Table of Contents

1. Introduction

The recent advancements in deep learning, especially transfer learning, have significantly transformed how we approach image classification and detection.

This report reflects the implementation of Transfer learning with Convolutional Neural Networks (CNNs) using pretrained models, to classify the images based on their features and attributes, using image processing techniques such as (Feature/edge detection and feature extractions).

The dataset used for this assignment is called “Shoe vs sandal vs Boot dataset” contains 3 classes of images called ‘Boot, Sandal, Shoe

1.1 Objective of the Coursework

The main objective of this project is to explore the effectiveness of transfer learning in the context of image classification and detection. Specifically, I aim to address the following research questions:

- How does transfer learning impact the performance of a pretrained model when applied to a diverse image dataset?
- What strategies yield the best results in fine-tuning the model for optimal classification accuracy?
- How does the choice of pretrained model affect the final classification and detection outcomes?

1.2 An Overview of the Report Content

This reports starts with laying out the questions (as done above) and my objectives throughout this project. Following which, I have elaborated the dataset using sample images, techniques that I have used to process the images before inputting the into a pretrained network which continues with highlighting the approach and network I chose for training.

The main goal with this coursework isn't just to learn how to apply transfer learning—it's more about advancing the understanding of how these techniques can be best used for classifying images, especially when it comes to categorising diverse set of images. This report is a detailed account of how I approached the project, what I have discovered along the way, and my thoughts and conclusion on the entire process.

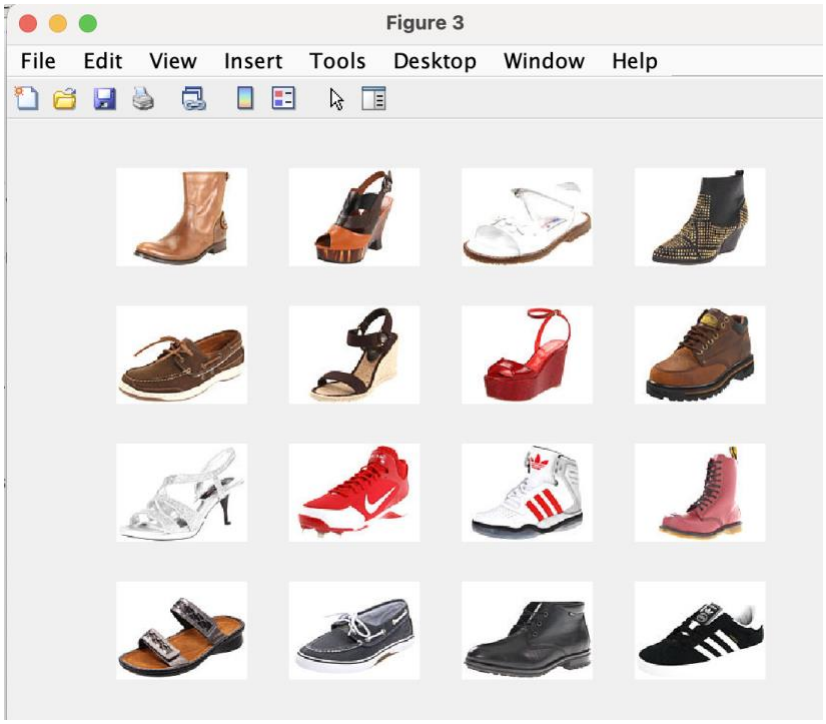
2. Simulations

2.1 Description of the dataset.

The dataset that I have used is called “Shoe vs sandal vs Boot dataset” downloaded from an online source called Kaggle. This dataset is consists of 3 classes (Boot, Shoe and Sandal). Each class name is very self-explanatory and identifies the different type of images in the dataset. The total number of images are 15000 as 5000 per each class.

The reason to choose this dataset specifically is the usability rating as per Kaggle and its diverse compatibility with CNNs and deep learning models which defines the flexibility of it.

Following are the sample images from the dataset to get a general idea of how diverse the data is;



Code used to generate this plot :

```
% Plotting a grid of random images from different classes
numTrainImages = numel(imdsTrain.Labels); idx = randperm(numTrainImages,16);
figure
for i = 1:16
    subplot(4,4,i)
    I = readimage(imdsTrain,idx(i));
    imshow(I)
end
```

2.2 Preprocessing and preparing the data for training.

Starting with the Image preprocessing part, I have used multiple techniques such as (Resizing images to a consistent size, Converting images to grayscale, Normalising pixel range and data augmentation).

Following is a snippet of a code used for image preprocessing.

