Lyles College of Engineering
Department of Electrical and Computer Engineering

Technical Report

Experiment Title:	Final Project
Course Title:	ECE 172 Fundamentals of Machine Learning
Instructor:	Dr. Hovannes Kulhandjian
Date Submitted:	May 6 2023

Prepared By:	Sections Written:
Puya Fard	All sections

INSTRUCTOR SECTION

Comments:

Final Grade: Team Member 1: Puya Fard

TABLE OF CONTENTS

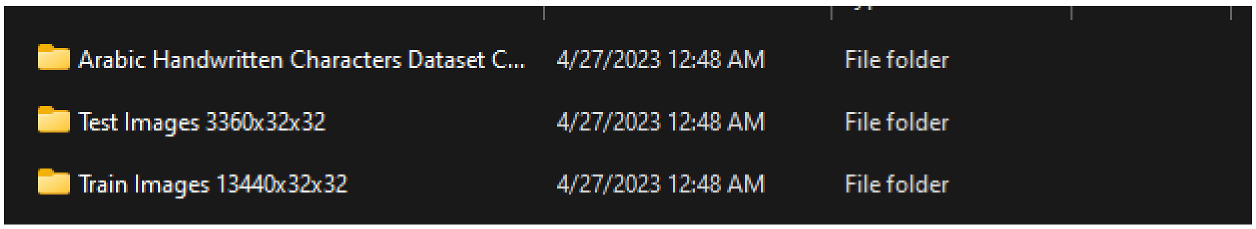
1. STATEMENT OF OBJECTIVES

The objective of this project is to develop and train a deep convolutional neural network (DCNN) architecture in Matlab using a professor approved image dataset, handwritten arabic characters. The aim is to achieve high accuracy in image classification by optimizing the learning rate and gradient descent algorithm. The project also involves plotting the accuracy vs epochs and iterations and analyzing the confusion matrix to identify the best performing classifier. Additionally, three pre-trained models from the Matlab Toolbox Deep Neural Designer will be trained, and their accuracy vs epoch will be plotted for comparison with the results obtained from the DCNN model. The goal is to identify the best performing model and comment on the performance and training time of each algorithm.

2. THEORETICAL BACKGROUND

The Dataset: Handwritten Arabic characters

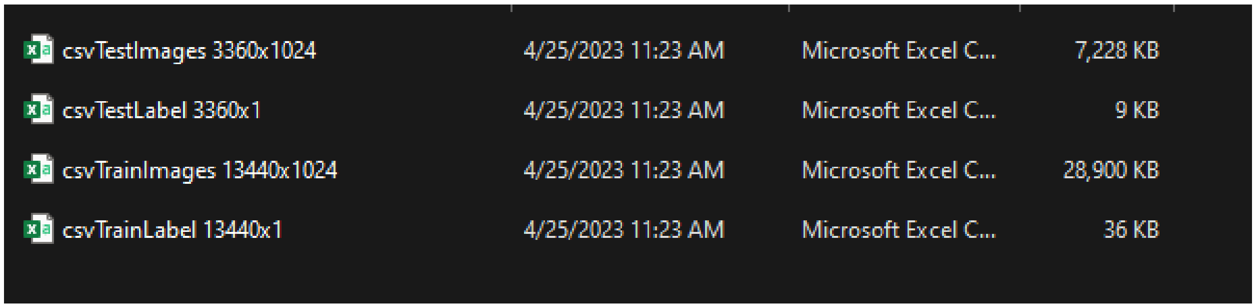
The handwritten Arabic character dataset used in this project consists of approximately 3400 training images and over 10000 test images, each of size 32x32 pixels. The dataset was collected specifically for this project and includes a diverse range of Arabic characters written by individuals with varying handwriting styles. The images were preprocessed to ensure that they are of high quality and are suitable for training a deep convolutional neural network (DCNN) model. The large size of the test set allows for a thorough evaluation of the model's performance on unseen data. The dataset serves as a valuable resource for research in the field of Arabic character recognition and can be used for further analysis and benchmarking of other DCNN architectures.



Arabic Handwritten Characters Dataset C...	4/27/2023 12:48 AM	File folder
Test Images 3360x32x32	4/27/2023 12:48 AM	File folder
Train Images 13440x32x32	4/27/2023 12:48 AM	File folder

Figure 2.1: Dataset

The labels of these images are also included in the folder named **Arabic Handwritten Character Dataset Csv.**



csvTestImages 3360x1024	4/25/2023 11:23 AM	Microsoft Excel C...	7,228 KB
csvTestLabel 3360x1	4/25/2023 11:23 AM	Microsoft Excel C...	9 KB
csvTrainImages 13440x1024	4/25/2023 11:23 AM	Microsoft Excel C...	28,900 KB
csvTrainLabel 13440x1	4/25/2023 11:23 AM	Microsoft Excel C...	36 KB

Figure 2.2: Labels

Deep Convolutional Neural Network (DCNN) - source: ("Understanding Deep Convolutional Neural Networks")

The strength of DCNNs is in their layering. A DCNN uses a three-dimensional neural network to process the Red, Green, and Blue elements of the image at the same time. This considerably reduces the number of artificial neurons required to process an image, compared to traditional feed forward neural networks.

Deep convolutional neural networks receive images as an input and use them to train a classifier. The network employs a special mathematical operation called a "convolution" instead of matrix multiplication.

The architecture of a convolutional network typically consists of four types of layers: convolution, pooling, activation, and fully connected.

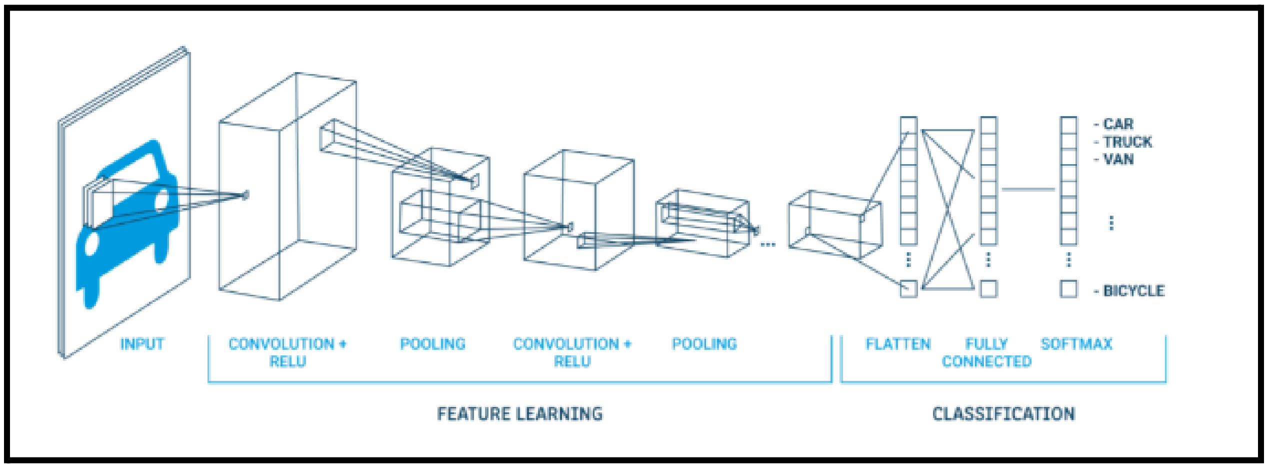
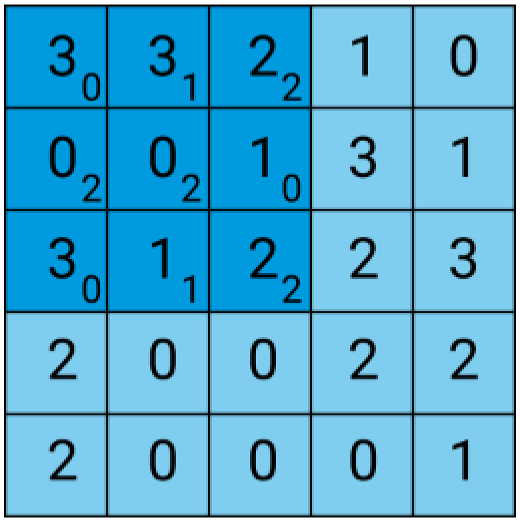


Figure 2.3: DCNN Architecture

Convolutional Layer

Applies a convolution filter to the image to detect features of the image. Here is how this process works:

- A convolution—takes a set of weights and multiplies them with inputs from the neural network.
- Kernels or filters—during the multiplication process, a kernel (applied for 2D arrays of weights) or a filter (applied for 3D structures) passes over an image multiple times. To cover the entire image, the filter is applied from right to left and from top to bottom.
- Dot or scalar product—a mathematical process performed during the convolution. Each filter multiplies the weights with different input values. The total inputs are summed, providing a unique value for each filter position.



3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

Figure 2.4: Matrix

ReLU Activation Layer

The convolution maps are passed through a nonlinear activation layer, such as Rectified Linear Unit (ReLu), which replaces negative numbers of the filtered images with zeros.

Pooling Layer

The pooling layers gradually reduce the size of the image, keeping only the most important information. For example, for each group of 4 pixels, the pixel having the maximum value is retained (this is called max pooling), or only the average is retained (average pooling).

Pooling layers help control overfitting by reducing the number of calculations and parameters in the network.

Fully Connected Layer

In many CNN architectures, there are multiple fully connected layers, with activation and pooling layers in between them. Fully connected layers receive an input vector containing the flattened pixels of the image, which have been filtered, corrected and reduced by convolution and pooling layers. The softmax function is applied at the end to the outputs of the fully connected layers, giving the probability of a class the image belongs to – for example, is it a car, a boat or an airplane.

After several iterations of convolution and pooling layers (in some deep convolutional neural network architectures this may happen thousands of times), at the end of the network there is a traditional multi layer perceptron or "fully connected" neural network.

Darknet19 - source: ("DarkNet-19 convolutional neural network - MATLAB darknet19")

DarkNet-19 is a convolutional neural network that is 19 layers deep. You can load a pre-trained version of the network trained on more than a million images from the ImageNet database [1]. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 256-by-256.

You can use classify to classify new images using the DarkNet-19 model. Follow the steps of Classify Image Using GoogLeNet and replace GoogLeNet with DarkNet-19.

DarkNet-19 is often used as the foundation for object detection problems and YOLO workflows [2]. For an example of how to train a you only look once (YOLO) v2 object detector, see Object Detection Using YOLO v2 Deep Learning. This example uses ResNet-50 for feature extraction. You can also use other pretrained networks such as DarkNet-19, DarkNet-53, MobileNet-v2, or ResNet-18 depending on application requirements.

net = darknet19 returns a DarkNet-19 network trained on the ImageNet data set.

This function requires the Deep Learning Toolbox™ Model for DarkNet-19 Network support package. If this support package is not installed, then the function provides a download link.

net = darknet19('Weights','imagenet') returns a DarkNet-19 network trained on the ImageNet data set. This syntax is equivalent to net = darknet19.

layers = darknet19('Weights','none') returns the untrained DarkNet-19 network architecture. The untrained model does not require the support package.

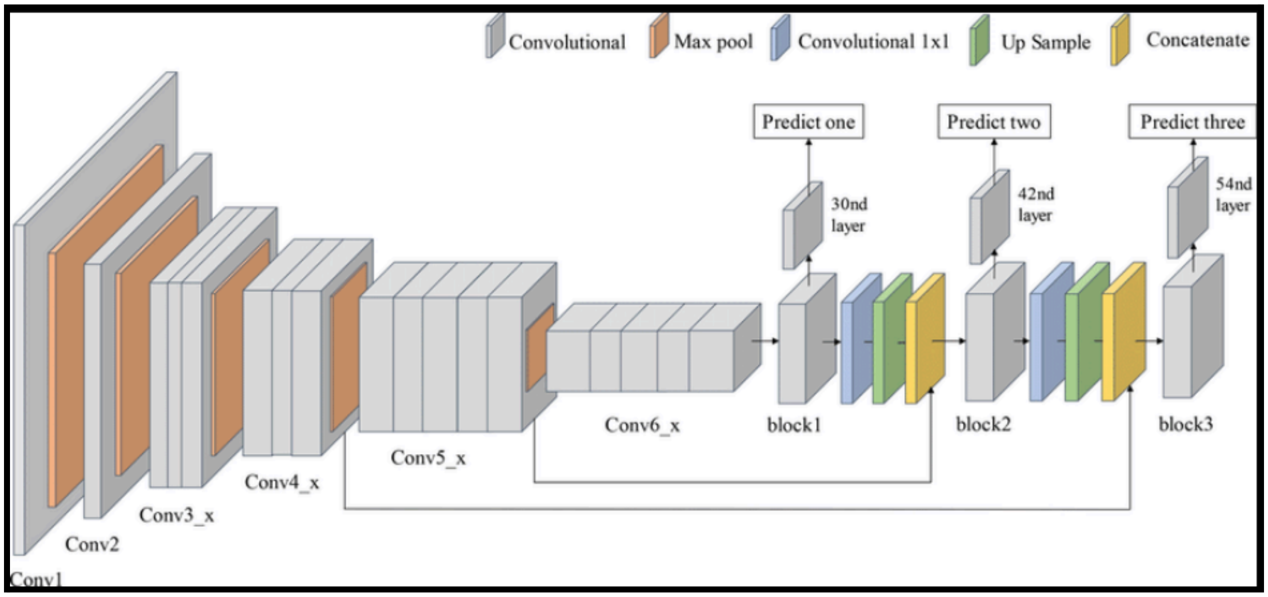


Figure 2.5: Darknet19 Architecture

ResNet-50 - source: ("ResNet-50 convolutional neural network - MATLAB resnet50")

ResNet-50 is a convolutional neural network that is 50 layers deep. You can load a pre trained version of the neural network trained on more than a million images from the ImageNet database [1]. The pre trained neural network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the neural network has learned rich feature representations for a wide range of images. The neural network has an image input size of 224-by-224. For more pre-trained neural networks in MATLAB®, see Pre Trained Deep Neural Networks.

