Pneumonia Detection from Chest X-Rays

Haqeeq Ruman

Problem definition

The objective is to develop a deep learning model to accurately detect pneumonia from chest X-ray images, addressing the challenge of class imbalance in the dataset. The model aims to provide interpretable visualizations using Grad-CAM to highlight critical lung regions, enhancing diagnostic reliability for radiologists.

Solution Explanation

Loading Data

First, I started by setting up the tools needed to build and train the models for detecting pneumonia from chest X-ray images. I used Python libraries like PyTorch for creating neural networks, torchvision for handling images, and matplotlib for plotting results. I also used kagglehub to download the dataset and scikit-learn to measure how well the models perform. Next, I checked if a GPU was available to speed up computations, but since I'm using a CPU, I set the device to CPU and printed a message to confirm this. Then, I needed to get the chest X-ray dataset, so I tried downloading it automatically from Kaggle using kagglehub.

Preprocessing Data

After securing the dataset, I defined how to prepare the X-ray images for the models. I set the image size to 224x224 pixels, a common size for neural networks, and chose a batch size of 32 images to process at once, balancing speed and memory usage. For training images, I applied transformations to make the model robust: I resized them, rotated them slightly, flipped them horizontally, shifted them a bit, and cropped them randomly to focus on different parts. I converted the images into a numerical format (tensors) and adjusted their pixel values to match what pre-trained models expect. For validation and test images, I only resized, converted, and normalized them without random changes to keep them consistent. Then, I loaded the images into datasets using these transformations, with separate sets for training (about 5216 images), validation (16 images), and testing (624 images). I created data loaders to feed these images to the models in batches, shuffling the training data to mix it up but keeping validation and test data in order. I printed the class labels (NORMAL as 0, PNEUMONIA as 1) and the number of images in each set to confirm everything was loaded correctly.

Since there are more PNEUMONIA images (3875) than NORMAL ones (1341), I calculated weights to balance them. I gave NORMAL images a higher weight (about 2.89) and PNEUMONIA a weight of 1, so the model pays more attention to NORMAL cases during training. I stored these weights in a format the model could use on the CPU.

CustomCNN

Next, I built the first model, called CustomCNN. I started by creating a class for it, inheriting from PyTorch's neural network module. Inside, I defined two main parts: a feature extractor and a classifier. The feature extractor has three convolutional layers. The first layer takes the 3 color channels of the image and creates 32 feature maps, scanning with a 3x3 window and adding padding to keep the size. I applied a ReLU function to highlight important features and reduced the image size to 112x112 with max-pooling. The second layer takes these 32 maps, creates 64, applies ReLU, and pools to 56x56. The third layer makes 128 maps, applies ReLU, and pools to 28x28. The classifier flattens these maps into a long vector (100352 numbers), connects to 256 units with ReLU, drops half randomly to prevent overfitting, and connects to one output unit with a sigmoid function to give a probability (0 for NORMAL, 1 for PNEUMONIA). When the model runs, it passes the image through the feature extractor, then the classifier, to predict the probability.

To train any model, I created a training function that runs for several epochs, each time going through all training images and checking performance on validation images. I kept track of loss (how wrong the predictions are) and accuracy (how many are correct) for both phases. For each epoch, I set the model to training mode, processed batches of images, and calculated a weighted loss using the class weights to handle imbalance. I updated the model's weights to reduce this loss, counted correct predictions, and averaged the loss and accuracy. In validation mode, I did the same without updating weights, just checking performance. If validation accuracy improved, I saved the model's weights. I printed the metrics for each epoch to monitor progress and returned the history and saved model path. For evaluation, I wrote a function to test the model on the test set, predicting labels (0 or 1) for each image, and calculating accuracy, precision, recall, and F1-score to measure how well it identifies PNEUMONIA. I also made a function to plot training and validation loss and accuracy curves, creating side-by-side graphs to visualize how the model improves over epochs.

Now, I trained the CustomCNN. I created the model, moved it to the CPU, used a binary cross-entropy loss function that computes loss per image, and chose the Adam optimizer with a learning rate of 0.001 to update all weights. I trained it for 10 epochs, saving the best model as CustomCNN_best.pth, and evaluated it on the test set, printing metrics like accuracy (previously around 81.57%).

ResNet18

Next, I worked on the second model, ResNet18, using transfer learning. I loaded a pre-trained ResNet18, trained on millions of general images, which has many layers to detect complex patterns. I replaced its final layer to output one probability with a sigmoid, matching our task. I moved it to the CPU, froze all layers except the new final layer to keep the pre-trained knowledge, and trained only the final layer for 5 epochs using the same loss and optimizer, saving as ResNet18_best.pth. Then, I fine-tuned ResNet18 by unfreezing its last block (layer4) to adapt deeper features to X-rays. I used a smaller learning rate (0.0001) for layer4 and 0.001 for the final layer, trained for 5 more epochs, and saved as ResNet18FineTuned_best.pth. I evaluated this model, expecting better accuracy (around 93.27%). Finally, I plotted curves for CustomCNN, ResNet18, and the fine-tuned ResNet18, and printed a comparison of their accuracy and F1-scores to show ResNet18's superiority due to its pre-trained knowledge and deeper structure.

