BMEN 4370 Biomedical Imaging Processing
Final Project

Mohana Gudhimalla

Due: 12/9/24
Date handed: 12/9/24

Abstract:
Images Chosen:

- The dataset contains images of fractured and non-fractured bones, particularly focused on hands, categorized into two classes [fractured and non-fractured].
- These images were resized to 128x128 and converted to grayscale for uniformity and simplicity in processing.

Algorithms Used:

- A Convolutional Neural Network (CNN) was designed with three convolutional layers, batch normalization, and pooling layers.
- The CNN was trained using the trainnet function, optimizing the cross-entropy loss function with stochastic gradient descent (SGD) and momentum.

Objective:

- The goal was to classify images as either "fractured" or "non-fractured" based on the training dataset and achieve high validation accuracy.
- Validate the model by testing it on a single external image and observing the predicted class and confidence

Expected Outcome:

- Train a robust classifier capable of distinguishing between fractured and non-fractured bones in hands with minimal misclassification.

Methods and Results:

Dataset Preparation: I started with a dataset containing images of fractured and non-fractured bones, organized into labeled folders. Images were first resized to 28 x 28 but the resolution of the images were very less so the trained images did not give the accuracy I wanted. I then used 128 x 128 pixels to ensure more clarity and compatibility with the CNN. I converted all images to grayscale to reduce complexity and maintain uniformity during training.

Processing: To resize, apply gray filter, and normalize the images to range of [0,1] I created a custom preprocessing function. I randomly displayed 20 images from the dataset to verify the integrity of the data. This visualization ensured that the images were correctly labeled and preprocessed. The dataset contained two classes: fractured and non-fractured bones. I counted the number of images per label to ensure balance. Each image's size was verified to confirm consistent dimensions of 128x128 pixels after preprocessing. The dataset was divided into training and validation sets. Each class contributed 750 images to the training set, while the remaining images were used for validation.

CNN structure:

- An input layer for 128x128 grayscale images.
- Three convolutional layers with 3x3 filters, batch normalization, and ReLU activation. The number of filters increased from 32 to 128 across layers.
- Two max pooling layers with a 2x2 kernel and stride of 2 for down sampling.
- A fully connected layer with two output nodes (fractured and non-fractured) and a SoftMax layer for classification probabilities.

Training: I trained the model using the SGDM optimizer with a learning rate of 0.005 for 12 epochs. I used different learning rates and epochs to train my set and chose them accordingly with my results. The dataset was shuffled after each epoch to prevent overfitting.

Testing:
After training, I tested the model on new images test.png which was a not fractured hand image and fracturedimg.png which was a fractured image. The test images were preprocessed using the same function as the dataset and reshaped for prediction. The predict function generated scores for each class, and the class with the highest probability was selected as the prediction for the image.

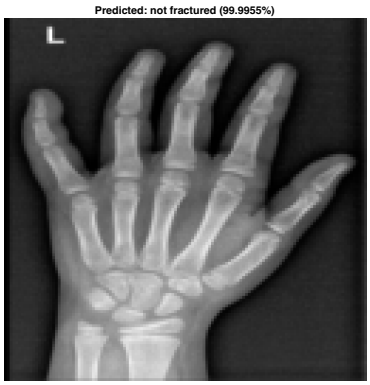
Results:

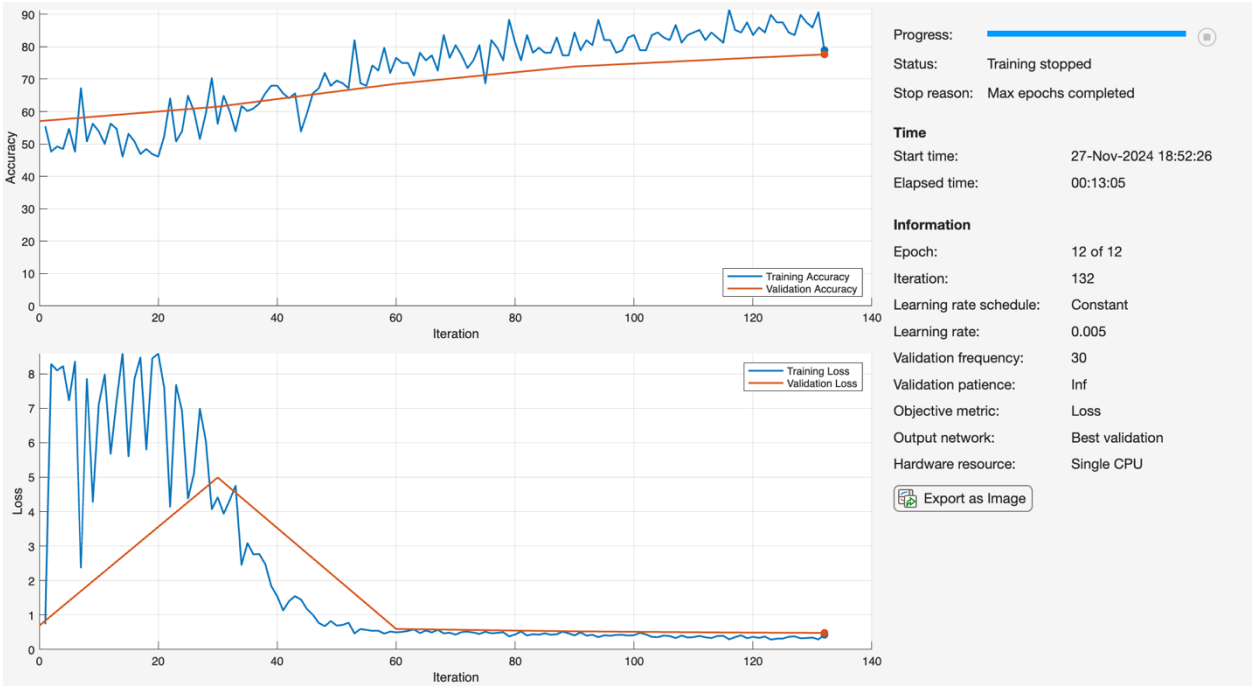
The model achieved a **validation accuracy of 80.284%**, indicating it correctly classified approximately 80% of the validation images. The training progress plot demonstrated consistent improvement in accuracy, reflecting effective learning over the epochs.

Phase 1: The first code (can be seen in appendix A) with minimal preprocessing:

For non-fractured image:

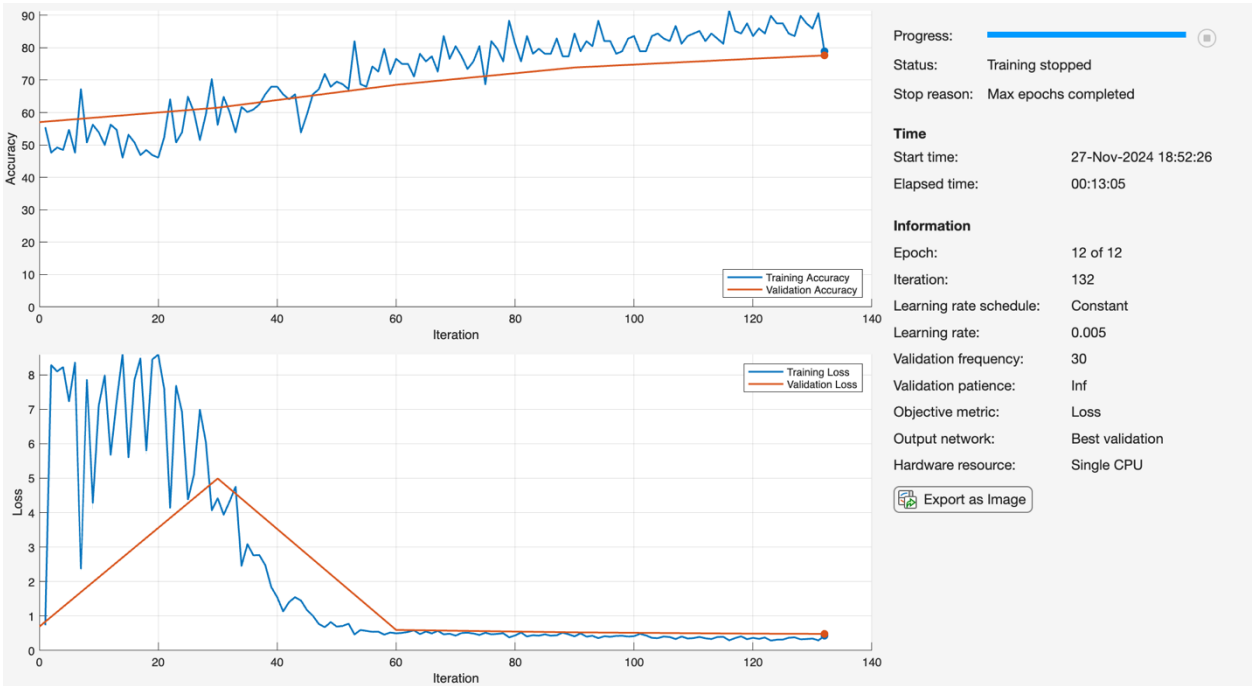
```
Test image: test.png
Validation Accuracy: 0.85985
Predicted Class: not fractured
Probability: 99.9955%
```