You can use classify to classify new images using the ResNet-50 model. Follow the steps of Classify Image Using GoogLeNet and replace GoogLeNet with ResNet-50.

To retrain the neural network on a new classification task, follow the steps of Train Deep Learning Network to Classify New Images and load ResNet-50 instead of GoogLeNet.

net = resnet50 returns a ResNet-50 neural network trained on the ImageNet data set.

This function requires the Deep Learning Toolbox™ Model for ResNet-50 Network support package. If this support package is not installed, then the function provides a download link.

net = resnet50('Weights','imagenet') returns a ResNet-50 neural network trained on the ImageNet data set. This syntax is equivalent to net = resnet50.

lgraph = resnet50('Weights','none') returns the untrained ResNet-50 neural network architecture. The untrained model does not require the support package.

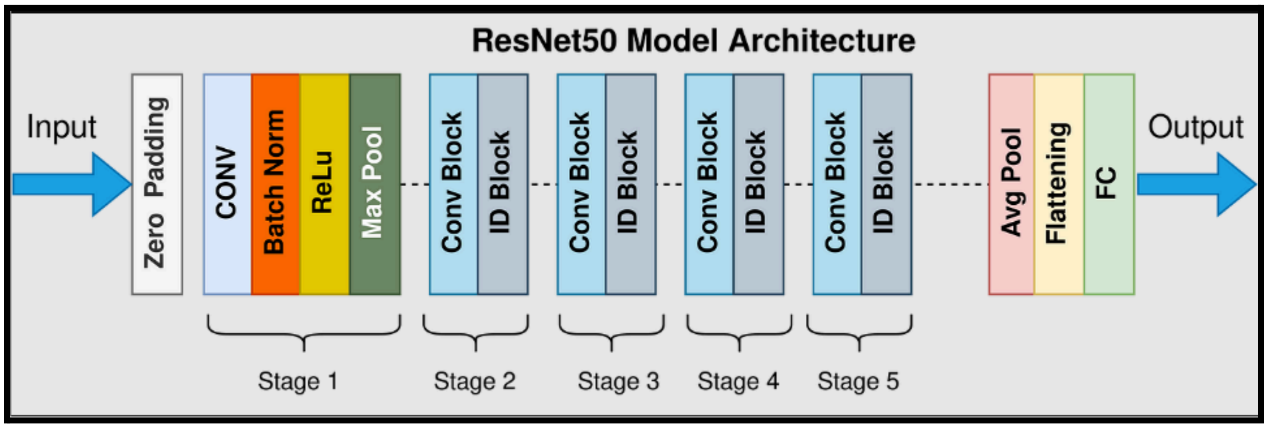


Figure 2.6: ResNet50 Architecture

Efficientnetb0

EfficientNet-b0 is a convolutional neural network that is trained on more than a million images from the ImageNet database [1]. The network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224. For more pretrained networks in MATLAB®, see Pre Trained Deep Neural Networks.

You can use classify to classify new images using the EfficientNet-b0 model. Follow the steps of Classify Image Using GoogLeNet and replace GoogLeNet with EfficientNet-b0.

To retrain the network on a new classification task, follow the steps of Train Deep Learning Network to Classify New Images and load EfficientNet-b0 instead of GoogLeNet.

If the Deep Learning Toolbox Model for EfficientNet-b0 Network support package is not installed, then the function provides a link to the required support package in the Add-On Explorer. To install the support package, click the link, and then click Install. Check that the installation is successful by typing efficientnetb0 at the command line. If the required support package is installed, then the function returns a DAGNetwork object.

3. EXPERIMENTAL PROCEDURE

3.1 Equipment used

1. Personal Computer
2. MATLAB R2023a
3. Online research journals
4. Online researched videos

3.2 Project Procedure Description

3.2.1. Task 1: Implement DCNN Architecture on dataset

1. Load the image data using the imageDatastore function. The data is organized into subfolders, with each subfolder containing images of a specific class.

2. Split the data into training and validation sets using the splitEachLabel function. In this case, 80% of the data is used for training, and 20% is used for validation.

3. Load the test data using the imageDatastore function. This data will be used to evaluate the trained network.

4. Define the architecture of the DCNN using a layer graph. The architecture consists of multiple convolutional layers, each followed by a ReLU activation function and a max pooling layer. The final layers are a global average pooling layer, a fully connected layer, a softmax layer, and a classification layer.

5. Replace the final fully connected layer and softmax layer with new layers that are tailored to the number of classes in the dataset. This is done using the replaceLayer function.

6. Define an imageDataAugmenter object to perform data augmentation during training. This includes random rotations, reflections, shears, and grayscale-to-RGB color conversion.

7. Create an augmentedImageDatastore object to handle the augmented data during training and validation.

8. Define the training options, including the optimization algorithm, number of epochs, batch size, learning rate, and validation data. Train the network using the trainNetwork function. The resulting trained network is saved to a file.

9. During testing, load the trained network from the saved file and use it to classify the test images.

3.2.2. Task 2: Implement Darknet19 Architecture on dataset

1. Install Darknet: Install the Darknet framework on your machine. You can download it from the official website https://github.com/pjreddie/darknet.

2. Prepare the dataset: You need to prepare your dataset in the correct format for Darknet. Darknet requires the dataset to be in the YOLO format. You can use tools like LabelImg to create bounding boxes for your images and save them in the YOLO format.

3. Create configuration files: Darknet requires two configuration files, one for the network architecture and one for the dataset. You need to create a network configuration file that specifies the layers and parameters for the Darknet19 architecture. You also need to create a dataset configuration file that specifies the location of the training and validation images, as well as the number of classes and the names of the classes.

4. Train the network: Use the Darknet command-line interface to train the network. Specify the location of the network and dataset configuration files, as well as the number of iterations to train for. Darknet will save the trained weights to a file.

5. Evaluate the network: Use the Darknet command-line interface to evaluate the performance of the trained network on a test set. Darknet will output the precision, recall, and average precision for each class.

6. Fine-tune the network: If the performance of the network is not satisfactory, you can fine-tune the network by adjusting the hyperparameters or using data augmentation techniques.

7. Deploy the network: Once the network is trained and evaluated, you can use it to make predictions on new data. You can use the Darknet command-line interface or the Darknet API to do this.

3.2.3. Task 3: Implement ResNet-50 Architecture on dataset

1. Prepare the dataset: You need to prepare your dataset and organize it in a way that can be used by your code. This includes resizing the images, creating a data augmentation pipeline, and splitting the dataset into training, validation, and testing sets.

2. Load the pre-trained model: You can use transfer learning to speed up the training process by using a pre-trained ResNet-50 model. You can download the pre-trained model from the official TensorFlow website or other sources. Once you have the model, you can load it into your code.

3. Modify the last layer: Since your dataset likely has a different number of classes than the pre-trained model was trained on, you need to modify the last layer of the model to match the number of classes in your dataset.

4. Fine-tune the model: Now that you have loaded the pre-trained model and modified the last layer, you can fine-tune the model on your dataset. Fine-tuning involves training the model on your dataset while keeping the pre-trained weights fixed for some of the layers.

5. Train the model: Once you have fine-tuned the model, you can train it on your dataset. This involves training all of the layers of the model, including the last layer that you modified.

6. Evaluate the model: After training the model, you should evaluate its performance on the testing set. You can calculate metrics such as accuracy, precision, recall, and F1-score to evaluate the performance of your model.

7. Save the model: Finally, you should save the trained model so that you can use it for inference on new data in the future. You can save the model in various formats such as TensorFlow SavedModel, Keras H5, or ONNX.

3.2.4. Task 4: Implement Efficient B0 Architecture on dataset

1. Prepare the dataset: You need to prepare your dataset by splitting it into training, validation, and test sets. It's important to have a balanced dataset and ensure that all classes are represented equally.

2. Load the pre-trained model: Download the pre-trained Efficient B0 model weights and load them into the model.

3. Fine-tune the model: Fine-tune the pre-trained model on your dataset. This involves freezing some layers and training the remaining layers on your dataset.

4. Data augmentation: Apply data augmentation techniques such as random cropping, flipping, and rotation to the training set to increase the diversity of the dataset.

5. Compile the model: Compile the model by choosing an optimizer, setting the loss function, and selecting the evaluation metric.

6. Train the model: Train the model on the training set and monitor the performance on the validation set. You can use early stopping to prevent overfitting.

7. Evaluate the model: Evaluate the performance of the model on the test set. Calculate metrics such as accuracy, precision, recall, and F1-score.

8. Predict using the model: Use the trained model to predict on new data.

3.2.5. Task 5: Analyze the results

1. Visualize the results: Visualization can help you to better understand the model's performance. For example, you can plot the confusion matrix to see how many true positives, true negatives, false positives, and false negatives the model is making. You can also plot the ROC curve to see how well the model is able to separate the positive and negative classes.

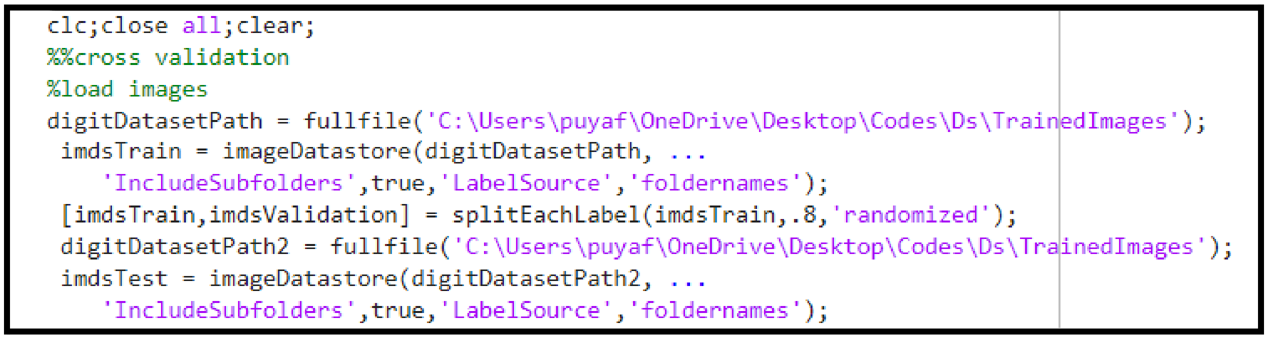
2. Identify areas for improvement: Based on the results of your analysis, identify areas where the model can be improved. For example, if the model is making a lot of false positive predictions, you may need to adjust the decision threshold. If the accuracy is low, you may need to collect more data or fine-tune the hyperparameters of the model.

3. Iterate: Finally, iterate on the model by making improvements based on the results of your analysis. Repeat the process until you are satisfied with the performance of the model on the validation or test data.

3.3 Project Execution

3.3.1. Task 1: Implement DCNN Architecture on dataset

1. Load the image data using the imageDatastore function. The data is organized into subfolders, with each subfolder containing images of a specific class.

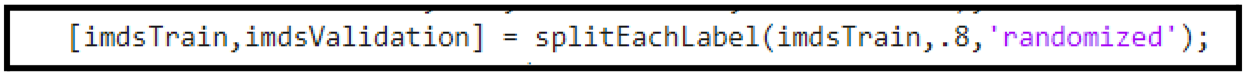


```
clc;close all;clear;
%%cross validation
%load images
digitDatasetPath = fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
 imdsTrain = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');
 [imdsTrain,imdsValidation] = splitEachLabel(imdsTrain,.8,'randomized');
 digitDatasetPath2 = fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
 imdsTest = imageDatastore(digitDatasetPath2, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');
```

Figure 3.3.1: Load images

This code imports a dataset of images from the specified directory and creates an imageDatastore object that stores the images and their corresponding labels. The 'IncludeSubfolders' option specifies whether or not to include images from subfolders in the dataset, and the 'LabelSource' option specifies that the folder names will be used as the labels for the images.

2. Split the data into training and validation sets using the splitEachLabel function. In this case, 80% of the data is used for training, and 20% is used for validation.