Results & evaluation

CustomCNN

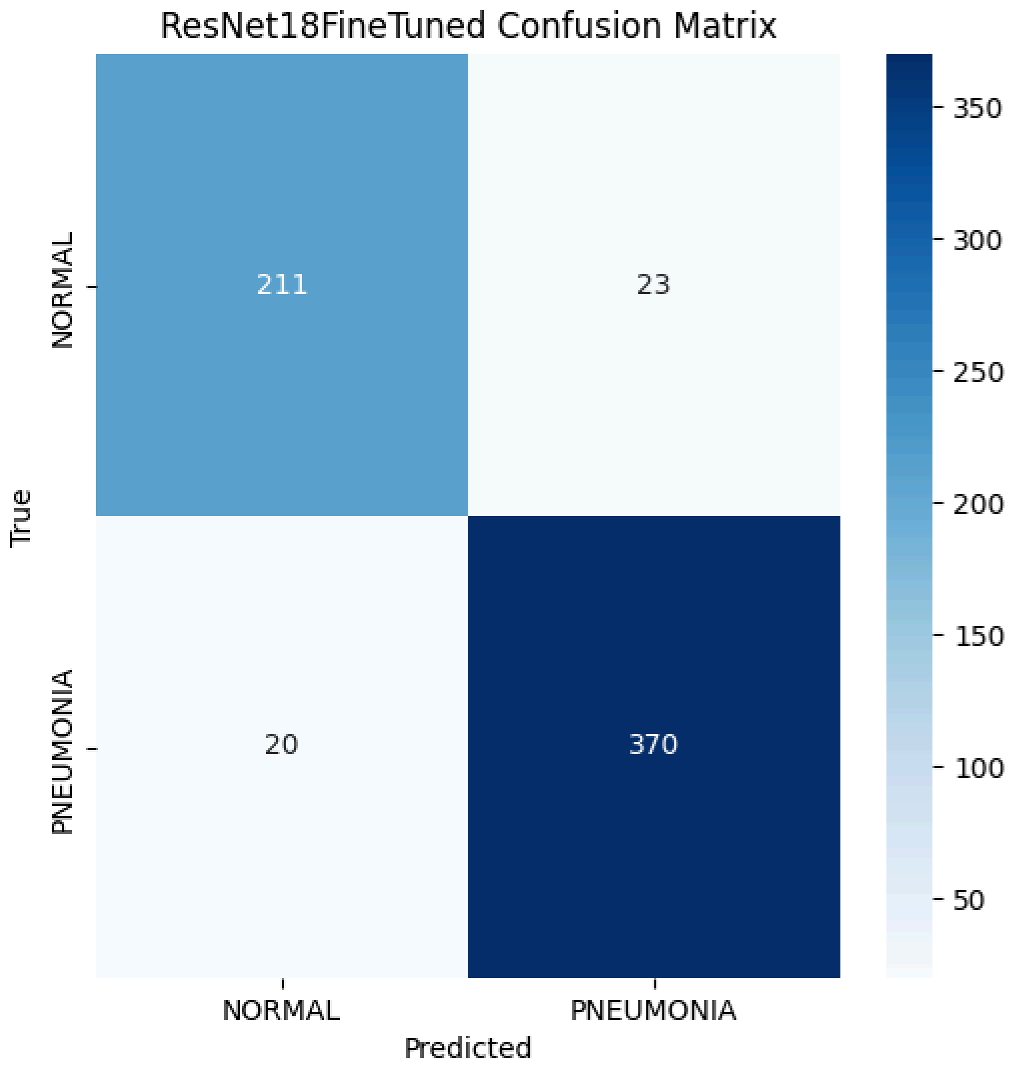
The CustomCNN and ResNet18FineTuned models were evaluated on a test set of 624 chest X-ray images (234 NORMAL, 390 PNEUMONIA) to assess their performance in detecting pneumonia. The CustomCNN achieved an accuracy of 81.57%, with a precision of 91.79%, recall of 64.62%, and F1-score of 75.97%, indicating strong performance but with limitations in sensitivity, likely due to class imbalance (1341 NORMAL vs. 3875 PNEUMONIA).