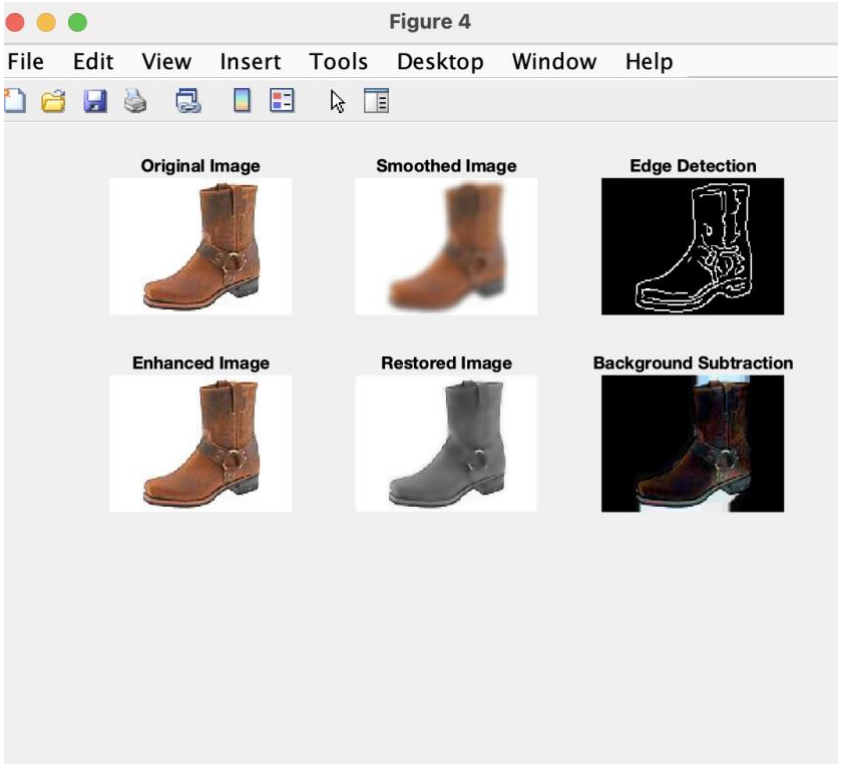
```
%%%%%%%%%%%%%% Image preprocessing techniques %%%%%%%%%%%%%%%%


%         <Define target size for resizing>
targetSize = [224 224]; % Specify the desired dimensions [height width]

%         <Resize training images>
imdsTrainResized = augmentedImageDatastore(targetSize, imdsTrain);
%         <Resize validation images>
imdsValidationResized = augmentedImageDatastore(targetSize, imdsValidation);

%         <Convert training images to grayscale>
imdsTrainGray = transform(imdsTrainResized, @(x) rgb2gray(x));
%         <Convert validation images to grayscale>
imdsValidationGray = transform(imdsValidationResized, @(x) rgb2gray(x));


inputSize = net.Layers(1).InputSize;

pixelRange = [-30 30];
imageAugmenter = imageDataAugmenter( ...
 'RandXReflection',true, ...
 'RandXTranslation',pixelRange, ...
 'RandYTranslation',pixelRange);
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
 'DataAugmentation',imageAugmenter);

augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);
```

I have used further image processing techniques to showcase the diversity of data that can be extracted from the images for model training. This includes techniques like Noise reduction, Edge detection, Image Enhancements (by adjusting the contrast of each RGB channel separately), Image Restoration (by applying Wiener Filter) and Background subtraction.

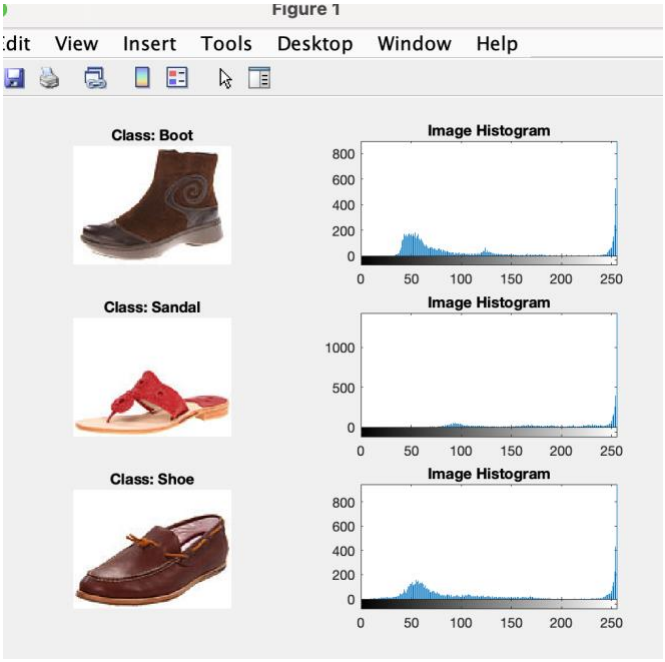
Following is the a code snippet to showcase the process using syntax.

```
%%%%%%%%%%%%%%%% Image processing Techniques    %%%%%%%%%%%%%%%%


% Original Image
im_original = img;

% Noise Reduction (Gaussian Smoothing)
sigma = 2;
im_smoothed = imgaussfilt(im_original, sigma); % Apply Gaussian smoothing with specified sigma

% Edge Detection (Canny Edge Detection)
edges = edge(rgb2gray(im_original), 'Canny'); % Perform Canny edge detection on the grayscale version of the image

% Image Enhancements (Contrast Adjustment)
% Adjust contrast of each RGB channel separately
im_enhanced = imadjust(im_original, [0 1], [], 1); % Adjust contrast using default parameters

%  Image Restoration (Wiener Filter)
% Convert RGB image to grayscale for noise reduction
im_gray = rgb2gray(im_original);

% Apply Wiener filter for restoration
noise_var = 0.01; % Estimate of noise variance
im_restored = wiener2(im_gray, [5 5], noise_var); % Specify filter size and noise variance

% since imds is an imageDatastore containing multiple images
imds_processed = transform(imds, @(x) imgaussfilt(x, sigma)); % Example: Apply Gaussian smoothing to all images

% Background Subtraction (Simple Background Subtraction)
background = imopen(im_original, strel('disk', 15)); % Estimate background
foreground = im_original - background; % Perform background subtraction

% Display results for visualization
figure;
subplot(3, 3, 1); imshow(im_original); title('Original Image');
subplot(3, 3, 2); imshow(im_smoothed); title('Smoothed Image');
subplot(3, 3, 3); imshow(edges); title('Edge Detection');
subplot(3, 3, 4); imshow(im_enhanced); title('Enhanced Image');
subplot(3, 3, 5); imshow(im_restored); title('Restored Image');
subplot(3, 3, 6); imshow(foreground); title('Background Subtraction');
```