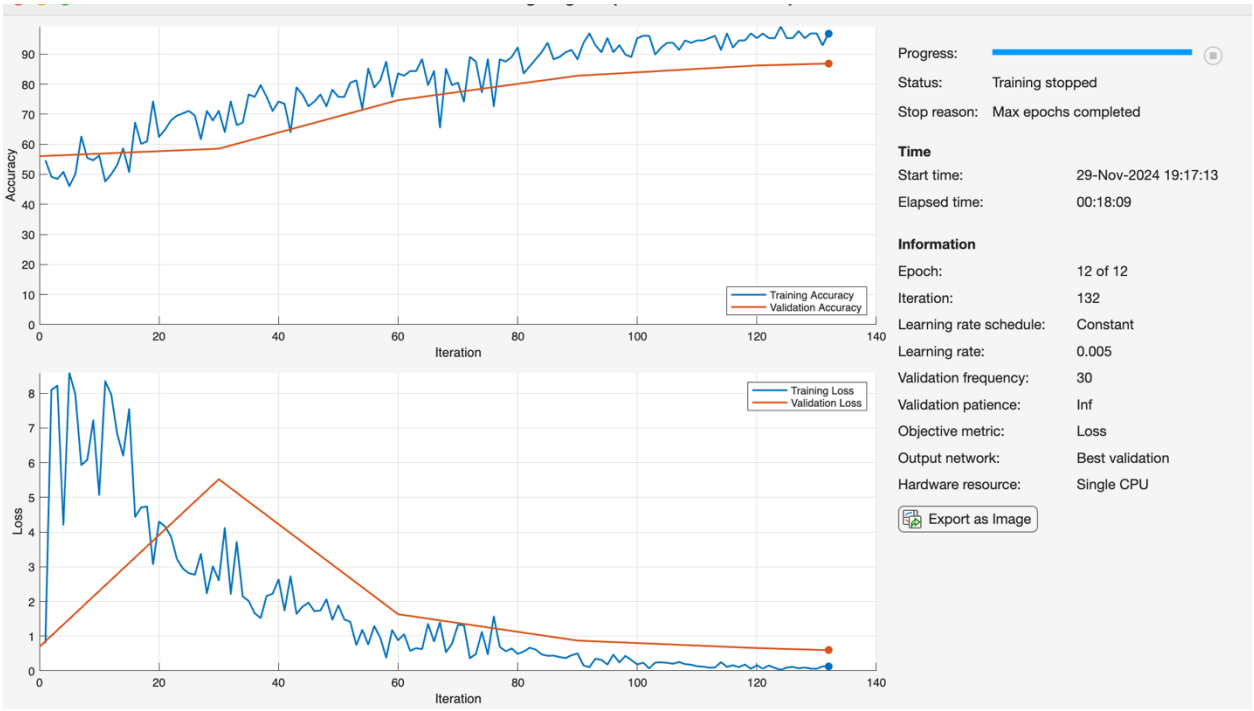


For fractured image:

```
Test image: fractureimg.png
Validation Accuracy: 0.77621
Predicted Class: fractured
Probability: 75.3982%
```



Validation Accuracy: 0.86852
Predicted Class: fractured
Probability: 76.1242%



Phase 2: In this phase, I have developed my preprocessing function to include more robust imaging techniques to increase the resolution of the images. I have also tried decreasing the sample training images to see how the accuracy is affected. But the accuracy was around 55%. Even though, the results predicted were all accurate to the images.

Validation Accuracy: 0.55655
Predicted Class: fractured
Probability: 100%

Validation Accuracy: 0.53284
Predicted Class: not fractured
Probability: 100%

Predicted: fractured (100%)

Validation Accuracy: 0.51351
Predicted Class: fractured
Probability: 100%
This model was able to predict even minor fractures.

Predicted: fractured (93.4696%)

After using 300 images for training, I got an increased validation accuracy:

Validation Accuracy: 0.61537
Predicted Class: fractured
Probability: 93.4696%

Therefore, increasing the size of the dataset gave us good results regarding the accuracy.

Summary

During this project, I gained insights about creating and training a convolutional network for binary classification. I learned about properly resizing, normalizing, and converting images to a consistent format significantly impacts the model's performance and training stability. I ensured an even distribution of images across classes preventing bias in the model, leading to more reliability. In future, I would ensure a more robust training network with even more accuracy that can help in differentiating images even more. Increasing the data set was also beneficial because it improved the second model. My model is useful because it helps in diagnosing fractures which can help support healthcare professionals by providing quick and accurate second opinions. Overall, this project sharpened my technical and analytical skills while opening the door to more advanced machine learning applications in real-world scenarios.

