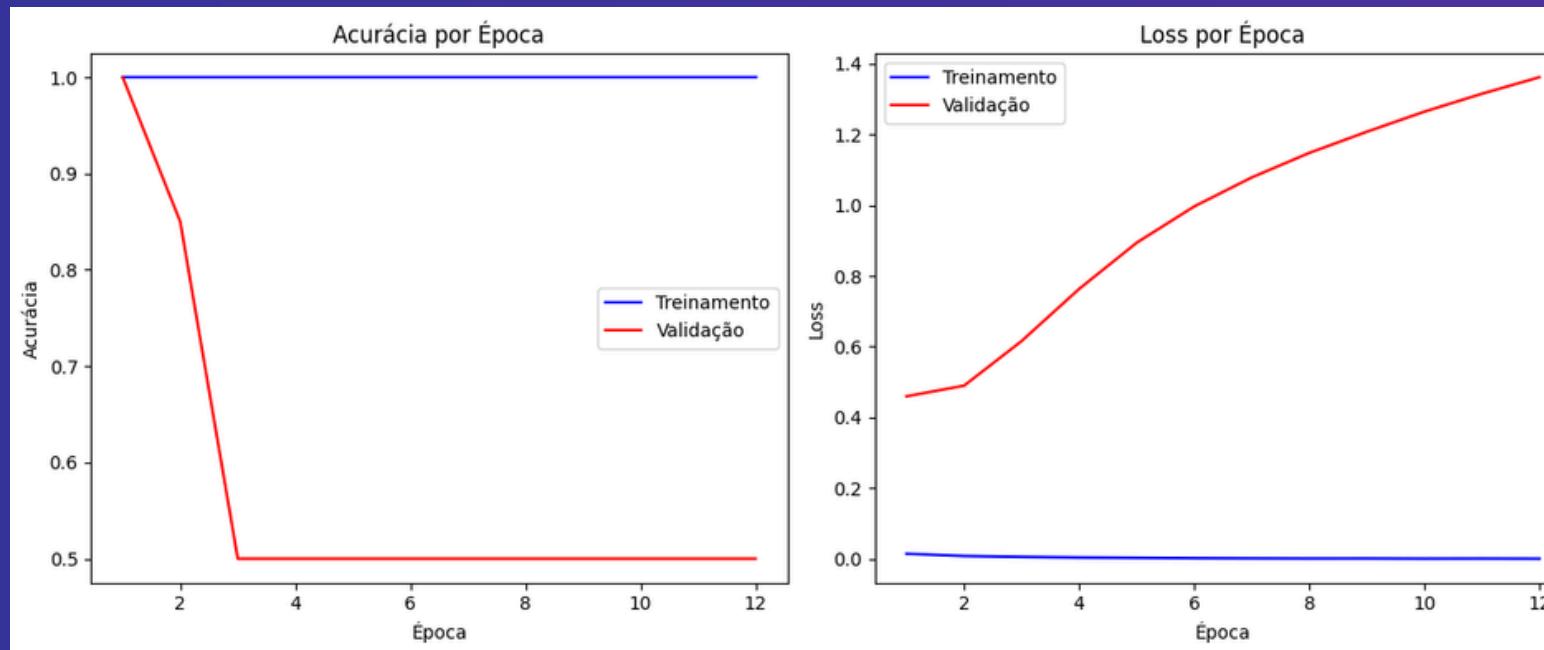
# Gráfico de Loss
plt.subplot(1, 2, 2)
plt.plot(epochs, loss, 'b-', label='Treinamento')
plt.plot(epochs, val_loss, 'r-', label='Validação')
plt.title('Loss por Época')
plt.xlabel('Época')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