Following is a visualization of the results.



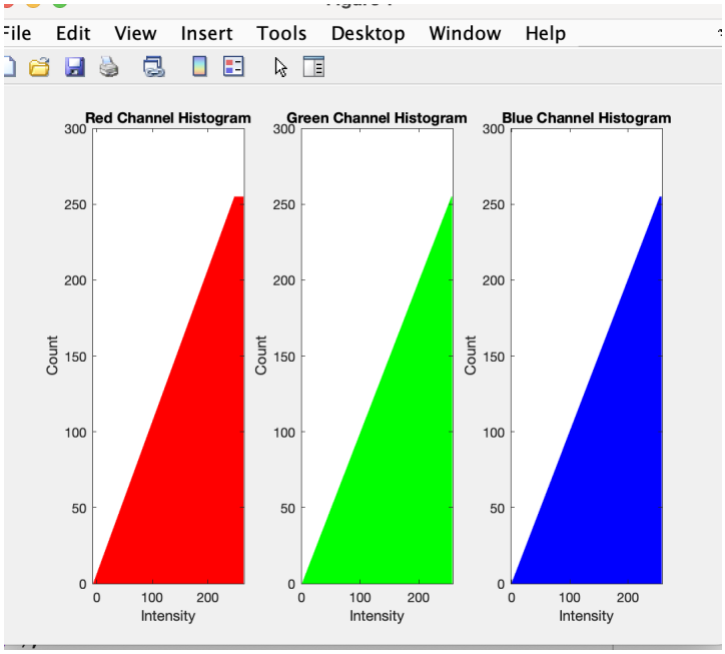
I have also plot a figure showing a comparison of histogram of 3 images (1 image per each class) to show the difference in the data attributes. Following is the code snippet along with the visualization.



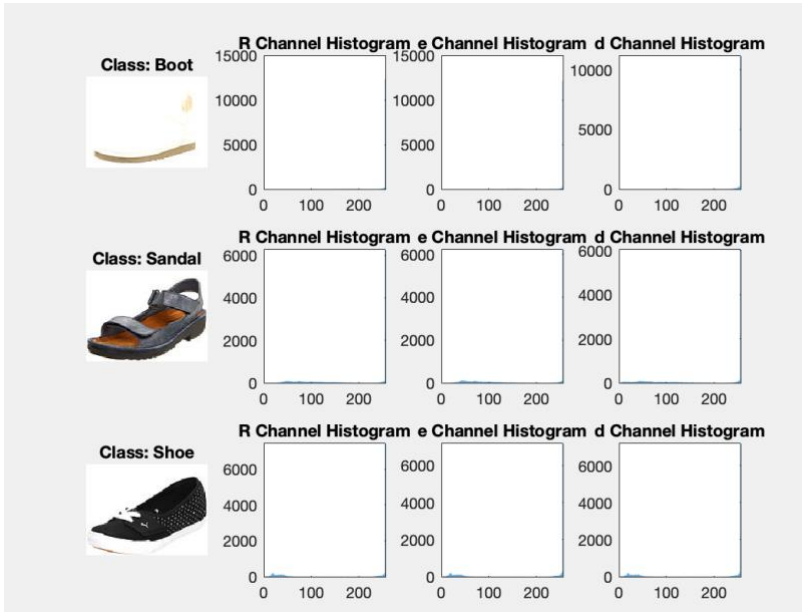
Furthermore, I have used data processing techniques such as Feature extraction, Extracting HOG features, Extracting Local Binary Patterns (LBP), and Extracting Color Histograms – I have also stored the extracted image features in the mat files(Called – Trainingfeatures.mat and Validationfeatures.mat) in order to pipeline them for the further network training.

