William Reed
CAP 4401 - Digital Image Processing
04/24/2024
Final Project
GitHub Repo: https://github.com/willmreed14/CAP4401-Final-Project/

Facial Recognition in MATLAB using Deep Learning

Motivation

Facial expression recognition has numerous practical applications in various fields, including human-computer interaction, security systems, and healthcare. Furthermore, deep learning techniques have shown remarkable performance in image recognition tasks, making them an ideal choice for this project.

Background

Facial Expression Recognition History

The study of facial expressions as a means to understand emotions dates back to the work of Charles Darwin, who first suggested that human expressions of emotion are universal across cultures in his book "The Expression of the Emotions in Man and Animals" (1872). However, the computational aspect of recognizing these expressions using algorithms began much later with advancements in digital imaging and computing.

Introduction of Deep Learning Approaches

The introduction of deep learning models, particularly Convolutional Neural Networks (CNNs), in image recognition tasks significantly advanced the field. These models are capable of learning hierarchical representations of image features, making them well-suited for the complex task of recognizing facial expressions, which involve subtle variations in facial muscle movements.

Key Contributions and Papers

- **1980s and 90s**: Early automated systems for facial expression recognition were proposed, which used techniques like optical flow and other handcrafted feature detection methods.
- **2000s**: The field saw the development of facial action coding systems and the beginning of using machine learning techniques for classification.
- **2010s**: With the advent of deep learning, researchers began to extensively use CNNs. A pivotal paper in this context was by Ian Goodfellow et al., "Challenges in Representation Learning: A report on three machine learning contests" (2013), which introduced the use of CNNs on the FER-2013 dataset, setting a benchmark for further research.
- **Recent Developments**: Current research often focuses on improving the accuracy and efficiency of these systems, handling diverse and uncontrolled environments, and integrating emotion recognition systems into various real-world applications such as surveillance, healthcare, and interactive services.\

Applications

Facial expression recognition has broad applications, ranging from enhancing user experience with computers, aiding in psychological studies, to more security-oriented applications in surveillance systems.

This project, building on these developments, uses MATLAB's Deep Learning Toolbox to implement a CNN for recognizing facial expressions from images, leveraging both the power of deep learning and the accessibility of MATLAB for prototyping and development.

Project Code

Since I wanted to stay committed to the topic of facial recognition like I had planned in my outline, I searched around the web for a working project to modify that was similar to the goals I had in mind. However, I didn't have much luck in finding an existing project that was both in line with my goals and simple enough for the scope of this project.

So, instead of modifying an existing project, I decided to build my own. I referenced previous assignments, used help from numerous sources on the internet, and utilized some ChatGPT guidance for debugging and understanding certain mechanisms in MATLAB.

The live script has three main sections, each with a dedicated task:

1. **Data Collection and Processing** - Acquire the dataset and perform initial preprocessing like resizing, normalization, and augmentation
2. **Model Design and Training** - Design a simple CNN architecture and train it using the processed dataset
3. **Model Evaluation** - Evaluate the model using metrics such as accuracy, precision, recall, and F1-score.

Data Collection and Processing

Acquire the dataset and perform pre-processing

```
% Define the paths to dataset
dataFolder = 'dataset';
% Define the paths to your training and testing directories
trainDir = fullfile(dataFolder, 'train');
testDir = fullfile(dataFolder, 'test');

% Create the imageDatastores and specify the ReadFcn for normalization
trainDatastore = imageDatastore(trainDir, 'IncludeSubfolders', true, 'LabelSource', 'foldernames', ...
    'ReadFcn', @(x) im2double(imread(x)));
testDatastore = imageDatastore(testDir, 'IncludeSubfolders', true, 'LabelSource', 'foldernames', ...
    'ReadFcn', @(x) im2double(imread(x)));
```

This section of the project code is setting up the groundwork to handle the image data used for training and testing a deep learning model for facial expression recognition.

The program reads images obtained from the CK+ facial recognition dataset, and defines which folders are for testing an which are for training. Then, it creates a respective datastore for each.

This setup ensures that the images are correctly loaded, labeled, and preprocessed, making them ready for the subsequent steps in building and training a deep learning model.