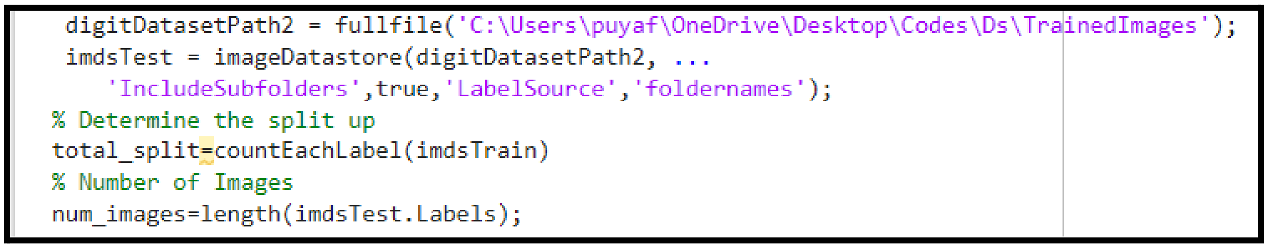


```
[imdsTrain,imdsValidation] = splitEachLabel(imdsTrain,.8,'randomized');
```

Figure 3.3.2: Image splitting

This code splits the training dataset into two sets - one for training and one for validation. The 'splitEachLabel' function is used to split the dataset, where the second argument (.8) specifies the percentage of images to include in the training set, and the third argument ('randomized') specifies that the split should be randomized.

3. Load the test data using the imageDatastore function. This data will be used to evaluate the trained network.

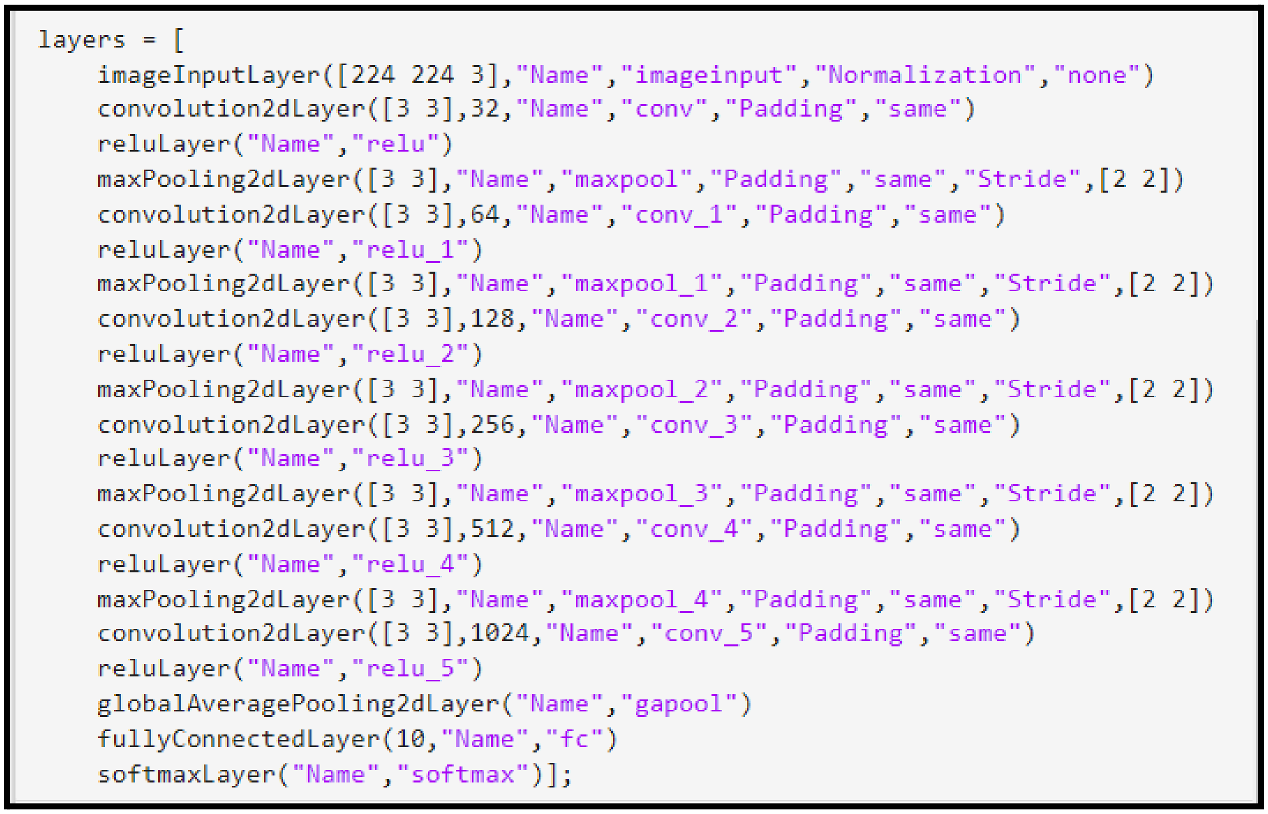


```
 digitDatasetPath2 = fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
 imdsTest = imageDatastore(digitDatasetPath2, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');
% Determine the split up
total_split=countEachLabel(imdsTrain)
% Number of Images
num_images=length(imdsTest.Labels);
```

Figure 3.3.3: Load the test data

This code imports another dataset of images from a different directory and creates an imageDatastore object that stores the images and their corresponding labels.

4. Define the architecture of the DCNN using a layer graph. The architecture consists of multiple convolutional layers, each followed by a ReLU activation function and a max pooling layer. The final layers are a global average pooling layer, a fully connected layer, a softmax layer, and a classification layer.



```
layers = [
    imageInputLayer([224 224 3],"Name","imageinput","Normalization","none")
    convolution2dLayer([3 3],32,"Name","conv","Padding","same")
    reluLayer("Name","relu")
    maxPooling2dLayer([3 3],"Name","maxpool","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],64,"Name","conv_1","Padding","same")
    reluLayer("Name","relu_1")
    maxPooling2dLayer([3 3],"Name","maxpool_1","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],128,"Name","conv_2","Padding","same")
    reluLayer("Name","relu_2")
    maxPooling2dLayer([3 3],"Name","maxpool_2","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],256,"Name","conv_3","Padding","same")
    reluLayer("Name","relu_3")
    maxPooling2dLayer([3 3],"Name","maxpool_3","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],512,"Name","conv_4","Padding","same")
    reluLayer("Name","relu_4")
    maxPooling2dLayer([3 3],"Name","maxpool_4","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],1024,"Name","conv_5","Padding","same")
    reluLayer("Name","relu_5")
    globalAveragePooling2dLayer("Name","gapool")
    fullyConnectedLayer(10,"Name","fc")
    softmaxLayer("Name","softmax")];
```

Figure 3.3.4: DCNN Layers

This is a code block that defines a convolutional neural network (CNN) in MATLAB using the Deep Learning Toolbox. The layers in this network are defined sequentially, with each layer being added to the list of layers.

The input layer is an image input layer with dimensions of 224x224x3, indicating that the network will take in RGB images of size 224x224. The normalization parameter is set to "none", which means that the input data is not normalized.

This is a code block that defines a convolutional neural network (CNN) in MATLAB using the Deep Learning Toolbox. The layers in this network are defined sequentially, with each layer being added to the list of layers.

The input layer is an image input layer with dimensions of 224x224x3, indicating that the network will take in RGB images of size 224x224. The normalization parameter is set to "none", which means that the input data is not normalized.

The next layers are convolutional layers, which apply a set of learnable filters to the input image to extract features. The first convolutional layer has 32 filters with a size of 3x3. The padding parameter is set to "same", which means that the output size of the layer will be the same as the input size.

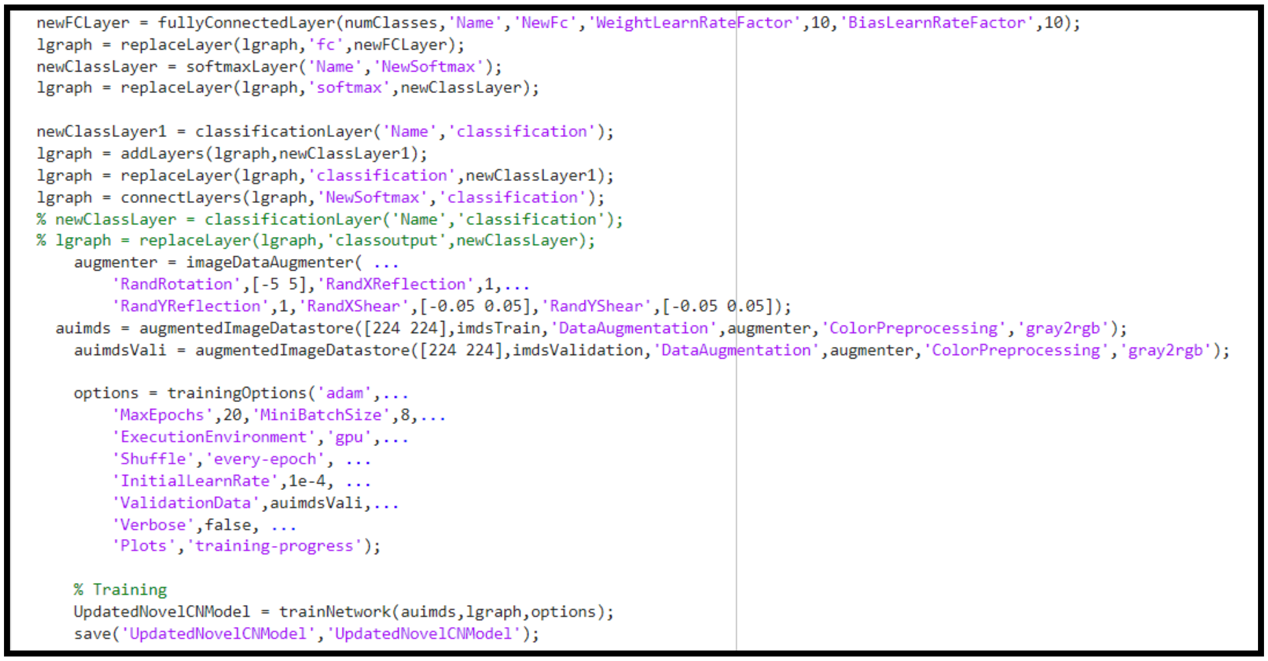
A rectified linear unit (ReLU) layer follows each convolutional layer. ReLU is an activation function that applies the function f(x) = max(0,x) to the output of the convolutional layer.

Max pooling layers follow each ReLU layer. Max pooling reduces the spatial dimensions of the feature maps, which helps to reduce the computational complexity of the network. The size of the pooling window is 3x3, and the padding and stride parameters are set to "same" and [2 2], respectively.

The last few layers are fully connected layers, which connect all the neurons from the previous layer to the current layer. The final layer is a softmax layer, which normalizes the output of the network to produce probabilities for each of the 10 output classes.

5. Replace the final fully connected layer and softmax layer with new layers that are tailored to the number of classes in the dataset. This is done using the replaceLayer function.

6. Define an imageDataAugmenter object to perform data augmentation during training. This includes random rotations, reflections, shears, and grayscale-to-RGB color conversion.
    a. Create an augmentedImageDatastore object to handle the augmented data during training and validation.
    b. Define the training options, including the optimization algorithm, number of epochs, batch size, learning rate, and validation data.
    c. Train the network using the trainNetwork function. The resulting trained network is saved to a file.
    d. During testing, load the trained network from the saved file and use it to classify the test images.



```
newFCLayer = fullyConnectedLayer(numClasses,'Name','NewFc','WeightLearnRateFactor',10,'BiasLearnRateFactor',10);
lgraph = replaceLayer(lgraph,'fc',newFCLayer);
newClassLayer = softmaxLayer('Name','NewSoftmax');
lgraph = replaceLayer(lgraph,'softmax',newClassLayer);

newClassLayer1 = classificationLayer('Name','classification');
lgraph = addLayers(lgraph,newClassLayer1);
lgraph = replaceLayer(lgraph,'classification',newClassLayer1);
lgraph = connectLayers(lgraph,'NewSoftmax','classification');
% newClassLayer = classificationLayer('Name','classification');
% lgraph = replaceLayer(lgraph,'classoutput',newClassLayer);
    augmenter = imageDataAugmenter( ...
        'RandRotation',[-5 5],'RandXReflection',1,...
        'RandYReflection',1,'RandXShear',[-0.05 0.05],'RandYShear',[-0.05 0.05]);
  auimds = augmentedImageDatastore([224 224],imdsTrain,'DataAugmentation',augmenter,'ColorPreprocessing','gray2rgb');
    auimdsVali = augmentedImageDatastore([224 224],imdsValidation,'DataAugmentation',augmenter,'ColorPreprocessing','gray2rgb');

    options = trainingOptions('adam',...
        'MaxEpochs',20,'MiniBatchSize',8,...
        'ExecutionEnvironment','gpu',...
        'Shuffle','every-epoch', ...
        'InitialLearnRate',1e-4, ...
        'ValidationData',auimdsVali,...
        'Verbose',false, ...
        'Plots','training-progress');

    % Training
    UpdatedNovelCNModel = trainNetwork(auimds,lgraph,options);
    save('UpdatedNovelCNModel','UpdatedNovelCNModel');
```