I have also visualised the colored histograms to get a good idea of the color diversity.

Following is the code snippet along with the results for the color histograms



Following is a visualization of a color histogram of each class.



2.3 The network architecture along with training, validation and testing the network, explain the learning algorithm used.

Network Architecture:

The network architecture used in this project is based on GoogleNet which a deep CNN(convolutional neural network) which seems ideal for image classification and feature extraction. The initial model (`net`) is loaded as GoogLeNet using the `googlenet` function as (net = googlenet;). Following with feature extraction for which I have chosen “pool5-7x7_s1”, which extract features from the final pooling layer of GoogLeNet just prior to the classification layers. To utilise the network for transfer

learning the final classification layers were replaced with new layers tailored to the specific task, such as the number of classes represented by `imdsTrain.Labels`. These modifications allows the network to be fine-tuned and optimised for a particular classification task.

Training, Validation, and Testing:

The process is based on multiple stages starting with defining the network architecture, training the model, and evaluating its performance:

1. Network Modification:

   After creating a layer graph (lgraph) from the pre-trained GoogLeNet model (net), I have modified the network architecture by replacing the final layers (learnableLayer and classLayer) with new layers (newLearnableLayer and newClassLayer) which are customised for the process.

```
%>>>>>>>>>>>> Training a network model (Googlenet) <<<<<<<<<<<<<<<<<<

% Create a layer graph from the pre-trained GoogLeNet model (net)
lgraph = layerGraph(net);

% Display the layers of the layer graph
lgraph.Layers

% Find the layers that can be replaced during transfer learning
[learnableLayer,classLayer] = findLayersToReplace(lgraph);

% Display the layers identified for replacement
[learnableLayer,classLayer]
```

```
% Replace the identified learnable layer with the new learnable layer
lgraph = replaceLayer(lgraph,learnableLayer.Name,newLearnableLayer);

% Create a new classification output layer
newClassLayer = classificationLayer('Name','new_classoutput');

% Replace the existing classification layer with the new classification laye
lgraph = replaceLayer(lgraph,classLayer.Name,newClassLayer);
```

2. Data Preprocessing:

   Since the images are already preprocessed for training and validation as “augimdsTrain” and “augimdsValidation” by using imageDataAugmenter function to apply random transformations and augmentations to increase dataset diversity and robustness.

```
% Retrieve the input size (height and width) of the first layer of the neural network
inputSize = net.Layers(1).InputSize;

% Define a range for random pixel translations (horizontal and vertical)
pixelRange = [-30 30];

% Create an imageDataAugmenter object to perform data augmentation on training images
% - RandXReflection: Randomly flip images horizontally (left-right reflection)
% - RandXTranslation: Randomly shift images horizontally within the specified pixel range
% - RandYTranslation: Randomly shift images vertically within the specified pixel range
imageAugmenter = imageDataAugmenter( ...
 'RandXReflection',true, ...
 'RandXTranslation',pixelRange, ...
 'RandYTranslation',pixelRange);

% Create an augmentedImageDatastore for training data
% - Resizes training images to match the inputSize (height and width) of the neural network
% - Applies the specified image augmentation techniques (imageAugmenter) during training
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
 'DataAugmentation',imageAugmenter);

% Create an augmentedImageDatastore for validation data
% - Resizes validation images to match the inputSize (height and width) of the neural network
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);
```

3. Training Options:

TrainingOptions using a variable name “options” were specified for training the modified network “netTransfer” using the RMSprop optimizer with specific configurations (mini-batch size, max epochs, initial learning rate, learning rate schedule, etc.).

I have only used a 7% of the whole dataset for training to reduce the processing time and only 3% for the Testing part.

```
% [imdsTrain,imdsValidation] = splitEachLabel(imds,0.7,'randomized');
[imdsRemain,imdsTrain,imdsValidation] = splitEachLabel(imds,0.9,0.07,0.03,'randomized');
```

```
%%

% >>>>>>>>>. Traininig Network <<<<<<<

% Modify training options with a higher initial learning rate and more epochs
options = trainingOptions("rmsprop", ...
    'MiniBatchSize', 32, ...
    'MaxEpochs', 10, ...
    'InitialLearnRate', 1e-4, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.1, ...
    'LearnRateDropPeriod', 5, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', augimdsValidation, ...
    'ValidationFrequency', 10, ...
    'Verbose', true, ...
    'Plots', 'training-progress');


% Train the modified transfer learning network
netTransfer = trainNetwork(augimdsTrain, lgraph, options);
```

4. Training the Network

The modified network “netTransfer” was trained using trainNetwork function with the augmented training data “augimdsTrain”, the modified network architecture “lgraph”, and the specified training options “options”.

```
% Train the modified transfer learning network
netTransfer = trainNetwork(augimdsTrain, lgraph, options);
```