Appendix:

My teammate was Kayla, and she chose option 1 for this project. She chose noise reduction, histogram equalization, frequency domain and medical filter. For each image she chose images to identify different conditions. Especially interesting part was that she was very well versed in using techniques accordingly for the images. I like her very well planned layout for the project. The author she chose was Chris Wanstranth.

My author:
Margaret Elaine Hamilton was born on August 17, 1936. She is an American computer scientist. She was director of the Software Engineering Division of the MIT Instrumentation Laboratory, which developed on-board flight software for NASA's Apollo program. She later founded two software companies—Higher Order Software in 1976 and Hamilton Technologies in 1986, both in Cambridge, Massachusetts. Hamilton has published more than 130 papers, proceedings, and reports, about sixty projects, and six major programs. She coined the term "software engineering", stating "I began to use the term 'software engineering' to distinguish it from hardware and other kinds of engineering, yet treat each type of engineering as part of the overall systems engineering process."

Initial code is attached to the report and the preprocessed code is below:

```
% Unzip and Load Dataset
unzip("data.zip");
dataFolder = "data";
imds = imageDatastore(dataFolder, ...
    IncludeSubfolders=true, ...
    LabelSource="foldernames");

% Resize Images and Convert to Grayscale
imds.ReadFcn = @(filename) preprocessImage(filename);
```

```
%  Visualize Random Images
figure
tiledlayout("flow");
numImages = numel(imds.Files);
perm = randperm(numImages, 20); % Randomly select 20 images
for i = 1:20
    nexttile
    imshow(readimage(imds, perm(i)));
end

% Analyze Dataset
classNames = categories(imds.Labels);         % Unique class labels
labelCount = countEachLabel(imds);            % Count of images per label
img = readimage(imds, 1);                     % Read first image
disp("Image Size: " + string(size(img)));     % Display size of an image

%  Split Dataset into Training and Validation Sets
numTrainFiles = 300; % Reduced number of training images per class for faster
training
[imdsTrain, imdsValidation] = splitEachLabel(imds, numTrainFiles,
"randomize");

%  Simplified CNN Architecture
layers = [
    imageInputLayer([360 360 1])                        % Updated input size

    convolution2dLayer(3, 16, Padding="same")           % Conv Layer 1
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, Stride=2)

    convolution2dLayer(3, 32, Padding="same")          % Conv Layer 2
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, Stride=2)

    fullyConnectedLayer(2)                              % Fully Connected Layer
    softmaxLayer];                                      % Output Layer

%  Training Options
options = trainingOptions("sgdm", ...
    InitialLearnRate=0.01, ...                      % Higher learning rate for
faster convergence
    MaxEpochs=5, ...                                % Fewer epochs for quicker
training
    MiniBatchSize=64, ...                           % Larger batch size
    Shuffle="every-epoch", ...
    ValidationData=imdsValidation, ...
    ValidationFrequency=10, ...
    Plots="training-progress", ...
    Metrics="accuracy", ...
    Verbose=false);

%  Train the Model
net_1 = trainnet(imdsTrain, layers, "crossentropy", options);
```

```
% Evaluate the Model
scores = minibatchpredict(net_1, imdsValidation);         % Predict scores
YValidation = scores2label(scores, classNames);           % Predicted labels
TValidation = imdsValidation.Labels;                      % True labels
accuracy = mean(YValidation == TValidation);              % Calculate accuracy
disp("Validation Accuracy: " + string(accuracy));

% Custom Preprocessing Function
function img = preprocessImage(filename)
    % Read image
    img = imread(filename);
    % Convert to grayscale if the image is RGB
    if size(img, 3) == 3
        img = rgb2gray(img);
    end
    % Apply Adaptive Histogram Equalization for local contrast enhancement
    img = adapthisteq(img);
    % Resize to 360x360 for larger output
    img = imresize(img, [360 360], "bilinear");
    % Normalize to range [0, 1]
    img = double(img) / 255;
end

%  Test a Single Image
testImagePath = "smallfracture.png"; % Specify the path to the test image
testImage = preprocessImage(testImagePath);    % Preprocess the image

% Reshape Test Image for Prediction
testImageReshaped = reshape(testImage, [360, 360, 1]);  % Ensure proper input
dimensions

% Predict the class and scores
predictedScores = predict(net_1, testImageReshaped); % Get scores for each
class
[predictedProbability, idx] = max(predictedScores);  % Get max score and its
index

% Get Predicted Class
predictedClass = classNames(idx);                    % Map index to class
name

% Display Results
disp("Predicted Class: " + string(predictedClass));
disp("Probability: " + string(predictedProbability * 100) + "%");

% Visualize Test Image with Prediction
figure;
imshow(testImage, []);
title("Predicted: " + string(predictedClass) + " (" +
string(predictedProbability * 100) + "%)");
```