Figure 3.3.5: Fine-tuning

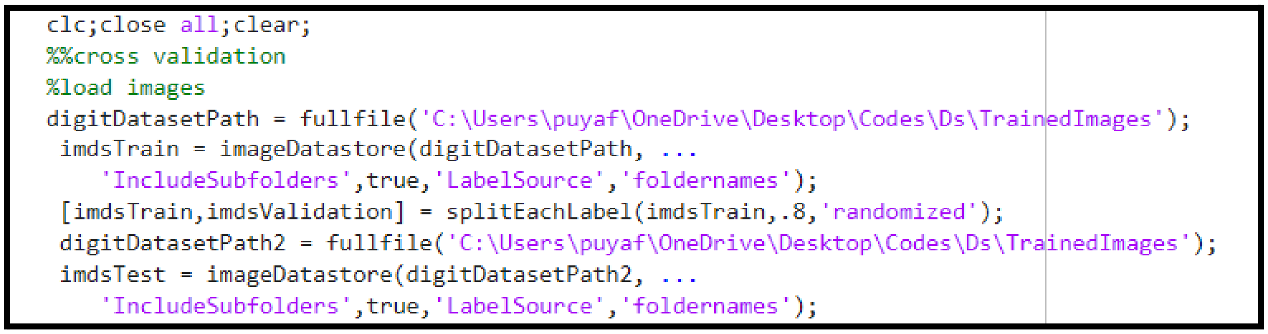
This code performs fine-tuning of a pre-trained convolutional neural network (CNN) for a classification task. Here's a breakdown of what the code does:

- Load the pre-trained CNN from a MAT-file 'updatedNet.mat' into a variable 'net'.
- Create a layer graph 'lgraph' from the loaded CNN using the 'layerGraph' function.
- Clear the 'net' variable to free up memory.
- Determine the number of classes in the training data by counting the number of unique labels in 'imdsTrain.Labels'.
- Add a new fully connected layer with 'numClasses' output neurons to the layer graph, named 'NewFc'.
- Replace the original fully connected layer in the layer graph with the newly added one.
- Add a new softmax layer to the layer graph, named 'NewSoftmax'.
- Replace the original softmax layer in the layer graph with the newly added one.
- Add a new classification layer to the layer graph, named 'classification', to specify the output format of the network.
- Connect the output of the 'NewSoftmax' layer to the 'classification' layer in the layer graph.
- Define an image data augmenter with a variety of augmentations such as rotation, reflection, and shearing.
- Create an augmented image datastore 'auimds' from the training data using the augmenter and specifying the desired image size of [224 224].
- Create an augmented image datastore 'auimdsVali' from the validation data using the same augmenter and image size.
- Define training options such as the optimizer, maximum number of epochs, mini-batch size, execution environment, learning rate, validation data, and verbosity.

- Train the updated CNN using the training data 'auimds', the layer graph 'lgraph', and the specified training options.
- Save the trained network as a MAT-file 'UpdatedNovelCNModel.mat' in the current directory.

3.3.2. Task 2: Implement Darknet19 Architecture on dataset

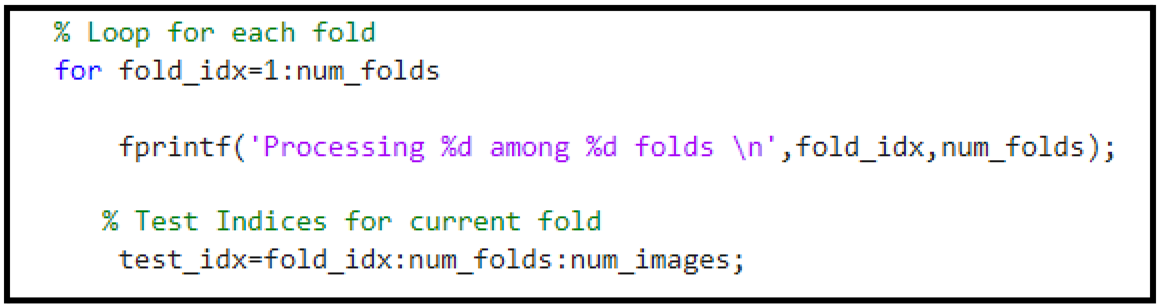
1. Loaded the images from the directory using imageDatastore function and split them into training and validation sets using the splitEachLabel function. Loaded the test images from the directory using imageDatastore function.



```
clc;close all;clear;
%%cross validation
%load images
digitDatasetPath = fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
 imdsTrain = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');
 [imdsTrain,imdsValidation] = splitEachLabel(imdsTrain,.8,'randomized');
 digitDatasetPath2 = fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
 imdsTest = imageDatastore(digitDatasetPath2, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');
```

Figure 3.3.6: Loading images

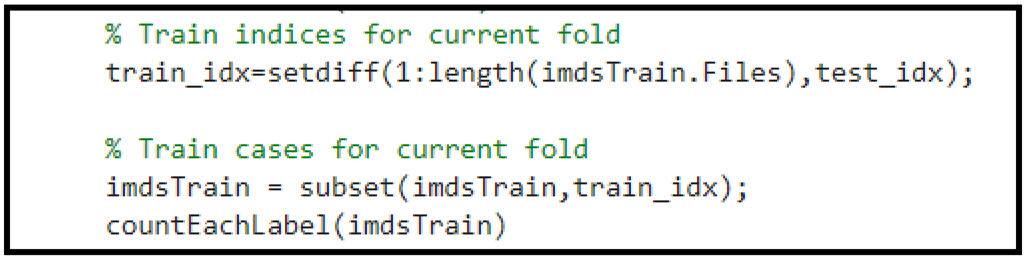
2. Loop for each fold. For each fold, determine the test indices using the fold index and the number of folds. Selected the test cases for the current fold using the subset function. Determine the train indices for the current fold using setdiff function.



```
% Loop for each fold
for fold_idx=1:num_folds

    fprintf('Processing %d among %d folds \n',fold_idx,num_folds);

   % Test Indices for current fold
    test_idx=fold_idx:num_folds:num_images;
```

Figure 3.3.7: Train indices

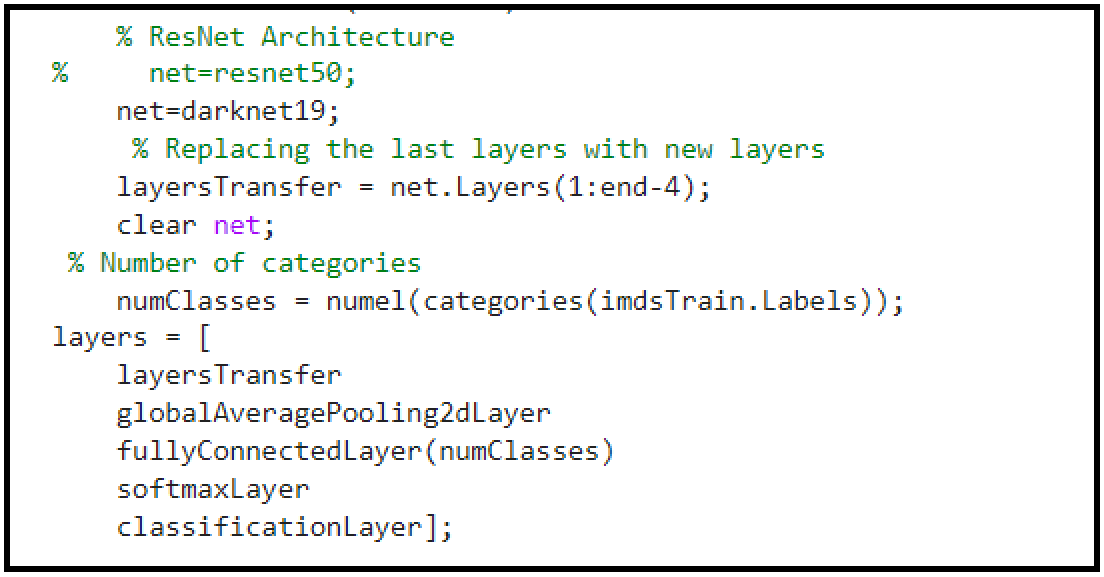
3. Selected the train cases for the current fold using the subset function.



```
% Train indices for current fold
train_idx=setdiff(1:length(imdsTrain.Files),test_idx);

% Train cases for current fold
imdsTrain = subset(imdsTrain,train_idx);
countEachLabel(imdsTrain)
```

Figure 3.3.8: Train cases

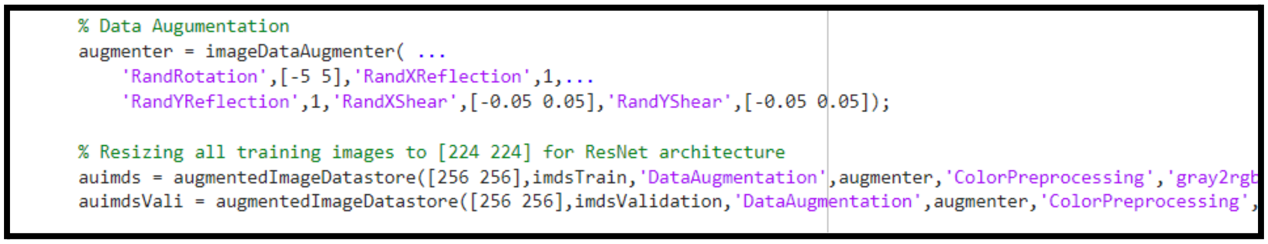
4. Loaded the Darknet19 architecture using the darknet19 function. Replaced the last layers of the architecture with new layers that match the number of categories using the globalAveragePooling2dLayer, fullyConnectedLayer, softmaxLayer and classificationLayer functions.



```
    % ResNet Architecture
%     net=resnet50;
    net=darknet19;
     % Replacing the last layers with new layers
    layersTransfer = net.Layers(1:end-4);
    clear net;
 % Number of categories
    numClasses = numel(categories(imdsTrain.Labels));
layers = [
    layersTransfer
    globalAveragePooling2dLayer
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

Figure 3.3.9: darknet Architecture

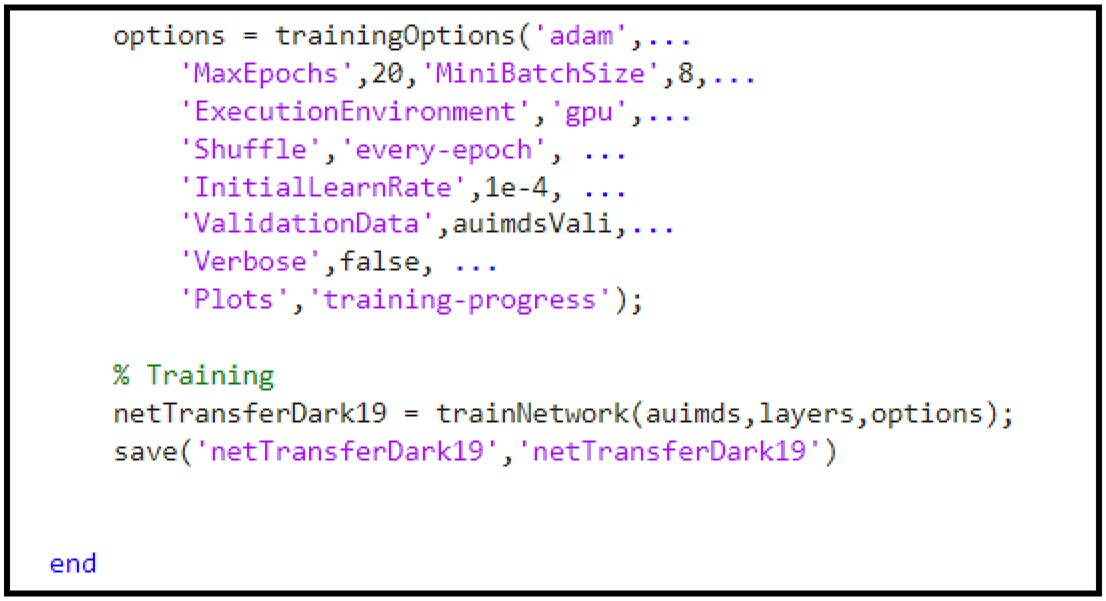
5. Defined the data augmentation using the imageDataAugmenter function. Resized all training and validation images to [256 256] for Darknet19 architecture.



```
% Data Augumentation
augmenter = imageDataAugmenter( ...
    'RandRotation',[-5 5],'RandXReflection',1,...
    'RandYReflection',1,'RandXShear',[-0.05 0.05],'RandYShear',[-0.05 0.05]);

% Resizing all training images to [224 224] for ResNet architecture
auimds = augmentedImageDatastore([256 256],imdsTrain,'DataAugmentation',augmenter,'ColorPreprocessing','gray2rgb
auimdsVali = augmentedImageDatastore([256 256],imdsValidation,'DataAugmentation',augmenter,'ColorPreprocessing',
```

Figure 3.3.10: Data augmentation

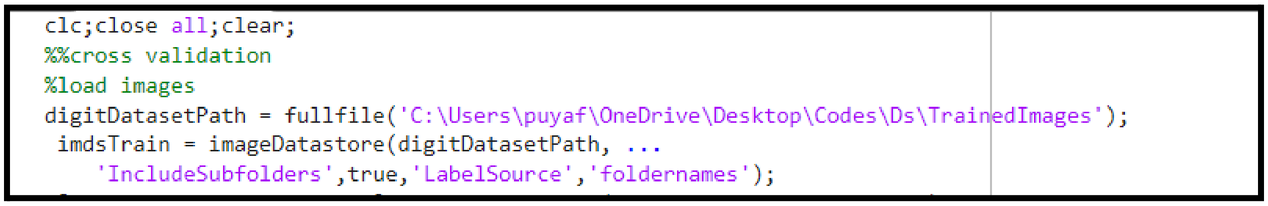
6. Defined the options for training using the trainingOptions function. Here, you set the maximum number of epochs to 20, the mini-batch size to 8, the execution environment to GPU, the shuffle option to 'every-epoch', the initial learning rate to 1e-4, and specified the validation data and whether to display training progress or not.