5. Evaluation:

The trained model “netTransfer” is evaluated on the augmented validation dataset “augimdsValidation” to assess its performance using classification accuracy and confusion matrix visualization.

```
% >>>>>> Evaluation and Confuion matrix Visualisation <<<<<<<<<<

% Evaluate the fine-tuned model on validation data
predictedLabels = classify(netTransfer, augimdsValidation);
trueLabels = imdsValidation.Labels;

% Calculate classification accuracy
accuracy = mean(predictedLabels == trueLabels) * 100;
fprintf('Validation Accuracy: %.2f%%\n', accuracy);

% Plot confusion matrix
figure;
cm = confusionchart(trueLabels, predictedLabels);
cm.Title = 'Confusion Matrix for Validation Data';
cm.ColumnSummary = 'column-normalized';
cm.RowSummary = 'row-normalized';
```

Learning Algorithm (RMSprop)

For learning algorithm, I have used RMSprop (Root Mean Square Propagation) in trainingOptions, which is a version of stochastic gradient descent (SGD), it adjusts the learning rate for each parameter based on the average of the magnitudes of recent gradients for that parameter. This adaptive learning rate strategy helps to optimise the training of deep neural networks more effectively by justifying issues such as vanishing or exploding gradients.

3. Results Obtained

3.1 Accuracy achieved

Coming to the most crucial part of the project was to make sure to achieve a higher possible accuracy to ensure the best functionality.

For this I have tried 2 different pretrained models to compare and analyse the best practices to improve the.

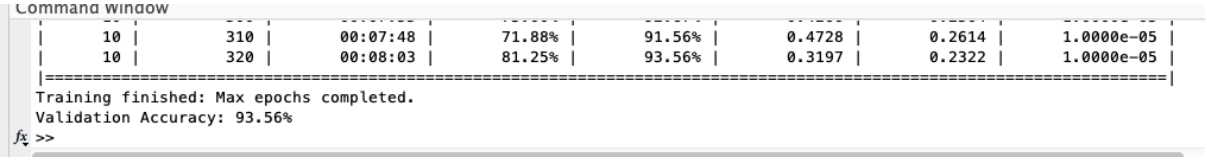
Alexnet:

Initially I had use Alexnet as (net = alexnet;) with the following settings;

```
% Modify training options with a higher initial learning rate and more epochs
options = trainingOptions('rmsprop', ...
    'MiniBatchSize', 32, ...
    'MaxEpochs', 10, ...
    'InitialLearnRate', 1e-4, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.1, ...
    'LearnRateDropPeriod', 5, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', augimdsValidation, ...
    'ValidationFrequency', 10, ...
    'Verbose', true, ...
    'Plots', 'training-progress');


% Train the modified transfer learning network (alexnet 86%)
netTransfer = trainNetwork(augimdsTrain, layers, options);
```

The final tests resulted in 93% of validation accuracy, which was already an improved result.

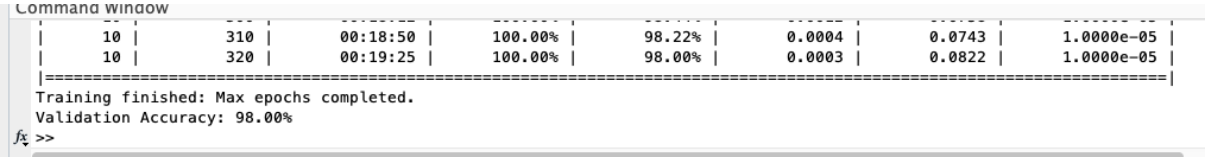
Following screenshots shows the evidence of the accuracy while using Alexnet;



GoogleNet:

