Relatório 28 - Projeto Final

Lucas Scheffer Hundsdorfer

Introdução

Como proposta do projeto final, eu decidi criar um modelo classificatório de visão computacional, meu objetivo além disso é verificar quais casos valem mais a pena utilizar modelos pré treinados e aplicar o transfer learning e quando é mais adequado utilizar um modelo criado. Dentro dessa idéia vou mostrar o acompanhamento do projeto.

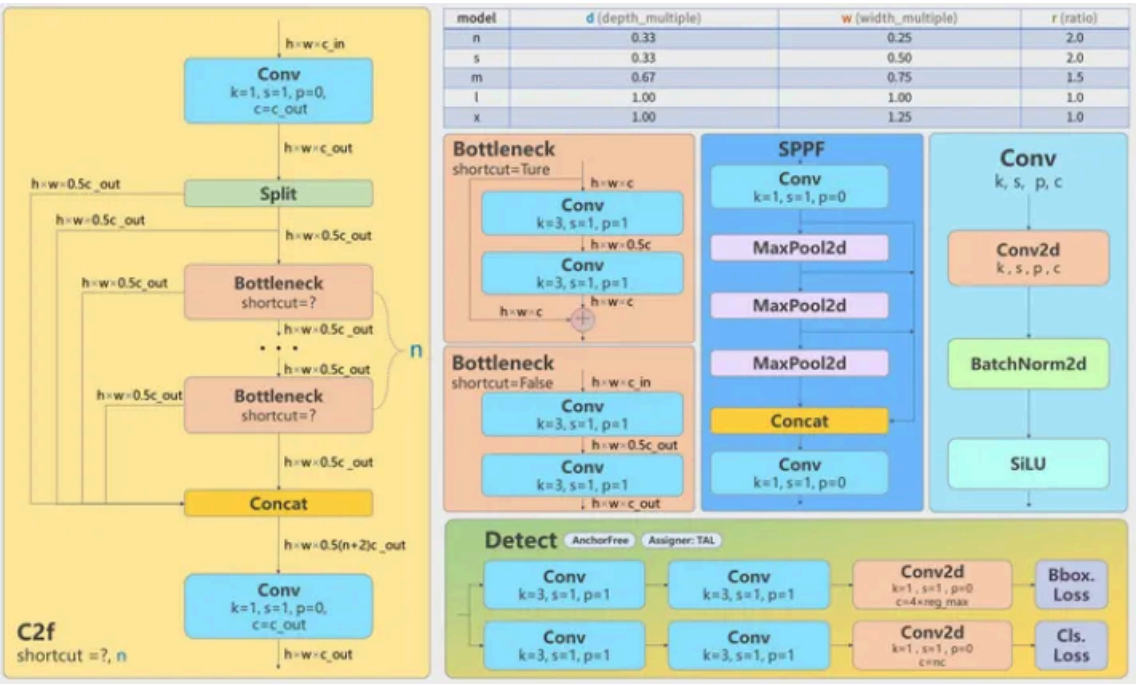
Base de dados

A base de dados utilizada foi uma achada no kaggle:
https://www.kaggle.com/datasets/gunavenkatdoddi/eye-diseases-classification

Modelos

O modelo pré-treinado que eu utilizei para fazer o fine tuning foi o yolo v8, ele é um modelo bem concreto e conhecido. O yolo é útil para vários tipos de tarefas além da classificação, de instância, pose/keypoints, detecção orientada. É oferecido vários tamanhos do modelo, Nano (n), Small (s), Medium (m), Large (l) e Extra-large (x).

Essa é arquitetura do YOLO v8:



Composta por 3 partes, Backbone, Neck e Head. Cada parte tem sua função dentro da arquitetura para poder ser feita toda a leitura e classificação da imagem.

Backbone é a primeira parte da rede neural, é onde a imagem entra e lá é extraído todas as características dela, transformando os pixels em valores numéricos. Utiliza-se de uma arquitetura baseada em CSPDarknet, aprimorada com blocos C2f que aumentam a eficiência e reduzem o custo computacional.

A segunda parte é o Neck, essa parte da estrutura age como um intermediário entre o backbone e o head, onde as informações são agregadas e combinadas, garantindo que objetos pequenos e grandes possam ser detectados com a mesma eficácia. Essa parte é baseada em PAnet (Path Aggregation Network).

