# Face Recognition Using Neural Networks

CMPT 412 Assignment 4

Alec Pershick

301190478

# Introduction

Machine learning is widely used to solve computer vision problems in the world today.  In this assignment, I set out to use machine learning with neural networks to develop a solution for facial recognition. Using Matlab and its Deep Learning Toolbox, I was able to put together a great algorithm for solving this problem.

# Image Preprocessing

Originally, I had tried to train the model without preprocessing. This had resulted in inconsistent results, with training data exploding every which way. This sparked me to try to preprocess the image after being read into the file.

```
function image = smoothImage(filename)
    img = imread(filename);
    image = imgaussfilt(img, 4);
end

function image = normalizeImage(filename)
    img = imread(filename);
    img = mat2gray(img);
    image = im2double(img);
end

function image = contrastImage(filename)
    img = imread(filename);
    image = imadjust(img);
end

function image = SmoothAndContrastImage(filename)
    img = imread(filename);
    image = imadjust(img);
    image = imgaussfilt(img, 4);

end
```

I first experimented with normalizing the image data. I created a function *normalizeImage()* which normalized the image between the range 0 and 1. This however had a negative affect on validation accuracy in my model, so decided to try other methods.

Next I tried smoothing the image with a function called *smoothImage()*. In my code, *smoothImage()* takes the image file and applies gaussian filtering to it in order to smooth out the pixels. This is theory, should have made for more consistent results. However in practice, it did not seem to affect my program too much.

The last preprocessing strategy I tried was to contrast enhance the image. This I did by creating a function *contrastImage()* that applied the built-in *imadjust()* function to the image. Furthermore, I combined both contrasting and smoothing together in a preprocessing function called SmoothAndContrastImage() and this seemed to make my results more consistent.

## Network Architecture

To further enhance my results, tinkering around with the layers in my network architecture was a step that needed to be worked in.

```
layers = [
    imageInputLayer([112 92 1])

    convolution2dLayer(2,8,'Padding',1)
    reluLayer
    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(2,16,'Padding',1)
    reluLayer
    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(2,32,'Padding',1)
    reluLayer

    fullyConnectedLayer(40)
    softmaxLayer
    classificationLayer];
```

The best setup I could achieve came with having an image input layer of *imageInputLayer([112 92 1])*, which describes the dimensions of the image we are working with. Followed by 3 separate convolution layers with a filter size of 2. The difference between the 3 convolution layers is that I changed the number of filters by a factor of 2 each time, starting with *convolution2dLayer(2,8,'Padding',1)*, and ending with *convolution2dLayer(2,32,'Padding',1).*

These convolution layers are each followed by a nonlinear activation function specified by a reluLayer, and then a max-pooling layer in between each convolution layer. The max-pooling layer downsamples the image, and is useful in between convolution layers when there are multiple in the network. I used the parameters *maxPooling2dLayer(2,'Stride',2)* which contains a step size (stride) of 2 when it scans across the image.

I ended the network architecture using 3 layers including a fully connected layer with parameter of 40, meaning that there are 40 classes in the target data. The fully

connected layer connects all of the neurons of the previous layers and finds the underlying patterns in the data. This was then followed by a softmax layer, which uses the softmax activation function for classification, and then a classification layer, which uses the probabilities returned from the softmax layer and assigns it to each class.

## Training Options

The last thing to tweak in my program was the training parameters made available by the *options* keyword which is being passed into the training network function. The training options I used were as follows:

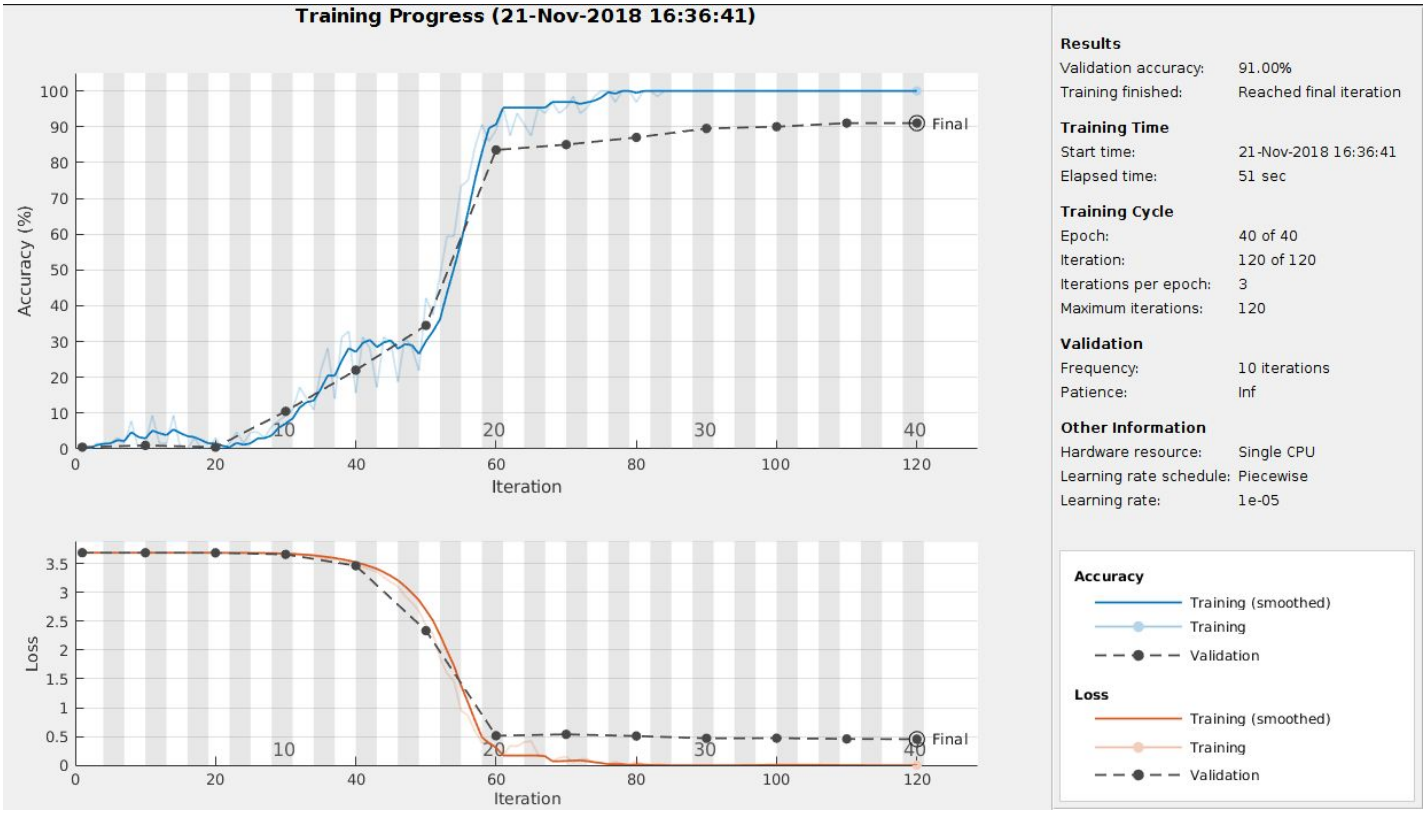
```
options = trainingOptions('sgdm', ...
'LearnRateSchedule','piecewise', ...
'InitialLearnRate',0.01, ...
'LearnRateDropFactor',0.1, ...
'LearnRateDropPeriod',10, ...
'MaxEpochs',40, ...
'MiniBatchSize',64, ...
'ValidationData',imdsValidation, ...
'ValidationFrequency',10, ...
'Plots','training-progress')
```