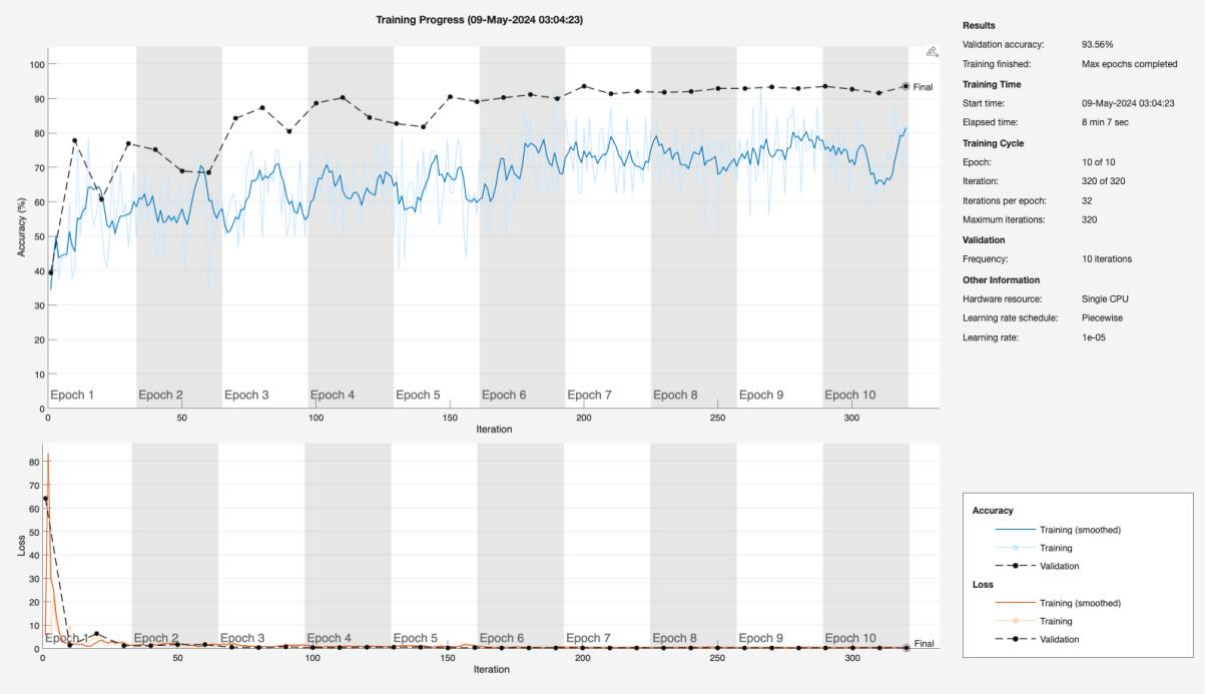
I have then proceeded with GoogleNet as it was the ideal competitor and a better pretrained network for CNN transfer learning and image processing.

The final accuracy I have achieved using GoogleNet is 98% - as shown in the below attached images.

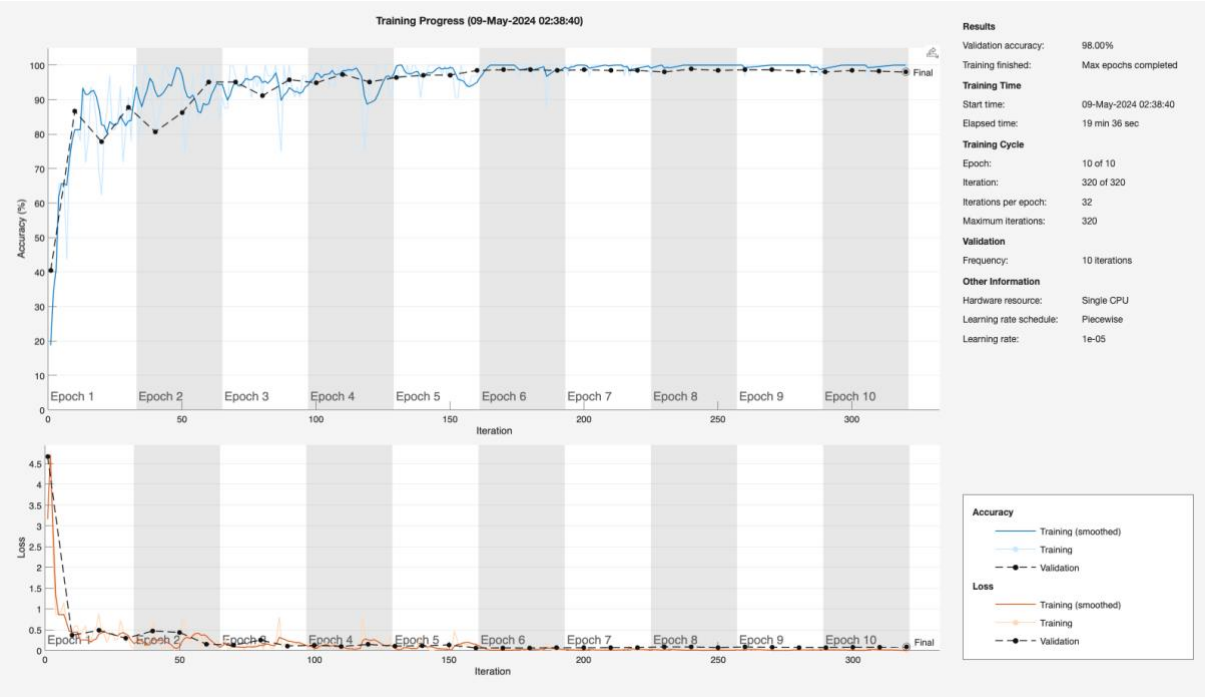


3.1 Accuracy curve graphs

Accuracy curve graph(Alexnet)



Accuracy curve graph(GoogleNet)



3.2 Confusion Matrices

The following confusion matrix provides a visual representation to evaluate how well a fine-tuned model performs on a validation dataset

The confusion matrix presented here provides a visual representation to evaluate how well a fine-tuned model performs on a validation dataset. Each row of the matrix represents the actual class labels “trueLabels” in this case (Boot, Sandal, shoe) from the validation data, while each column represents the predicted classLabels generated by the model. This matrix helps us understand the model's classification accuracy across different categories.