```
    options = trainingOptions('adam',...
        'MaxEpochs',20,'MiniBatchSize',8,...
        'ExecutionEnvironment','gpu',...
        'Shuffle','every-epoch', ...
        'InitialLearnRate',1e-4, ...
        'ValidationData',auimdsVali,...
        'Verbose',false, ...
        'Plots','training-progress');

    % Training
    netTransferDark19 = trainNetwork(auimds,layers,options);
    save('netTransferDark19','netTransferDark19')


end
```

Figure 3.3.11: Options

7. Trai



```
clc;close all;clear;
%%cross validation
%load images
digitDatasetPath = fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
 imdsTrain = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');
```

ned the network using the trainNetwork function.

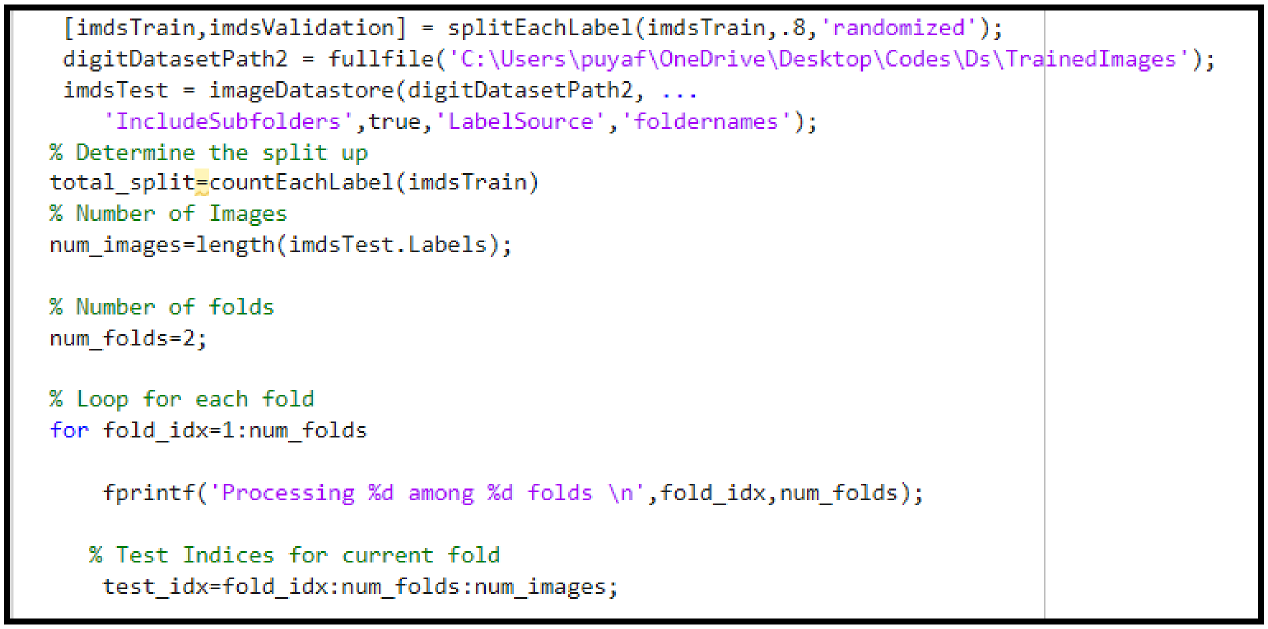
8. Saved the trained network to a file named 'netTransferDark19' using the save function.

3.3.3. Task 3: Implement ResNet-50 Architecture on dataset

1. Set the current folder to the directory where you have saved the code file. Clear any existing variables, figures, and command window. Load the digit images for training and validation using the "imagedatastore" function.

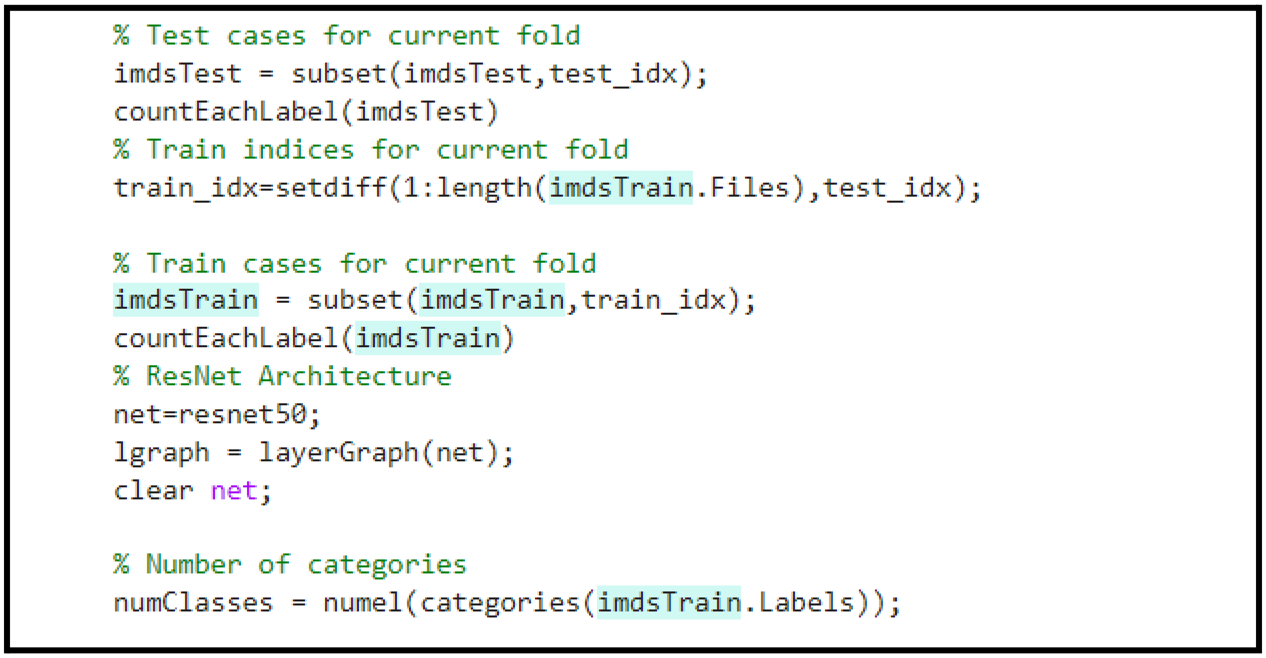
Figure 3.3.12: Initialize

2. Split the training images into 80% for training and 20% for validation using the "splitEachLabel" function. Load the test images using the "imagedatastore" function. Count the number of images in each label of the training data using the "countEachLabel" function. Set the number of images and folds for cross-validation.



```
 [imdsTrain,imdsValidation] = splitEachLabel(imdsTrain,.8,'randomized');
 digitDatasetPath2 = fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
 imdsTest = imageDatastore(digitDatasetPath2, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');
% Determine the split up
total_split=countEachLabel(imdsTrain)
% Number of Images
num_images=length(imdsTest.Labels);

% Number of folds
num_folds=2;

% Loop for each fold
for fold_idx=1:num_folds

    fprintf('Processing %d among %d folds \n',fold_idx,num_folds);

   % Test Indices for current fold
    test_idx=fold_idx:num_folds:num_images;
```

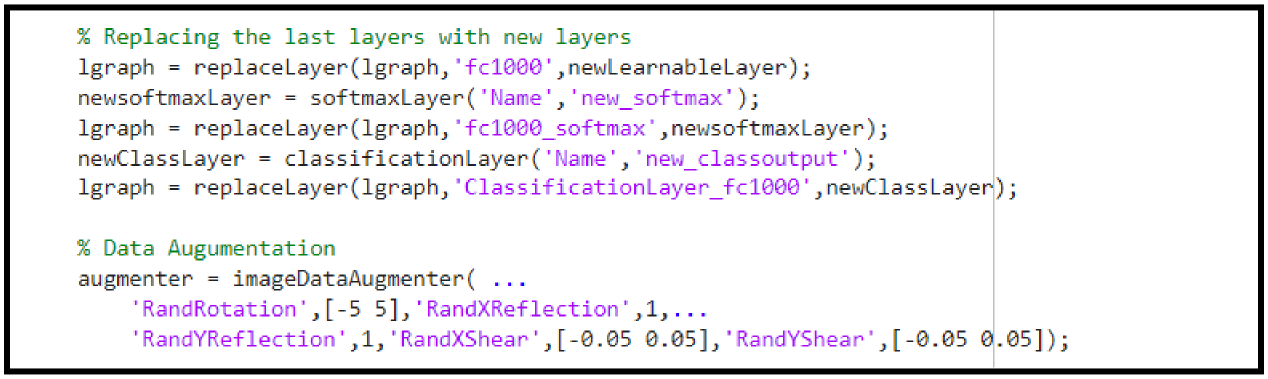
Figure 3.3.13: Split data



```
    % Test cases for current fold
    imdsTest = subset(imdsTest,test_idx);
    countEachLabel(imdsTest)
    % Train indices for current fold
    train_idx=setdiff(1:length(imdsTrain.Files),test_idx);

    % Train cases for current fold
    imdsTrain = subset(imdsTrain,train_idx);
    countEachLabel(imdsTrain)
    % ResNet Architecture
    net=resnet50;
    lgraph = layerGraph(net);
    clear net;

    % Number of categories
    numClasses = numel(categories(imdsTrain.Labels));
```

Figure 3.3.14: Train cases

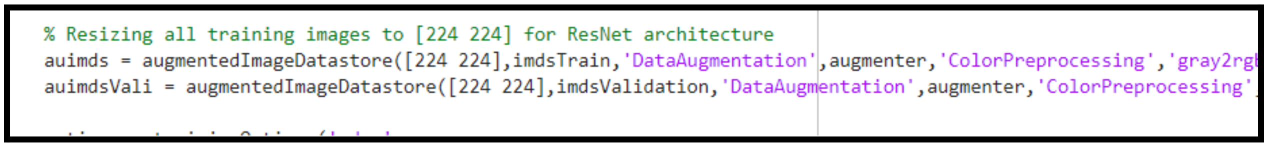
3. Loop through each fold and get the test and train indices using the "subset" and "setdiff" functions, respectively.Define a ResNet-50 architecture using the "resnet50" function. Replace the last layers of the ResNet-50 architecture with new fully connected, softmax, and classification layers using the "replaceLayer" function.



```
% Replacing the last layers with new layers
lgraph = replaceLayer(lgraph,'fc1000',newLearnableLayer);
newsoftmaxLayer = softmaxLayer('Name','new_softmax');
lgraph = replaceLayer(lgraph,'fc1000_softmax',newsoftmaxLayer);
newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,'ClassificationLayer_fc1000',newClassLayer);

% Data Augumentation
augmenter = imageDataAugmenter( ...
    'RandRotation',[-5 5],'RandXReflection',1,...
    'RandYReflection',1,'RandXShear',[-0.05 0.05],'RandYShear',[-0.05 0.05]);
```

Figure 3.3.15: Data augmentation

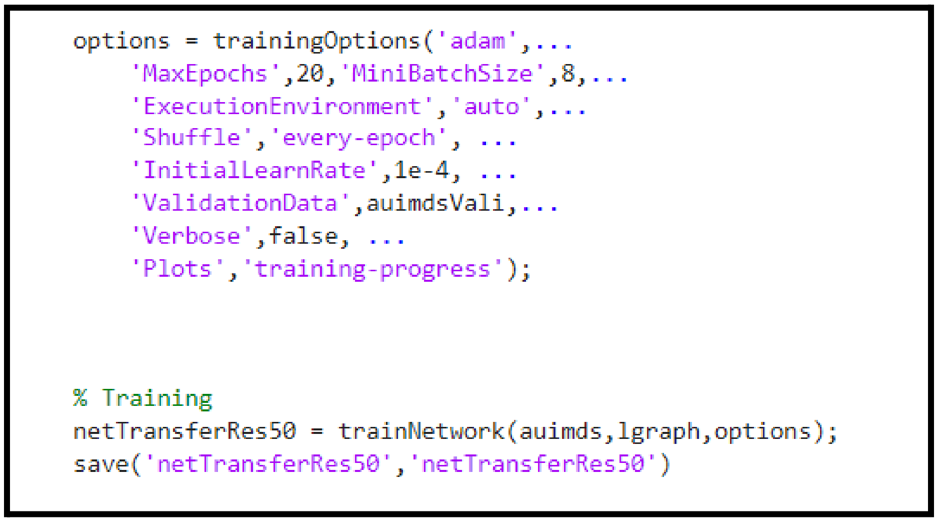
4. Define an image data augmenter using the "imageDataAugmenter" function.Resize all the training and validation images to [224 224] using the "augmentedImageDatastore" function.