Define the Data Augmentation and create the augmented datastore

```
% Define augmentation options
augmenter = imageDataAugmenter(...
    'RandRotation', [-30, 30], ...
    'RandXTranslation', [-3, 3], ...
    'RandYTranslation', [-3, 3], ...
    'RandXScale', [0.8, 1.2], ...
    'RandYScale', [0.8, 1.2]);

% Using 48x48 grayscale images from CK+ dataset
imageSize = [48 48 1];  % Change to [48 48 3] if using color images

% Create augmented image datastore
trainAugmentedDatastore = augmentedImageDatastore(imageSize, trainDatastore, 'DataAugmentation', augmenter);
```

This section of the project code configures and creates an augmented image datastore to enhance the diversity of training data for a machine learning model, which is crucial for improving the model's generalization capability.

The program creates a new datastore that combines the original training images with the augmentation options defined. When the model is trained using this augmented datastore, it reads images that are dynamically modified according to the specified augmentation parameters. This process can improve the robustness of the facial expression recognition model by simulating a wider range of real-world conditions.

Model Design and Training

Define a simple CNN architecture used for facial expression recognition.

```
layers = [
    imageInputLayer(imageSize)

    convolution2dLayer(3, 8, 'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2, 'Stride', 2)

    convolution2dLayer(3, 16, 'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2, 'Stride', 2)

    convolution2dLayer(3, 32, 'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(7) % 7 neurons for the 7 classes in the data set
    softmaxLayer
    classificationLayer];
```

The network is structured to progressively refine and reduce the spatial size of the representation, abstracting the input image into a form that can be used to predict the emotional expression of the faces in the images. Each layer plays a role in extracting features, applying transformations, and finally classifying these features into distinct classes.

Configure training options and train the network

```
options = trainingOptions('sgdm', ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',30, ...
    'Shuffle','every-epoch', ...
    'ValidationData',testDatastore, ...
    'ValidationFrequency',30, ...
    'Verbose',true, ...
    'Plots','training-progress');

net = trainNetwork(trainDatastore, layers, options);
```

The training options are configured in a very similar way to the past few projects in this course. This setup is common in deep learning workflows where the model's performance and training progress need to be closely monitored to make adjustments if necessary, ensuring optimal learning and performance on the task of facial expression recognition.

Model Evaluation

Calculate accuracy, precision, recall, and F-1 score for each facial expression

```
YPred = classify(net, testDatastore);
YTest = testDatastore.Labels;

% Calculate accuracy
accuracy = sum(YPred == YTest)/numel(YTest);
disp(['Accuracy: ', num2str(accuracy)])
```

```
Accuracy: 1
```

```

% Calculate precision, recall, and F1-score
confMat = confusionmat(YTest, YPred);
precision = diag(confMat)./sum(confMat, 2);
recall = diag(confMat)./sum(confMat, 1)';
f1Scores = 2*(precision.*recall)./(precision + recall);

disp(table(unique(YTest), precision, recall, f1Scores, 'VariableNames',
```

Sticking to my plan in the project proposal, I evaluated the model's performance using accuracy, precision, recall, and f1 score. However, I noticed a possible issue, which is that according to these metrics the model performed perfectly, as they all have a value of 1. Since it is fairly impractical that the model actually performed perfectly, I think I have an error in the logic of calculating these evaluation metrics.

Lessons Learned & Ideas for the Future

Throughout this project, I gained a much better understanding of how Deep Learning and CNNs are designed and implemented. Starting from scratch helped me understand each step of the process a lot better, since I was writing the code myself instead of editing what was already there. Debugging and researching along the way taught me little by little about how each piece of the entire network comes together.

If I had more time to work on this project, I would experiment with several things, such as:

- **Training Options**: I would like to test out different combinations of options, such as changing optimizers and number of epochs.
- **Data Set**: I would like to test my model on a more complex data set, such as more photos of each expressions, or possibly even more classifications of facial emotions.
- **Model Evaluation**: If I had more time, I would certainly fine tune my evaluation metrics to ensure they are giving accurate readings, as well as enable a more in-depth summary of how the model performs.