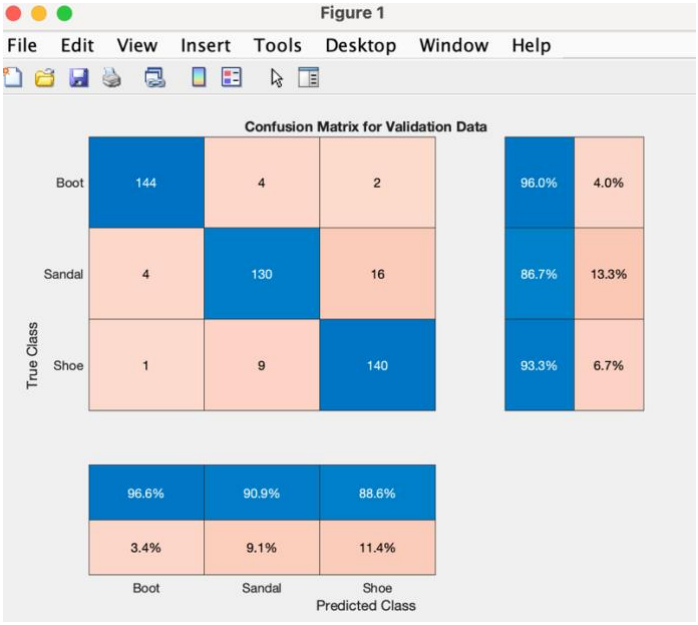
The diagonal elements of the confusion matrix show the number (or percentage) of correctly predicted instances for each class. Essentially, they indicate how well the model is able to accurately predict each specific class. The off-diagonal elements reveal instances where the model has misclassified samples, showing which classes are often confused with each other.

It can be seen from the following matrix that there were very few misclassified instances in comparison with the accurate predictions.

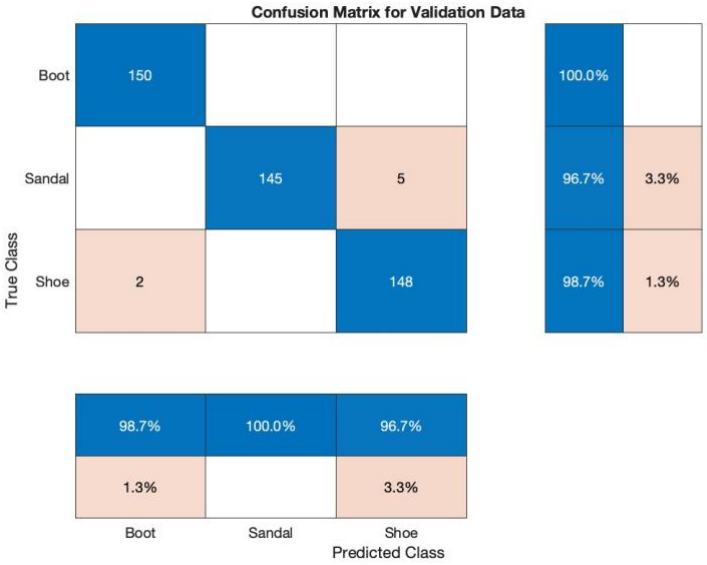
Moreover, the column summary and row summary provide normalized views of the predictions and true labels, respectively. The column summary displays the proportion of predicted labels for each class relative to the total predictions, offering insights into the model's prediction tendencies. On the other hand, the row summary depicts the distribution of true labels for each class relative to the total samples in that class, shedding light on the dataset's class distribution and the model's performance across different categories.

In summary, the confusion matrix serves as a valuable visualisation tool for assessing classification performance, identifying areas where the model excels and areas where it may need improvement. This analysis helps guide decisions for refining the model or adjusting the dataset to enhance overall accuracy and reliability.

Confusion Matrix (AlexNet)



Confusion Matrix (GoogleNet)



4. Critical analysis of results

4.1 Alterations done to achieve results.

Improvements in AlexNet;

In order to achieve higher accuracy while using Alexnet I had to make various changes in the training settings which led to a huge improvement from 76% to 93%. Which verify how different training settings can enhance a model's learning and generalisation capabilities.

Firstly, I have changed the solver from “sgdm” (Stochastic Gradient Descent with Momentum) to “adam” (Adaptive Moment Estimation) which already proved to be beneficial. The Adam optimiser adapts learning rates for each parameter based on gradient moments, which make it more effective in handling sparse gradients and noisy data. This adaptability likely facilitated smoother convergence and faster progress towards a good solution, resulting in an accuracy increase to 86% compared to the initial 76%.

For further improvements, I have changed the solver to “rmsprp” (Root Mean Square Propagation). Since RMSprop adjusts learning rates based on the magnitude of gradients, it address issues like vanishing or exploding gradients during training, which improved the handling of gradient dynamics, which most likely contributed to better generalisation and increased the accuracy up to 93%.