A última parte o Head é o responsável pela saída da rede neural, ou seja as predições finais, para cada tipo de modelo, classificação, detecção entre outros. Essa head é uma anchor-free, o que é uma novidade entre os modelos da Ultralytics e traz uma melhora referente a diferentes proporções e orientações dos objetos.

Essa é toda a base da arquitetura do YOLO v8 e é por causa dessa arquitetura que esse modelo se demonstra tão eficiente nas suas tarefas.

Já o modelo que eu criei tem uma arquitetura um tanto mais simples:

```
model = models.Sequential([
        layers.Input(shape=(224, 224, 3)),
        layers.Rescaling(1./255),

        layers.Conv2D(64, (3, 3), padding='same', activation='relu', input_shape=(224,224,3)),
        layers.BatchNormalization(),
        layers.Conv2D(64, (3, 3), padding='same', activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.SpatialDropout2D(0.15),

        layers.Conv2D(128, (3, 3), padding='same', activation='relu'),
        layers.BatchNormalization(),
        layers.Conv2D(128, (3, 3), padding='same', activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.SpatialDropout2D(0.20),

        layers.Conv2D(256, (3, 3), padding='same', activation='relu'),
        layers.BatchNormalization(),
        layers.Conv2D(256, (3, 3), padding='same', activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.SpatialDropout2D(0.25),

        layers.Conv2D(256, (3, 3), padding='same', activation='relu'),
        layers.BatchNormalization(),
        layers.Conv2D(256, (3, 3), padding='same', activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.SpatialDropout2D(0.30),


        layers.GlobalAveragePooling2D(),
        layers.Dense(512, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.4),
        layers.Dense(128, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Dense(4, activation='softmax')
])
```

Também é uma rede neural convolucional, e é composta por 4 blocos de convolução seguidos e uma última camada densa, seguindo uma lógica de extração progressiva de características.

Dentro dos blocos de convolução temos camadas de convolução, batch normalization, max pooling e o dropout, as camadas convolucionais ficam responsáveis por identificar os padrões e aplicar filtros que geram os mapas de de ativação correspondentes às características, todas as camadas estão utilizando a função de ativação ReLu que introduz uma não linearidade e acelera o treinamento. Já os batches normalizations servem para estabilizar o processo todo, ajustando a média e a variância das saídas anteriores, por isso sempre tem uma após a convolução. As camadas de max pooling servem para reduzir a dimensionalidade dos mapas de ativação que a camada convolucional geral, diminuindo o custo computacional. E os spatial dropouts são dropouts especializados para redes neurais convolucionais, pois invés de desativar apenas neurônios eles desativam mapas inteiros de ativação, evitando um possível overfitting. E a arquitetura demonstra que a cada bloco convolucional o número de filtros gerados cresce de 64 até 256, isso foi feito de maneira proposital para as primeiras camadas obterem as características mais simples e as mais profundas camadas obterem as características mais complexas e abstratas.

A camada densa vem após toda a extração das características das imagens, e é onde se faz a classificação, onde cada imagem é associada a uma probabilidade de cada classe, esse bloco também tem dropout e batch normalization pelos mesmos motivos.