```
% Resizing all training images to [224 224] for ResNet architecture
auimds = augmentedImageDatastore([224 224],imdsTrain,'DataAugmentation',augmenter,'ColorPreprocessing','gray2rg
auimdsVali = augmentedImageDatastore([224 224],imdsValidation,'DataAugmentation',augmenter,'ColorPreprocessing'
```

Figure 3.3.16: Resize the image

5. Define the training options for the transfer learning using the "trainingOptions" function. Train the network using the "trainNetwork" function and save the trained network for each fold.

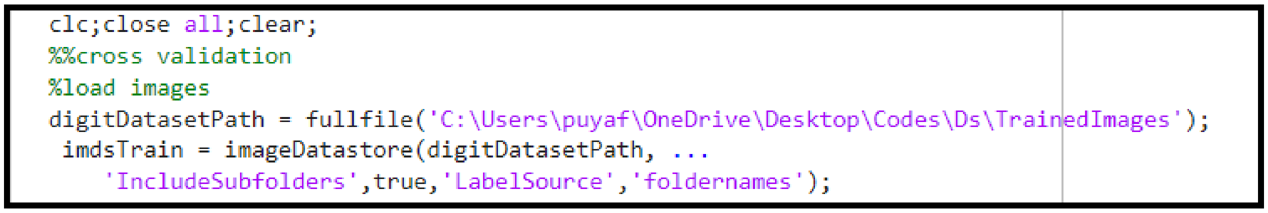


```
options = trainingOptions('adam',...
    'MaxEpochs',20,'MiniBatchSize',8,...
    'ExecutionEnvironment','auto',...
    'Shuffle','every-epoch', ...
    'InitialLearnRate',1e-4, ...
    'ValidationData',auimdsVali,...
    'Verbose',false, ...
    'Plots','training-progress');



% Training
netTransferRes50 = trainNetwork(auimds,lgraph,options);
save('netTransferRes50','netTransferRes50')
```

Figure 3.3.17: Options and training

3.3.4. Task 4: Implement Efficient B0 Architecture on dataset

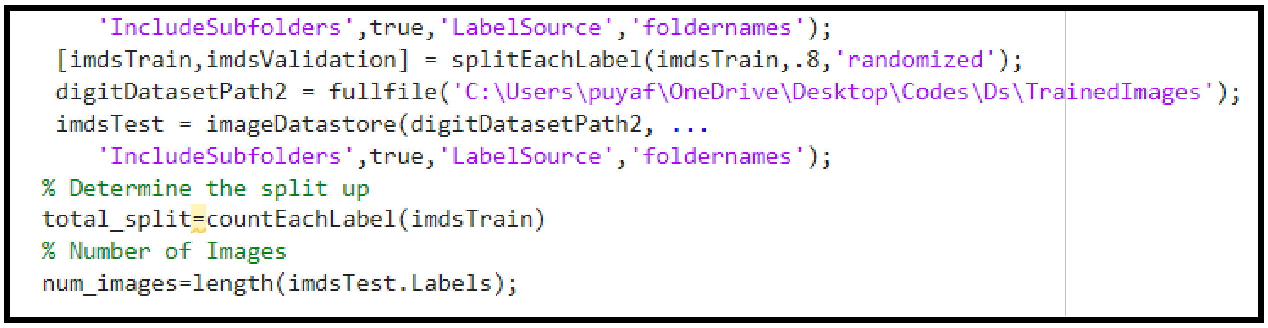
1. Set the current folder to the directory where you have saved the code file. Clear any existing variables, figures, and command window. Load the digit images for training and validation using the "imagedatastore" function.



```
clc;close all;clear;
%%cross validation
%load images
digitDatasetPath = fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
 imdsTrain = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');
```

Figure 3.3.18: Set directory

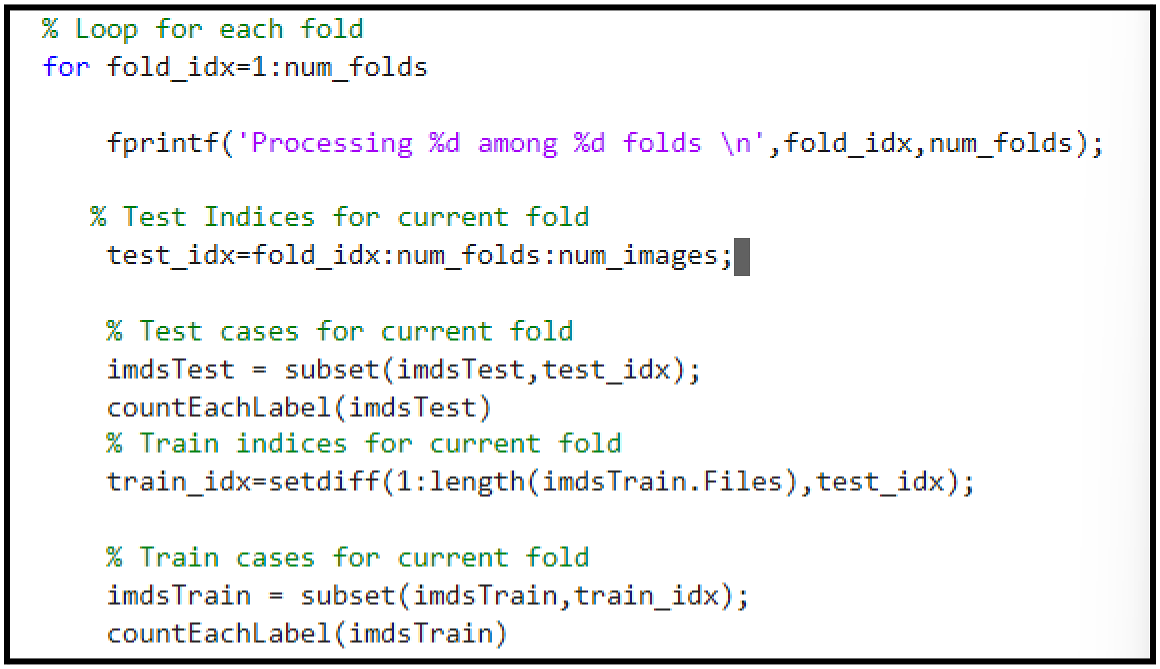
2. Split the training images into 80% for training and 20% for validation using the "splitEachLabel" function. Load the test images using the "imagedatastore" function. Count the number of images in each label of the training data using the "countEachLabel" function. Set the number of images and folds for cross-validation.



```
    'IncludeSubfolders',true,'LabelSource','foldernames');
 [imdsTrain,imdsValidation] = splitEachLabel(imdsTrain,.8,'randomized');
 digitDatasetPath2 = fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
 imdsTest = imageDatastore(digitDatasetPath2, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');
% Determine the split up
total_split=countEachLabel(imdsTrain)
% Number of Images
num_images=length(imdsTest.Labels);
```

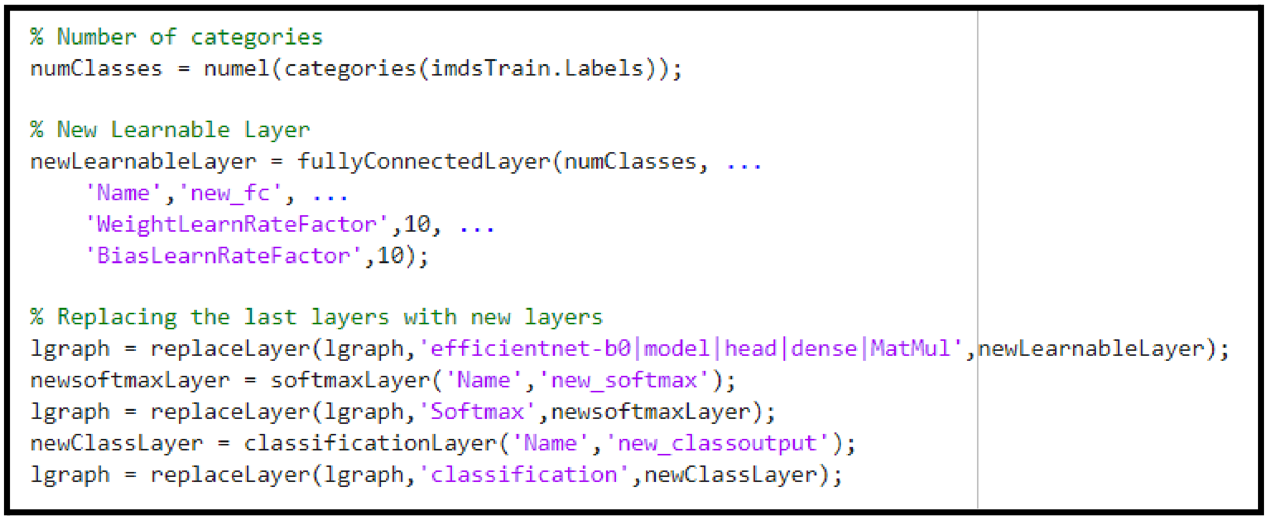
Figure 3.3.19: Split data

3. Loop through each fold and get the test and train indices using the "subset" and "setdiff" functions, respectively.Define a efficientnetb0 architecture using the "efficientnetb0" function. Replace the last layers of the efficientnetb0 architecture with new fully connected, softmax, and classification layers using the "replaceLayer" function.



```
% Loop for each fold
for fold_idx=1:num_folds

    fprintf('Processing %d among %d folds \n',fold_idx,num_folds);

   % Test Indices for current fold
    test_idx=fold_idx:num_folds:num_images;

    % Test cases for current fold
    imdsTest = subset(imdsTest,test_idx);
    countEachLabel(imdsTest)
    % Train indices for current fold
    train_idx=setdiff(1:length(imdsTrain.Files),test_idx);

    % Train cases for current fold
    imdsTrain = subset(imdsTrain,train_idx);
    countEachLabel(imdsTrain)
```

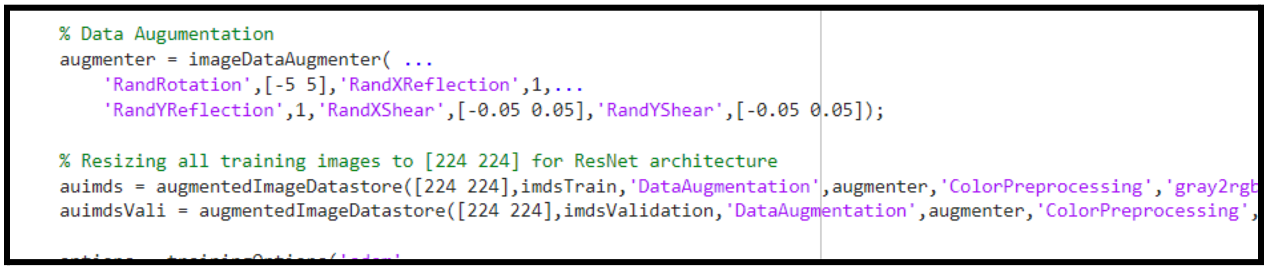
Figure 3.3.20: Loop for each folder



```
% Number of categories
numClasses = numel(categories(imdsTrain.Labels));

% New Learnable Layer
newLearnableLayer = fullyConnectedLayer(numClasses, ...
    'Name','new_fc', ...
    'WeightLearnRateFactor',10, ...
    'BiasLearnRateFactor',10);

% Replacing the last layers with new layers
lgraph = replaceLayer(lgraph,'efficientnet-b0|model|head|dense|MatMul',newLearnableLayer);
newsoftmaxLayer = softmaxLayer('Name','new_softmax');
lgraph = replaceLayer(lgraph,'Softmax',newsoftmaxLayer);
newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,'classification',newClassLayer);
```

Figure 3.3.20: Replacing last layer

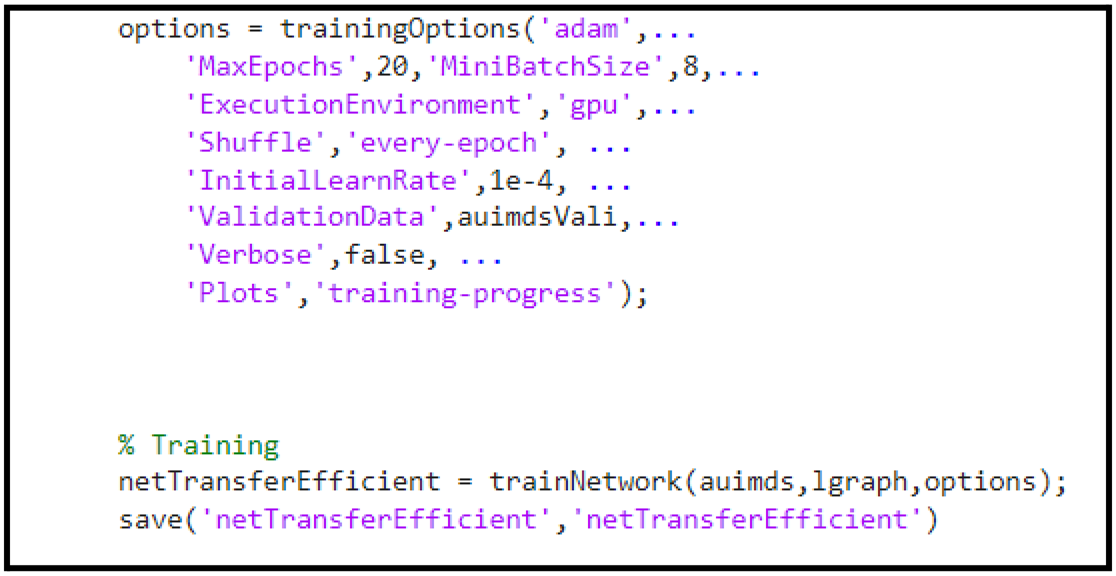
4. Define an image data augmenter using the "imageDataAugmenter" function.Resize all the training and validation images to [224 224] using the "augmentedImageDatastore" function.