Moreover, the adjustments in “mini-batch size” and epochs played a role in enhancing performance too. Initially I was using ‘10’, Increasing the mini-batch size from 10 to 32 allowed the model to process more data simultaneously, leading to more stable updates in the model parameters and potentially better generalisation. Increasing the training duration with more epochs (from 6 to 10) also provides additional learning opportunities, leading to capture the underlying patterns in the data more comprehensively.

The implementation of a ‘piecewise’ learning rate schedule with a drop factor and period also helped prevent overfitting by gradually reducing the learning rate during training. This adaptive strategy fine-tuned the model's parameters effectively over time, contributing to improved generalisation and higher accuracy on unseen data.

In summary, the reiterative refinement of training settings, including the choice of optimiser, mini-batch size, epochs, and learning rate schedule, collectively enhanced the model's ability to learn and generalize from the data. Each adjustment addressed specific challenges in training deep neural networks, ultimately resulting in a more accurate and robust model for the classification task at hand. These insights underscore the importance of thoughtful parameter tuning and optimisation in a deep learning model development.

Improvements in GoogleNet;

After achieving improved accuracy with previous tests, I applied similar techniques to fine-tune a GoogLeNet model, resulting in an exceptional accuracy of 98%. This involved leveraging transfer learning by adapting the pre-trained GoogLeNet architecture for the specific classification task. By replacing the final layers with custom fully connected and classification layers, the model was tailored to the target classes. Similar to Alexnet model, optimised training options also played a crucial role in maximizing accuracy.

4.2 Detailed analysis and discussion

In order to summarise the achievement of highly accurate results, I would say that there are several key aspects that contributed. Such as Transfer learning from a pre-trained network like GoogLeNet allowed us to take advantage of learned features from a large dataset and adapt them to our specific classification task. This approach saves computation time and resources by building upon existing knowledge within the network's weights.

Moreover, the choice of RMSprop as an optimiser played a crucial role in achieving high accuracy. RMSprop's adaptive learning rate mechanism, which adjusts based on recent gradient magnitudes, stabilized the training process and eliminated various processing challenges, which contributed to a more effective model training. Additionally, Incorporating data augmentation techniques during training, significantly enhanced the model's ability to generalise. By exposing the model to varied versions of the training data, it learned robust features that were invariant to small variations in input, leading to improved performance on unseen data.

Furthermore, the fine-tuning parameters like mini-batch size, number of epochs, and learning rate schedules optimised the training process, ensuring effective model convergence and meaningful pattern learning from the data.

The validation accuracy of 98% indicates strong generalisation capabilities of the model, demonstrating its ability to make accurate predictions on new instances beyond the training set. Which is considered excellent for many practical applications in computer vision, such as image recognition and object detection.

Looking ahead, future improvements in could involve exploring different model architectures, experimenting with additional data augmentation strategies, or fine-tuning hyperparameters to further enhance performance. Regular monitoring and evaluation of the model's performance on real-world data will be essential for maintaining accuracy and reliability in practical scenarios.

In summary, this analysis showcases the successful application of transfer learning, optimised training methods, and thoughtful parameter tuning in developing a high-accuracy GoogLeNet model. These results underscores the efficacy of deep learning approaches in tackling complex image classification tasks and allows a path for further advancements and real-world dataset testing and research.

5. Conclusion

In summary, I would like to point out my findings from this course. The transfer learning on a pretrained network like GoogLeNet offers significant advantages and challenges in developing deep learning models. Choosing a pretrained network allow you a utilization of learned features from extensive datasets, such as ImageNet, which accelerates a model development and adaptation to specific tasks with minimal data requirements. This approach reduces computational demands and training time, making it an efficient strategy for deploying deep learning models quickly. However, selecting and fine-tuning a pretrained network poses challenges, as it requires a careful consideration of model complexity, task-specific requirements, optimal layer freezing and retraining strategies in order to achieve optimal performance without overfitting.

Using a compatible IDE such as MATLAB provides a comprehensive ecosystem equipped with prebuilt networks, training algorithms, and data preprocessing tools. MATLAB's integration with various computing resources facilitates efficient model training, specially with large datasets, and enables the use of advanced optimisation techniques like RMSprop. Coding deep learning models in MATLAB involves careful parameter tuning, selection of appropriate training options, and validation strategies to ensure high accuracy and robust generalization. MATLAB's visualization tools, such as training progress plots and confusion matrices, aid in monitoring model performance and evaluating results effectively.

5.1 Key takeaways from this project:

Following are the key takeaways from this project;

1. Transfer learning with GoogLeNet efficiently speeds up model development by beniffiting pretrained features.

2. Optimised training techniques like RMSprp and data augmentation significantly enhance model performance and accuracy.

3. Selection and fine-tuning pretrained networks require careful balancing of complexity and task-specific requirements.

4. Confidence in practicing and keep improving the datasets