Roteiro de Projeto: Classificação de Imagens com CNN (3 Classes)

10 de junho de 2025

Tema 1: Classificação com CNN

Este roteiro detalha as fases e ações necessárias para um projeto de classificação de imagens utilizando Redes Neurais Convolucionais (CNNs), com foco em 3 classes distintas e no tratamento de dados.

Fase 1: Preparação e Organização dos Dados

1. **Revisão e Confirmação da Estrutura do Dataset:**
   - *Objetivo:* Entender a organização das imagens e metadados no dataset `Coronahack Chest X-Ray Dataset`.
   - *Ações:*
     - Confirmar o caminho para a pasta raiz das imagens originais (aquela com o duplo `Coronahack-Chest-XRay-Dataset` aninhado).
     - Localizar o arquivo `Chest_xray_Corona_Metadata.csv`.
     - Verificar a estrutura de subpastas dentro de `test` e `train` (ex: `NORMAL`, `PNEUMONIA/COVID`, `PNEUMONIA/VIRUS`, etc.).
     - Identificar as colunas no CSV que contêm o nome do arquivo da imagem (`X_ray_image_name`), o tipo de dataset (`Dataset_type` - TRAIN/TEST) e os rótulos de diagnóstico (`Label`, `Label_1_Virus_category`, `Label_2_Virus_category`).
   - *Resultado:* Conhecimento claro da hierarquia de arquivos e das informações de rótulos.

2. **Definição das 3 Classes Alvo para Classificação:**
   - *Objetivo:* Simplificar o problema de classificação para 3 classes relevantes do dataset.
   - *Ações:*

- Analisar os rótulos disponíveis no `Chest_xray_Corona_Metadata.csv`.
- Propor 3 classes distintas e balanceadas, se possível. Ex:
  - Classe 1: **Normal**
  - Classe 2: **COVID-19**
  - Classe 3: **Outras Pneumonias** (combinando Viral, Bacterial, ARDS, Strep, ou deixando como uma única "Pneumonia Não-COVID").
- Documentar as regras para mapear os rótulos originais para estas 3 classes.

- *Resultado:* Definição clara das 3 classes e estratégia de mapeamento de rótulos.

3. Redimensionamento e Organização das Imagens (para Treino, Validação e Teste):

- *Objetivo:* Padronizar o tamanho das imagens para a entrada da CNN e organizar em pastas de saída claras.
- *Ações:*
  - Definir um tamanho padrão para as imagens (ex: 224x224, 256x256).
  - Criar uma **pasta raiz para o dataset redimensionado** (ex: `coronahack_processed`).
  - Dentro dela, criar subpastas para `train`, `validation` e `test`.
  - **Script de Redimensionamento:** Desenvolver ou adaptar o script Python para:
    - Percorrer as imagens originais da pasta `train` e `test`.
    - Redimensionar cada imagem para o tamanho padrão.
    - Salvar as imagens redimensionadas nas respectivas subpastas de **treino, validação e teste** dentro de `coronahack_processed`, mantendo a estrutura de subpastas por classe (ex: coronahack_processed/train/Normal, `coronahack_processed/train/COVID-19`).
    - **Crucial:** Decidir a estratégia de divisão do conjunto de validação (ex: 80% do `train` original para `train_processed` e 20% para `validation_processed`). O conjunto `test` original será seu `test_processed`.
- *Resultado:* Todas as imagens processadas e organizadas em pastas `train`, `validation`, `test` com o tamanho padrão.

4. Criação do DataFrame de Metadados Unificado:

- *Objetivo:* Consolidar informações de imagens redimensionadas, rótulos e caminhos em um único DataFrame Pandas.
- *Ações:*

- Carregar o `Chest_xray_Corona_Metadata.csv` original.
- **Filtrar o DataFrame para incluir apenas as amostras pertinentes** às 3 classes selecionadas.
- Criar uma nova coluna (`final_label`) que contenha as 3 classes definidas.
- **Codificar a coluna `final_label`** para valores numéricos (0, 1, 2) e armazenar o mapeamento (ex: {0: 'Normal', 1: 'COVID-19', 2: 'Outras Pneumonias'}).
- **Criar uma nova coluna (`image_path_resized`)** que aponte para o caminho completo da imagem redimensionada correspondente em suas novas pastas (`coronahack_processed/train`, `coronahack_processed/validation`, `coronahack_processed/test`).
- **Dividir o DataFrame em 3 sub-DataFrames**: `df_train`, `df_validation`, `df_test`, com base nas imagens que foram salvas em cada respectiva pasta redimensionada.
- Verificar o balanço de classes em cada sub-DataFrame.

- *Resultado:* Três DataFrames (`df_train`, `df_validation`, `df_test`) limpos, com caminhos de imagens redimensionadas e rótulos codificados.