Starting with the learn rate schedule, I found that instead of having a constant learning rate, it was more beneficial to implement a dynamic learning rate which decreased as the training data became more learned. This helped with eliminating random spikes causing my training data to fluctuate abnormally. I used a piecewise schedule with an initial learn rate of 0.01, that was multiplied by 0.1 every 10 epochs.

I used a maximum of 10 epochs throughout the training, which meant it would go through 40 full cycles on the whole training data. I used a batch size of 64 in my training set, this made for 3 iterations per epoch. I also set the validation frequency to 10, which seemed to somewhat follow the smoothed training curve. This seemed to be a perfect number for the training data given, and gave excellent results when implemented.

## Results

Results before utilizing the strategies above varied, generally with fairly low results that stagnated at a certain point and would not continue. After implementing everything above however, I managed to consistently achieve > 80% accuracy on each run of the training set. With the right tweaking I was able to get 85-95% consistently with each run through the training algorithm. As an example below, I copied a run with a nice upward trend that achieved 100% batch accuracy at around the 80th epoch and forward.



## Extra Thoughts

Overall I thought this method was fairly productive for facial recognition and could be adapted for other types of recognition as well. In the real world, one might consider using python along with tensorflow or something along those lines for more scalable approaches at deep learning however.

## Full Code

```
digitDatasetPath = fullfile('/home/alec/Documents/412/A4', 'orl_faces');
imds = imageDatastore(digitDatasetPath, ...
  'IncludeSubfolders',true,'LabelSource','foldernames','ReadFcn',
@SmoothAndContrastImage);

% imds.ReadFcn = @smoothImage;
% imds.ReadFcn = @contrastImage;
% imds.ReadFcn = @normalizeImage;
% imds.ReadFcn = @SmoothAndContractImage;

close all;

figure;
perm = randperm(400,20);
for i = 1:20
  subplot(4,5,i);
  imshow(imds.Files{perm(i)});
end

labelCount = countEachLabel(imds);
```

```
img = readimage(imds,1);
size(img)

numTrainFiles = 5;
[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');

layers = [
  imageInputLayer([112 92 1])
  convolution2dLayer(2,8,'Padding',1)
  reluLayer
  maxPooling2dLayer(2,'Stride',2)

  convolution2dLayer(2,16,'Padding',1)
  reluLayer
  maxPooling2dLayer(2,'Stride',2)

  convolution2dLayer(2,32,'Padding',1)
  reluLayer

  fullyConnectedLayer(40)
  softmaxLayer
  classificationLayer];

options = trainingOptions('sgdm', ...
'LearnRateSchedule','piecewise', ...
'InitialLearnRate',0.01, ...
'LearnRateDropFactor',0.1, ...
'LearnRateDropPeriod',10, ...
```

```
'MaxEpochs',40, ...
'MiniBatchSize',64, ...
'ValidationData',imdsValidation, ...
'ValidationFrequency',10, ...
'Plots','training-progress')

net = trainNetwork(imdsTrain,layers,options);

YPred = classify(net,imdsValidation);
YValidation = imdsValidation.Labels;

accuracy = sum(YPred == YValidation)/numel(YValidation)

function image = smoothImage(filename)
  img = imread(filename);
  image = imgaussfilt(img, 4);
end

function image = normalizeImage(filename)
  img = imread(filename);
  img = mat2gray(img);
  image = im2double(img);
end

function image = contrastImage(filename)
  img = imread(filename);
  image = imadjust(img);
end
```

```
function image = SmoothAndContrastImage(filename)
  img = imread(filename);
  image = imadjust(img);
  image = imgaussfilt(img, 4);
end
```