```
% Data Augumentation
augmenter = imageDataAugmenter( ...
    'RandRotation',[-5 5],'RandXReflection',1,...
    'RandYReflection',1,'RandXShear',[-0.05 0.05],'RandYShear',[-0.05 0.05]);

% Resizing all training images to [224 224] for ResNet architecture
auimds = augmentedImageDatastore([224 224],imdsTrain,'DataAugmentation',augmenter,'ColorPreprocessing','gray2rgb
auimdsVali = augmentedImageDatastore([224 224],imdsValidation,'DataAugmentation',augmenter,'ColorPreprocessing',

options = trainingOptions('adam',
```

Figure 3.3.21: Data augmentation and resize

5. Define the training options for the transfer learning using the "trainingOptions" function. Train the network using the "trainNetwork" function and save the trained network for each fold.



```
options = trainingOptions('adam',...
    'MaxEpochs',20,'MiniBatchSize',8,...
    'ExecutionEnvironment','gpu',...
    'Shuffle','every-epoch', ...
    'InitialLearnRate',1e-4, ...
    'ValidationData',auimdsVali,...
    'Verbose',false, ...
    'Plots','training-progress');



% Training
netTransferEfficient = trainNetwork(auimds,lgraph,options);
save('netTransferEfficient','netTransferEfficient')
```

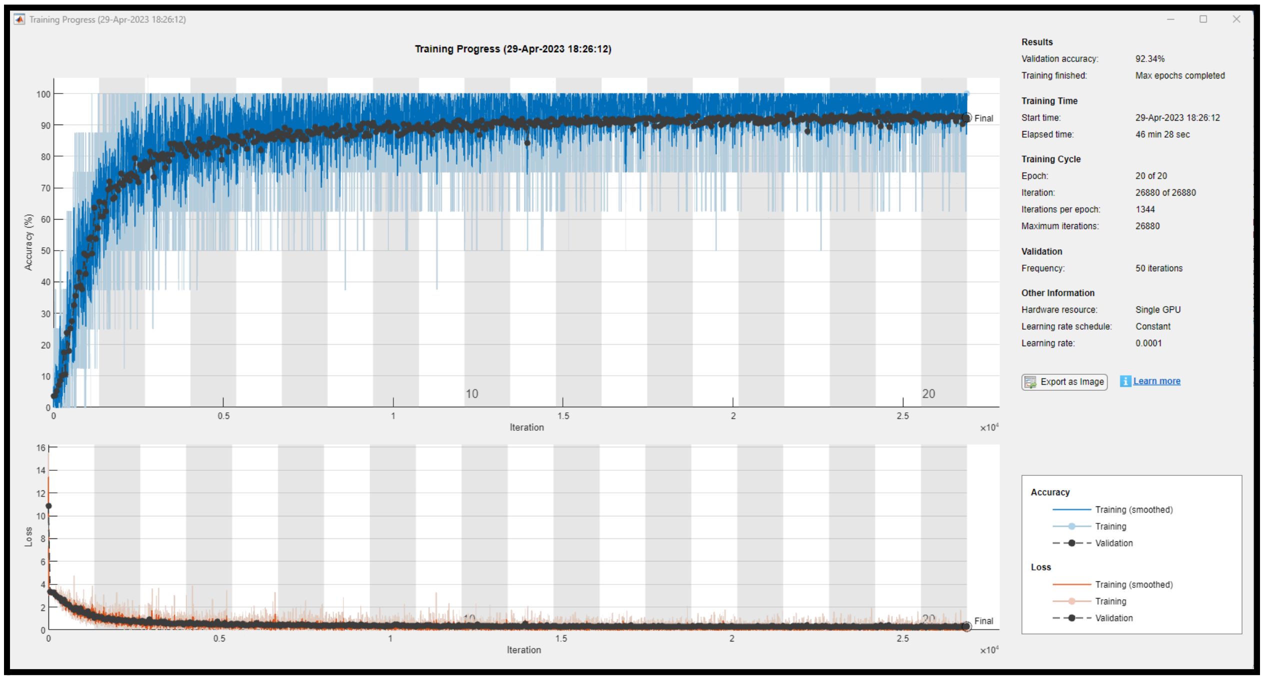
Figure 3.3.22: Training

4. ANALYSIS

4.1 Experimental Results

Task 1: Implement DCNN Architecture on the dataset

The training is performed using the Adam optimizer with a mini-batch size of 8 and a maximum of 20 epochs. The execution is done on a GPU to speed up the training process. Finally, the trained model is saved for future use. The objective of this process is to improve the accuracy of the pre-trained model in recognizing images in a specific domain by fine-tuning it to the new task.



Results

Validation accuracy:	92.34%
Training finished:	Max epochs completed

Training Time

Start time:	29-Apr-2023 18:26:12
Elapsed time:	46 min 28 sec

Training Cycle

Epoch:	20 of 20
Iteration:	26880 of 26880
Iterations per epoch:	1344
Maximum iterations:	26880

Validation

Frequency:	50 iterations

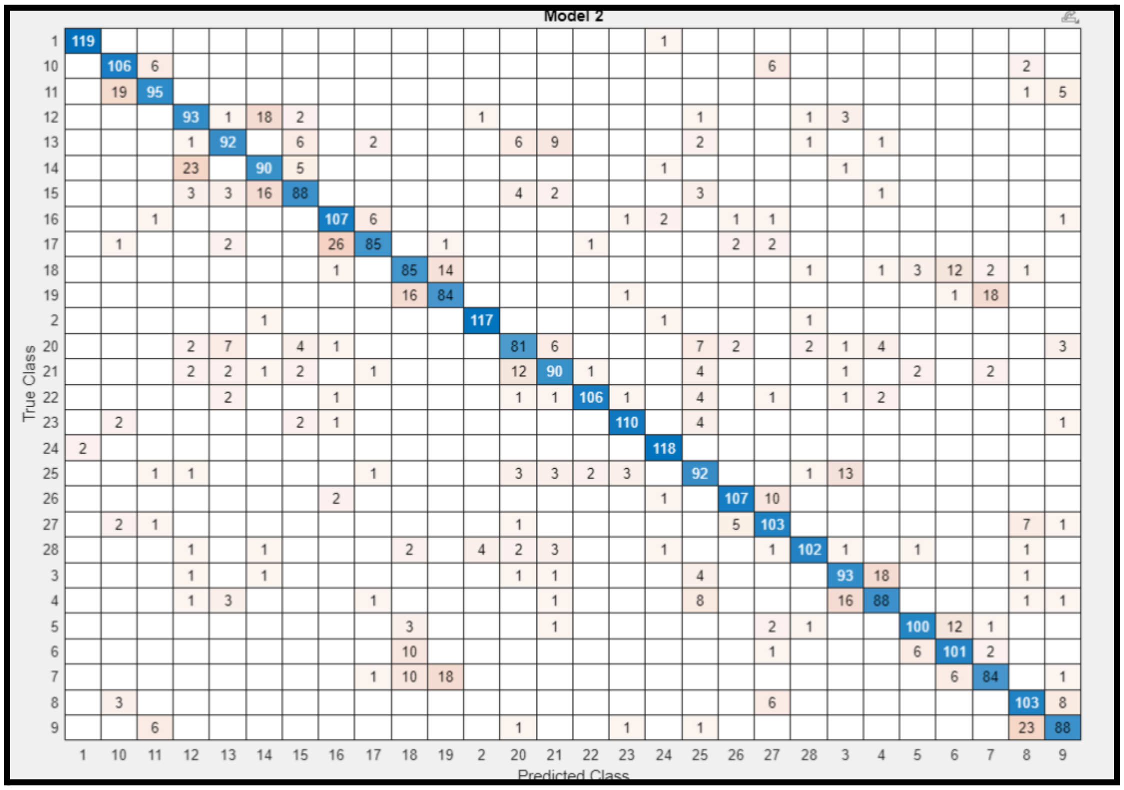
Other Information

Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.0001

Results:

Validation accuracy: 92.34%
Elapsed time: 46 min 28 sec
Epoch: 20 of 20
Max Iterations: 26880
Iterations per epoch: 1344
Frequency: 50 Iterations
Hardware resource: Single GPU
Learning rate: 0.0001

Confusion matrix for DCNN:



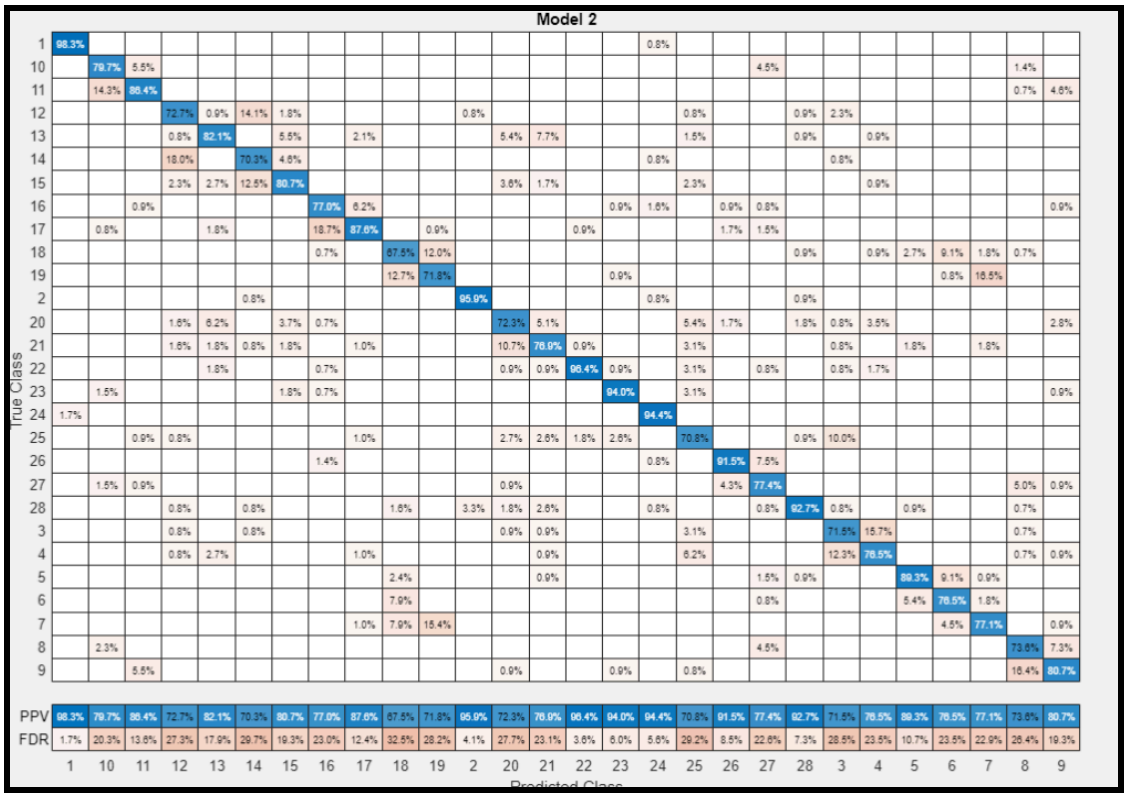
Model 2

	1	10	11	12	13	14	15	16	17	18	19	2	20	21	22	23	24	25	26	27	28	3	4	5	6	7	8	9
1	119																1											
10		106	6																	6							2	
11		19	95																								1	5
12				93	1	18	2					1						1			1	3						
13				1	92		6		2				6	9				2			1		1					
14				23		90	5										1					1						
15				3	3	16	88						4	2				3					1					
16			1					107	6							1	2		1	1								1
17		1			2			26	85		1				1				2	2								
18								1		85	14										1		1	3	12	2	1	
19										16	84					1									1	18		
2						1						117					1				1							
20				2	7		4	1					81	6				7	2		2	1	4					3
21				2	2	1	2		1				12	90	1			4				1		2		2		
22					2			1					1	1	106	1		4		1		1	2					
23		2					2	1								110		4										1
24	2																118											
25			1	1					1				3	3	2	3		92			1	13						
26								2									1		107	10								
27		2	1										1						5	103							7	1
28				1		1				2		4	2	3			1			1	102	1		1			1	
3				1		1							1	1				4				93	18				1	
4				1	3				1					1				8				16	88				1	1
5										3				1						2	1			100	12	1		
6										10										1				6	101	2		