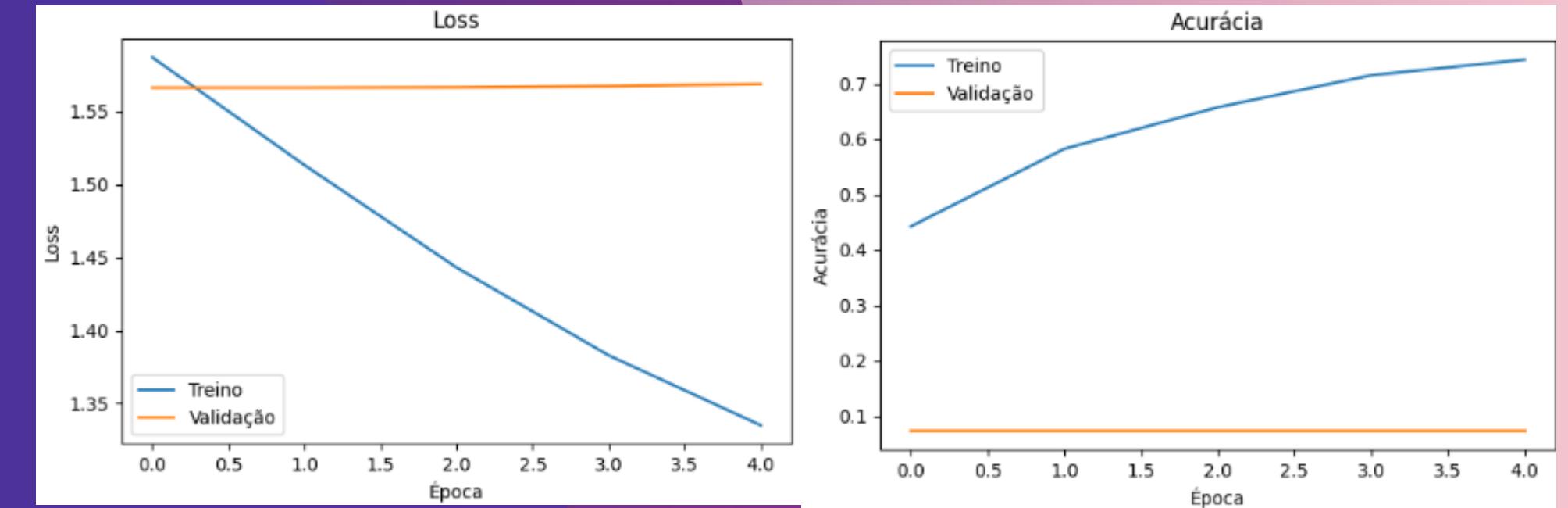
Resultados

5) Gráficos e análises de loss (overfitting):

No código do Kaggle:



No nosso:



O modelo está memorizing (decorando) os dados de treinamento e não está conseguindo generalizar para novos dados (validação). Isso é um claro sinal de overfitting

O modelo está aprendendo nos dados de treino, e a acurácia melhora constante, chegando a ~72%.

Conclusões

- Incluímos etapas de análises exploratórias de dados;
- Melhoramos o overfitting do modelo;
- Trouxemos novas técnicas para métricas importantes;
- Levantamos pontos de melhorias a partir dos resultados dessas novas métricas.

Perspectivas futuras

- Metodologias que diminuam underfitting;
- Balanceamento de classes;
- Testes com menos camadas e filtros menores (3x3);
- Implementação completa do Unetmer (Unet +Transformer);
- Métodos para aumentar a quantidade de imagens (Data Augmentation - outros métodos).

Referências Bibliográficas

SANJEEWANI, N. A. et al. Retinal blood vessel segmentation using a deep learning method based on modified U-NET model. TechRxiv, 2021. Disponível em: <https://www.techrxiv.org/doi/full/10.36227/techrxiv.16653238.v1>. Acesso em: 22 abr. 2025.

LIU, Congjun et al. Multiscale U-Net with Spatial Positional Attention for Retinal Vessel Segmentation. Journal of Healthcare Engineering, v. 2022, Artigo ID 5188362, 2022. Disponível em: <https://onlinelibrary.wiley.com/doi/10.1155/2022/5188362>. Acesso em: 21 abr. 2025

REN, Kan et al. An improved U-net based retinal vessel image segmentation method. Heliyon, v. 8, n. 10, e11187, 2022. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2405844022024756>. Acesso em: 21 abr. 2025.

GUO, Changlu et al. Channel Attention Residual U-Net for Retinal Vessel Segmentation. arXiv preprint arXiv:2004.03702, 2020. Disponível em: <https://arxiv.org/abs/2004.03702>. Acesso em: 24 abr. 2025.