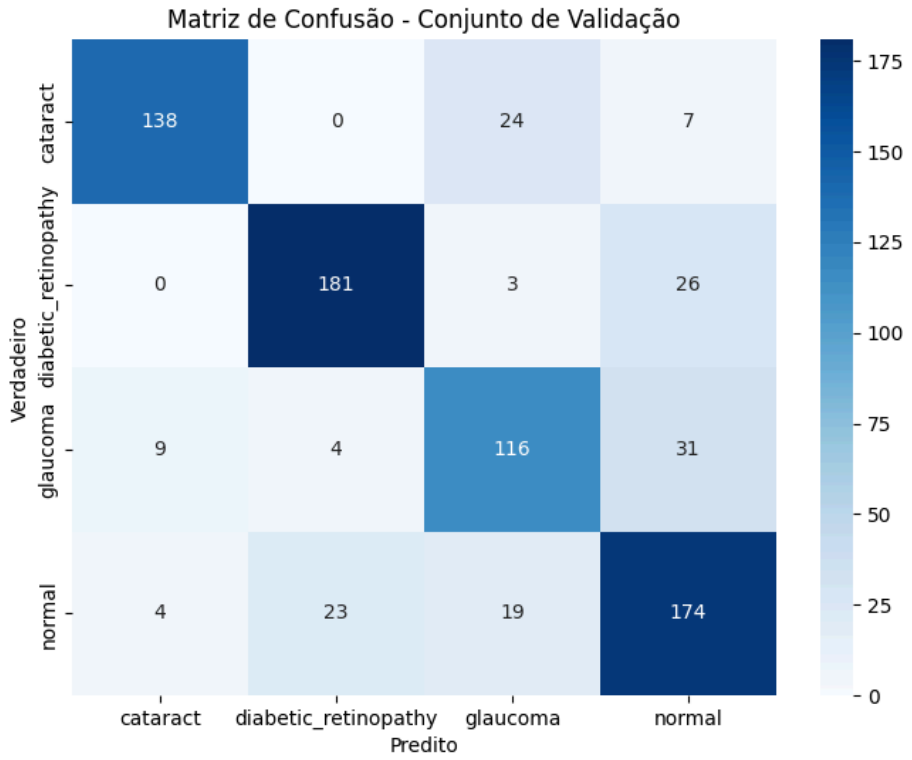
Métodos utilizados

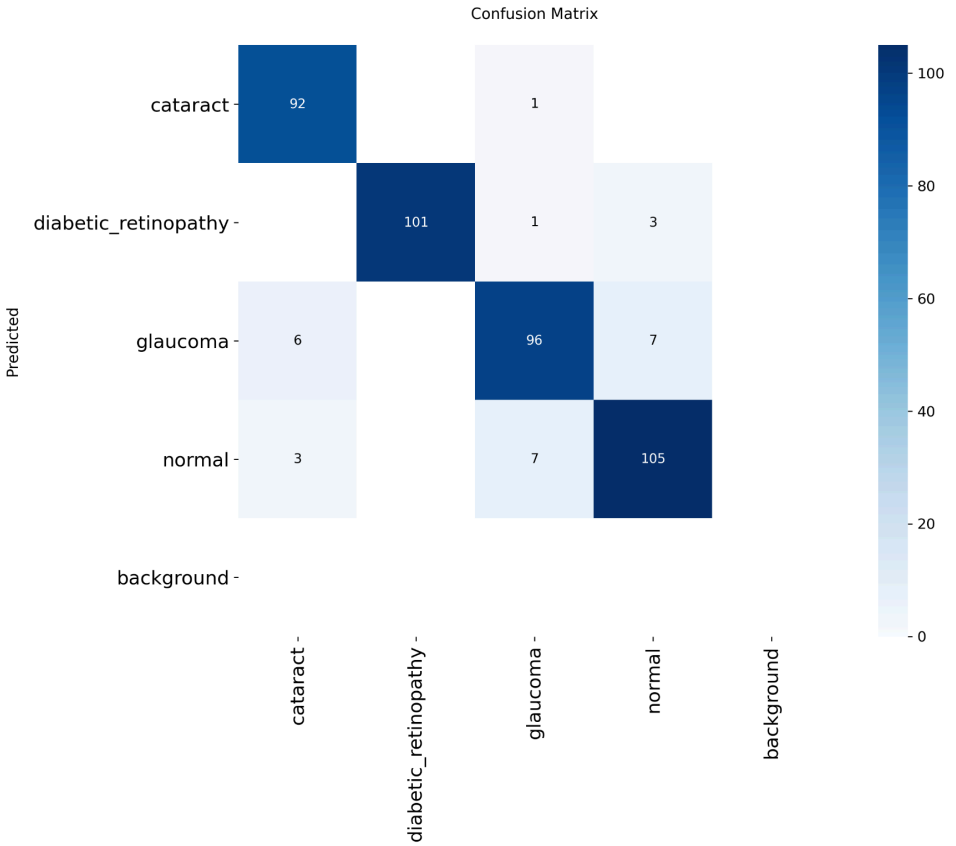
Métodos utilizados dentro do modelo criado, o pré-processamento dos dados foi a normalização dos pixels para ficar entre 0 e 1 e a base de dados foi dividida em 80% dos dados para treino e os outros 20% para validação. Foi definido o tamanho da imagem como 224x224 pixels, e o batch size de 32. Foi utilizado do data augmentation tanto para reduzir as chances de overfitting e tanto para generalização do modelo, foi aplicadas transformações leves por se tratar de um dataset clínico e não era o ideal distorções muito grandes, também é ideal ter o mínimo de data augmentation pois imagens clínicas não são tão fáceis de se obter e é feito esse processo para gerar um aumento na quantidade de dados. E por fim o treinamento foi feito com uma quantidade total de 80 épocas, utilizando o otimizador AdamW com agendamento do learning rate para não ficar preso e também utilizando o Early Stopping para evitar treinos demais sem a melhora do modelo.