Fase 2: Pré-processamento e Aumentação de Dados

0. Definição das Transformações de Pré-processamento:

- *Objetivo:* Preparar os dados para a entrada da CNN.
- *Ações:*
  - **Normalização:** Converter pixels para o intervalo $[0, 1]$ e aplicar normalização com média e desvio padrão (usar valores do ImageNet ou calcular do próprio dataset).
  - **Conversão para Tensor:** Transformar imagens em tensores (PyTorch/TensorFlow).
  - **Aumentação de Dados (Data Augmentation) para Treinamento:**
    - *Objetivo:* Aumentar a diversidade do conjunto de treinamento e reduzir overfitting.
    - *Ações:* Aplicar transformações aleatórias como rotação, espelhamento horizontal/vertical, zoom, brilho/contraste, etc. (Escolher as mais relevantes para imagens de raio-X).
  - **Importante:** As transformações para **validação e teste** devem ser apenas a normalização e conversão para tensor (sem data augmentation).
- *Ferramentas:* `torchvision.transforms` (PyTorch) ou `tf.keras.Sequential` com camadas de pré-processamento (TensorFlow).

- *Resultado:* Funções de transformação definidas para treino, validação e teste.

0. Criação de DataLoaders:

- *Objetivo:* Carregar os dados em batches de forma eficiente para o treinamento da CNN.
- *Ações:*
  - **Custom Dataset Class:** Implementar uma classe de Dataset personalizada que:
    - Receba um dos DataFrames (`df_train`, `df_validation`, `df_test`).
    - No método `__getitem__`, carregue a imagem do `image_path_resized`, aplique as transformações e retorne a imagem (como tensor) e o rótulo codificado.
  - **DataLoaders:** Criar instâncias de DataLoader para treino, validação e teste.
    - Definir o tamanho do batch (`batch_size`).
    - Definir `shuffle=True` para o DataLoader de treino.
    - Definir `num_workers` para carregamento paralelo (se a máquina tiver muitos núcleos).
- *Ferramentas:* `torch.utils.data.Dataset`, `torch.utils.data.DataLoader` (PyTorch).
- *Resultado:* DataLoaders prontos para alimentar a CNN.

Fase 3: Treinamento e Avaliação da CNN

0. Definição da Arquitetura da CNN:

- *Objetivo:* Escolher ou construir um modelo de CNN adequado para a tarefa.
- *Ações:*
  - **Transferência de Aprendizagem (Recomendado):** Utilizar um modelo pré-treinado em ImageNet (ex: ResNet, VGG, MobileNet) e adaptar a última camada de classificação para as 3 classes.
  - **Modelo do Zero (Opcional):** Construir uma CNN simples do zero (mais complexo e geralmente menos performático para datasets menores).
  - Definir a arquitetura (camadas, filtros, ativações).
- *Ferramentas:* `torchvision.models` (PyTorch), `tf.keras.applications` (TensorFlow).
- *Resultado:* Modelo de CNN instanciado e pronto para o treinamento.

0. Configuração do Treinamento:

- *Objetivo:* Preparar o ambiente para o treinamento do modelo.
- *Ações:*
  - **Função de Perda (Loss Function):** Escolher uma função de perda apropriada para classificação multiclasse (ex: `CrossEntropyLoss` para PyTorch, `SparseCategoricalCrossentropy` ou `CategoricalCrossentropy` para TensorFlow).
  - **Otimizador:** Selecionar um otimizador (ex: Adam, SGD) e definir a taxa de aprendizado (`learning_rate`).
  - **Monitoramento:** Definir métricas a serem monitoradas durante o treinamento (precisão/accuracy, loss).
  - **Dispositivo:** Configurar para usar GPU se disponível (`CUDA` no PyTorch, `GPU` no TensorFlow).
- *Resultado:* Otimizador, função de perda e métricas definidos.

0. Ciclo de Treinamento e Validação:

- *Objetivo:* Treinar o modelo e monitorar seu desempenho.
- *Ações:*
  - Loop de `epochs`.
  - **Fase de Treinamento por Época:**
    - Iterar sobre o DataLoader de treino.
    - Passar os dados pela CNN (forward pass).
    - Calcular a perda.
    - Calcular gradientes (backward pass).
    - Atualizar pesos do modelo (optimizer step).
    - Acumular métricas (loss e accuracy).
  - **Fase de Validação por Época:**
    - Iterar sobre o DataLoader de validação (sem calcular gradientes).
    - Calcular a perda e as métricas.
    - Salvar o melhor modelo com base na métrica de validação (ex: menor loss de validação ou maior accuracy de validação).
  - **Logging:** Registrar métricas (loss, accuracy) para treino e validação a cada época.
- *Resultado:* Modelo treinado e salvo (checkpoint do melhor desempenho).

0. Avaliação Final no Conjunto de Teste:

- *Objetivo:* Avaliar o desempenho generalizado do modelo em dados nunca vistos.

- *Ações:*
  - Carregar o melhor modelo salvo.
  - Executar o modelo no DataLoader de teste (sem calcular gradientes).
  - Calcular métricas de avaliação:
    - **Precisão (Accuracy)**: Geral e por classe.
    - **Matriz de Confusão:** Visualizar acertos e erros por classe.
    - **Recall, Precision, F1-Score:** Métricas mais detalhadas para classes desbalanceadas.
    - **Curva ROC e AUC** (se aplicável para classificação binária ou multiclasse com extensões).
  - Analisar **overfitting** (se o desempenho de treino é muito bom e o de validação/teste é muito pior).
- *Ferramentas:* `sklearn.metrics` (para matriz de confusão, relatórios de classificação).
- *Resultado:* Relatório completo de avaliação do modelo, com métricas e gráficos.