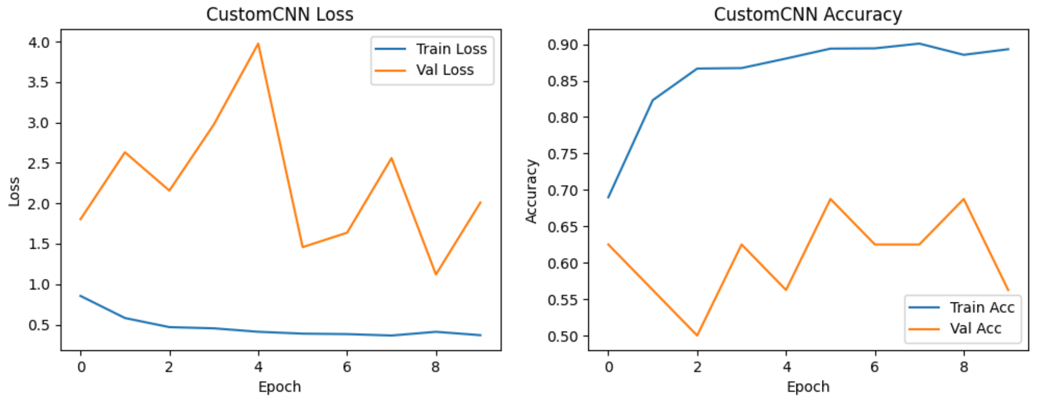
ResNet18

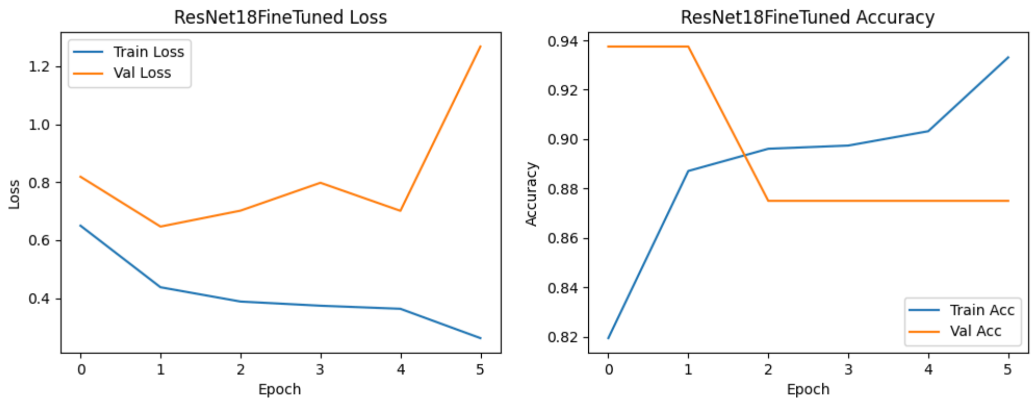
In contrast, ResNet18FineTuned, leveraging pre-trained features from ImageNet, significantly outperformed with an accuracy of 93.27%, precision of 95.79%, recall of 93.33%, and F1-score of 94.55%, demonstrating superior generalization and balanced detection of both classes.

Confusion Matrix and Training curves

Training curves showed stable convergence for both models, while confusion matrices (generated via evaluation.py) highlighted ResNet18's fewer false negatives, critical for medical diagnosis. These results underscore the effectiveness of transfer learning in ResNet18FineTuned, making it more suitable for clinical deployment, though CustomCNN remains a viable lightweight alternative.

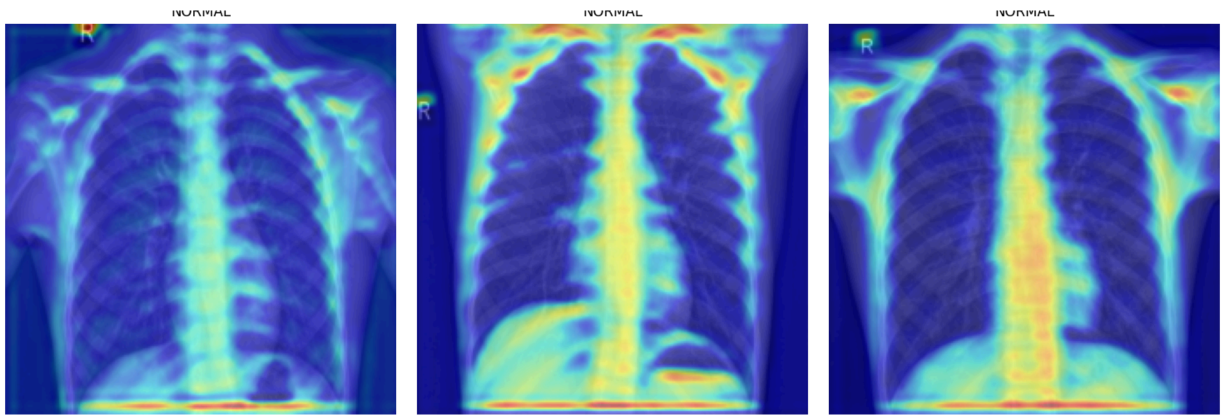

ResNet18FineTuned Confusion Matrix
True
NORMAL
PNEUMONIA
211
23
20
370
NORMAL
PNEUMONIA
Predicted
350
300
250
200
150
100
50


CustomCNN Loss
Train Loss
Val Loss
Loss
Epoch
CustomCNN Accuracy
Train Acc
Val Acc
Accuracy
Epoch



Visual examples (Grad-CAM)

CustomCNN



ResNet18