Primeiramente o modelo escolhido foi yolov8n-cls, versão nano do modelo com foco em classificação. De pré-processamento foi feito o redimensionamento para 224x224 que nem no modelo próprio, as imagens foram convertidas para RGB e os pixels normalizados para valores entre 0 e 1, e um batch size de 16, com uma separação de 90% da base de dados para treino e 10% para testes. Foi aplicado o data augmentation também de uma forma leve apenas para evitar o overfitting e ganhar uma generalização. Foram executadas 100 épocas, foi utilizado o AMP(automatic mixed precision) que combina operações entre float16 e float32 economizando tempo e custo computacional, não foi utilizado dropout nesse modelo, e para controlar a subida e descida do learning rate o warmup epochs foi setado como 3.

Resultados

Após os treinamentos chegamos nessas duas matrizes de confusão:





A primeira matriz do modelo criado e a segunda do modelo pré-treinado. Claramente o YOLO se saiu melhor, agora só vou detalhar os possíveis motivos e avaliar os resultados de ambos.

Essa diferença das matrizes de confusão se dá devido ao fato que o YOLO conseguiu atingir 93% de acurácia total,se demonstrado eficiente para a identificação de todas as classes, errando apenas 18 vezes de mais de 400 imagens presentes no dataset de validação. E o erro mais comum foi entre glaucoma e normal, que são realmente mais parecidos e com poucas diferenças visuais, catarata e retinopatia diabética que são mais perceptíveis são as que têm menos erros. Apresentou um loss baixo tanto de treino e de validação, 0,11 no loss de treino e 0,25 no loss de validação. Basicamente um modelo confiável que trouxe bons resultados.

Agora o modelo criado, trouxe uma acurácia total um pouco menor de 80%, com alguns erros a mais para classificar como glaucoma, pois não é tão evidente, e com uma certa facilidade a mais com as doenças fáceis de serem classificadas. Outras métricas importantes, o recall que é quanto que o modelo conseguiu predizer corretamente, ficou em 0,7986. A precisão que marca quantos das imagens que ele classificou como positivas ele acertou de fato, ficou em 0,8078. E o f1-score é a média entre essas duas estatísticas e ficou em 0,8022. Esses dados são importantes porque como se trata de um dataset clínico a precisão não é tão importante como o recall, já que é melhor alertar falsamente (um falso positivo) do que deixar passar um caso real (um falso negativo). O modelo customizado trouxe resultados sólidos, com algumas melhoras poderia se tornar melhor que o modelo customizado mas provavelmente iria trazer um custo computacional bem maior.

Conclusões

Minhas conclusões com este trabalho permitiu compreender um pouco mais sobre a diferença entre modelos pré-treinado e modelos desenvolvidos do zero, mostrando que ambas abordagens têm papéis complementares e que são dependentes do objetivo e recursos do projeto.

Para uma base de dados menor como a que foi utilizada o transfer learning se demonstra muito eficiente como foi visto, uma acurácia total de 93% porém como desvantagem tem uma menor interpretabilidade comparada ao modelo desenvolvido e um menor entendimento e controle da arquitetura.

Já o modelo customizado se mostra totalmente interpretável já que é você quem cria cada camada da rede neural tendo controle sobre a arquitetura toda, porém tem um treino mais pesado e lento e um menor desempenho total comparado ao YOLO.

Modelos pré-treinados são recomendados para uso quando você precisa economizar tempo, dados e recursos computacionais, pois eles já foram treinados em grandes conjuntos de dados e aprendem padrões gerais que podem ser adaptados para novas tarefas por fine-tuning. Eles são indicados especialmente quando você tem um conjunto de dados pequeno ou limitado.

Modelos customizados, por outro lado, são recomendados quando o objetivo é máxima personalização e acurácia para um domínio específico. Eles podem ser justificados se o seu problema requer aprendizado de padrões muito particulares ou conteúdos que não estavam presentes no treinamento original do modelo pré-treinado. No entanto, treinamentos customizados demandam mais dados, tempo, recursos computacionais e expertise para ajustar hiperparâmetros e evitar overfitting.