7									1	10	18														6	84		1
8		3																		6							103	8
9			6										1			1		1									23	88

True Class

Predicted Class

Confusion matrix with % success rate on each class:



Model 2

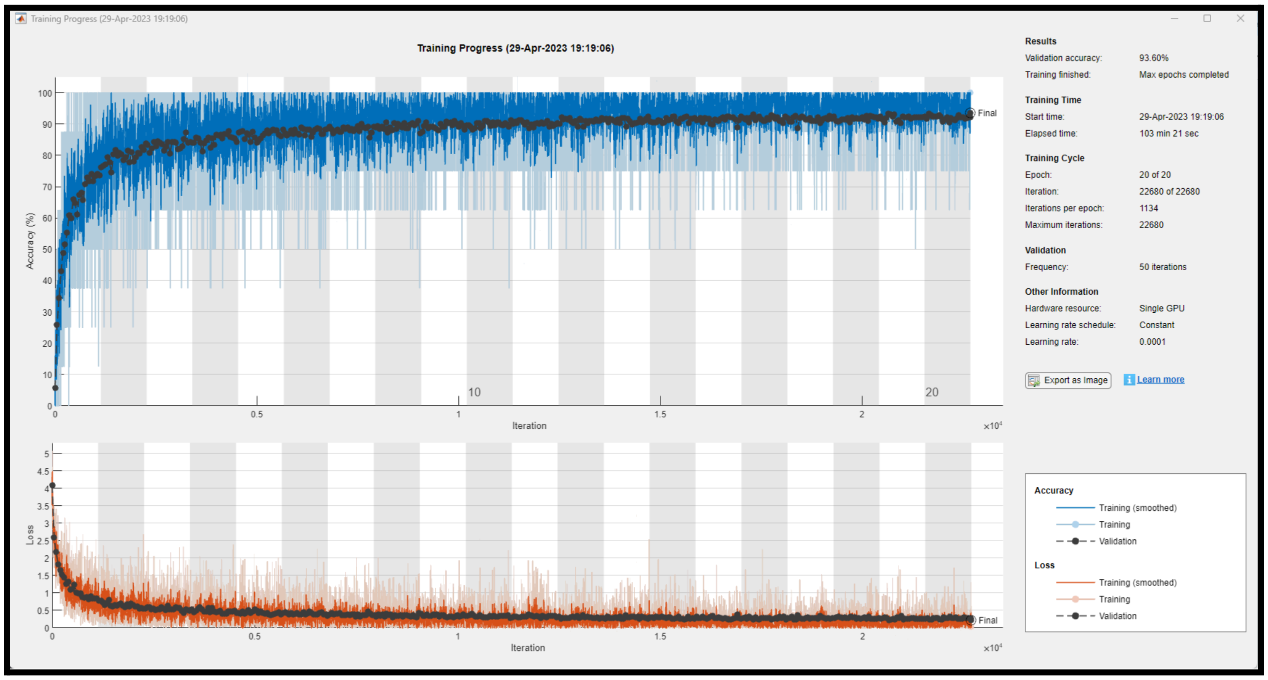
	1	10	11	12	13	14	15	16	17	18	19	2	20	21	22	23	24	25	26	27	28	3	4	5	6	7	8	9
1	98.3%																0.8%											
10		79.7%	5.5%																	4.5%							1.4%	
11		14.3%	86.4%																								0.7%	4.6%
12				72.7%	0.9%	14.1%	1.8%					0.8%						0.8%			0.9%	2.3%						
13				0.8%	82.1%		5.5%		2.1%				5.4%	7.7%				1.5%			0.9%		0.9%					
14				18.0%		70.3%	4.6%										0.8%					0.8%						
15				2.3%	2.7%	12.5%	80.7%						3.6%	1.7%				2.3%					0.9%					
16			0.9%					77.0%	6.2%							0.9%	1.6%		0.9%	0.8%								0.9%
17		0.8%			1.8%			18.7%	87.6%		0.9%				0.9%				1.7%	1.5%								
18								0.7%		67.5%	12.0%										0.9%		0.9%	2.7%	9.1%	1.8%	0.7%	
19										12.7%	71.8%					0.9%									0.8%	16.5%		
2						0.8%						95.9%					0.8%				0.9%							
20				1.6%	6.2%		3.7%	0.7%					72.3%	5.1%				5.4%	1.7%		1.8%	0.8%	3.5%					2.8%
21				1.6%	1.8%	0.8%	1.8%		1.0%				10.7%	76.9%	0.9%			3.1%				0.8%		1.8%		1.8%		
22					1.8%			0.7%					0.9%	0.9%	96.4%	0.9%		3.1%		0.8%		0.8%	1.7%					
23		1.5%					1.8%	0.7%								94.0%		3.1%										0.9%
24	1.7%																94.4%											
25			0.9%	0.8%					1.0%				2.7%	2.6%	1.8%	2.6%		70.8%			0.9%	10.0%						
26								1.4%									0.8%		91.5%	7.5%								
27		1.5%	0.9%										0.9%						4.3%	77.4%							5.0%	0.9%
28				0.8%		0.8%				1.6%		3.3%	1.8%	2.6%			0.8%			0.8%	92.7%	0.8%		0.9%			0.7%	
3				0.8%		0.8%							0.9%	0.9%				3.1%				71.5%	15.7%				0.7%	
4				0.8%	2.7%				1.0%					0.9%				6.2%				12.3%	76.5%				0.7%	0.9%
5										2.4%				0.9%						1.5%	0.9%			89.3%	9.1%	0.9%		
6										7.9%										0.8%				5.4%	76.5%	1.8%		
7									1.0%	7.9%	15.4%														4.5%	77.1%		0.9%
8		2.3%																		4.5%							73.6%	7.3%
9			5.5%										0.9%			0.9%		0.8%									16.4%	80.7%
PPV	98.3%	79.7%	86.4%	72.7%	82.1%	70.3%	80.7%	77.0%	87.6%	67.5%	71.8%	95.9%	72.3%	76.9%	96.4%	94.0%	94.4%	70.8%	91.5%	77.4%	92.7%	71.5%	76.5%	89.3%	76.5%	77.1%	73.6%	80.7%
FDR	1.7%	20.3%	13.6%	27.3%	17.9%	29.7%	19.3%	23.0%	12.4%	32.5%	28.2%	4.1%	27.7%	23.1%	3.6%	6.0%	5.6%	29.2%	8.5%	22.6%	7.3%	28.5%	23.5%	10.7%	23.5%	22.9%	26.4%	19.3%

True Class

Predicted Class

Task 2: Implement DarkNet-19 Architecture on dataset

The training is performed using the Adam optimizer with a mini-batch size of 8 and a maximum of 20 epochs. The execution is done on a GPU to speed up the training process. Finally, the trained model is saved for future use. The objective of this process is to improve the accuracy of the pre-trained model in recognizing images in a specific domain by fine-tuning it to the new task.



Results

Validation accuracy:	93.60%
Training finished:	Max epochs completed

Training Time

Start time:	29-Apr-2023 19:19:06
Elapsed time:	103 min 21 sec

Training Cycle

Epoch:	20 of 20
Iteration:	22680 of 22680
Iterations per epoch:	1134
Maximum iterations:	22680

Validation

Frequency:	50 iterations

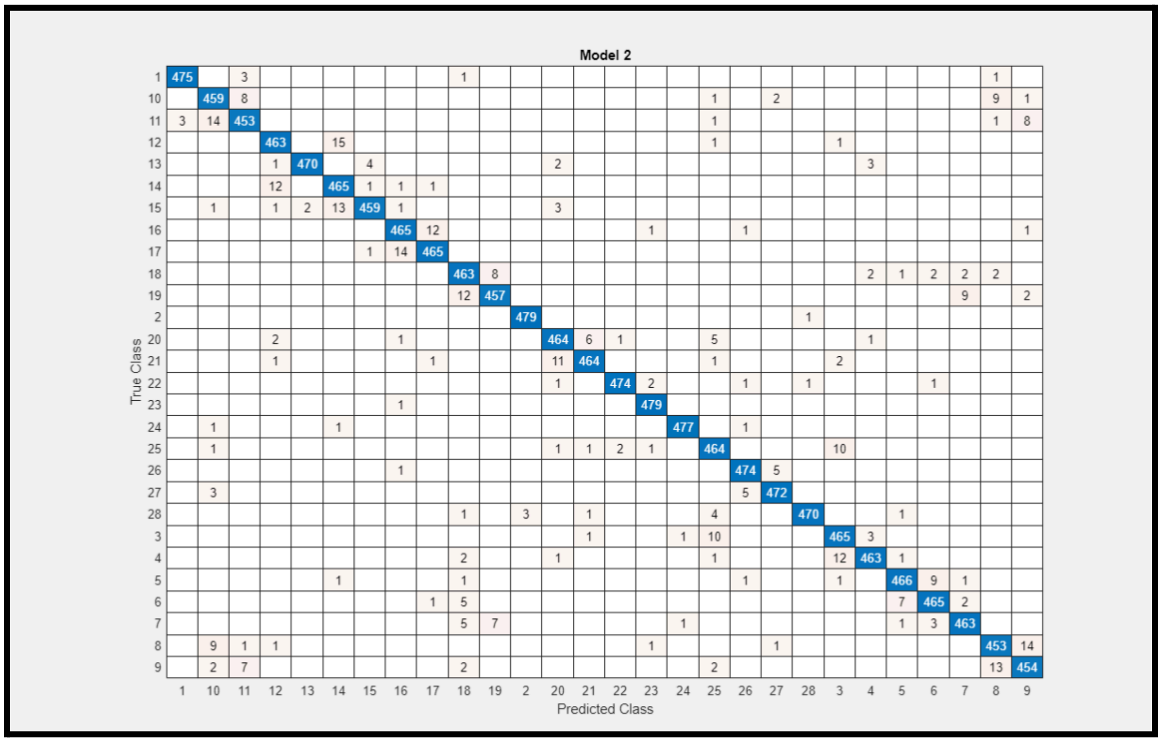
Other Information

Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.0001

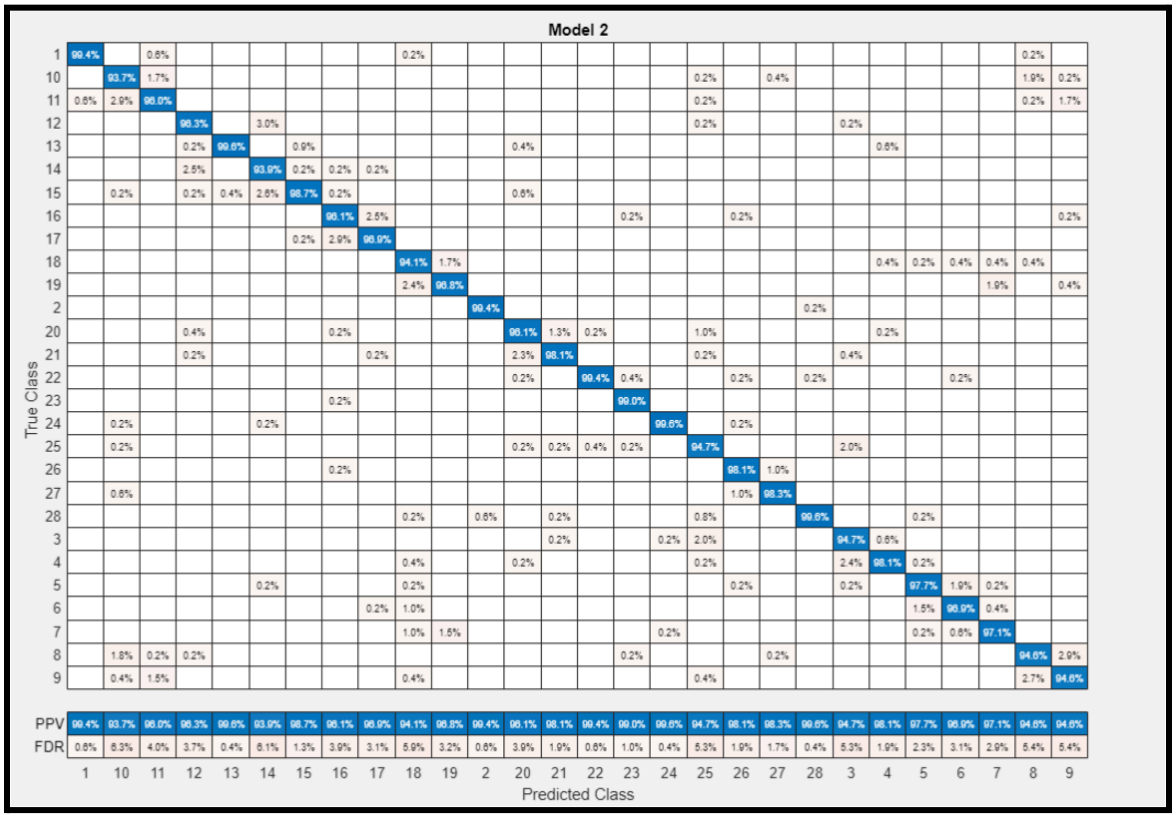
Results:

Validation accuracy: 93.60%
Elapsed time: 103 min 21 sec
Epoch: 20 of 20
Max Iterations: 26880
Iterations per epoch: 1137
Frequency: 50 Iterations
Hardware resource: Single GPU
Learning rate: 0.0001

Confusion matrix for DarkNet19

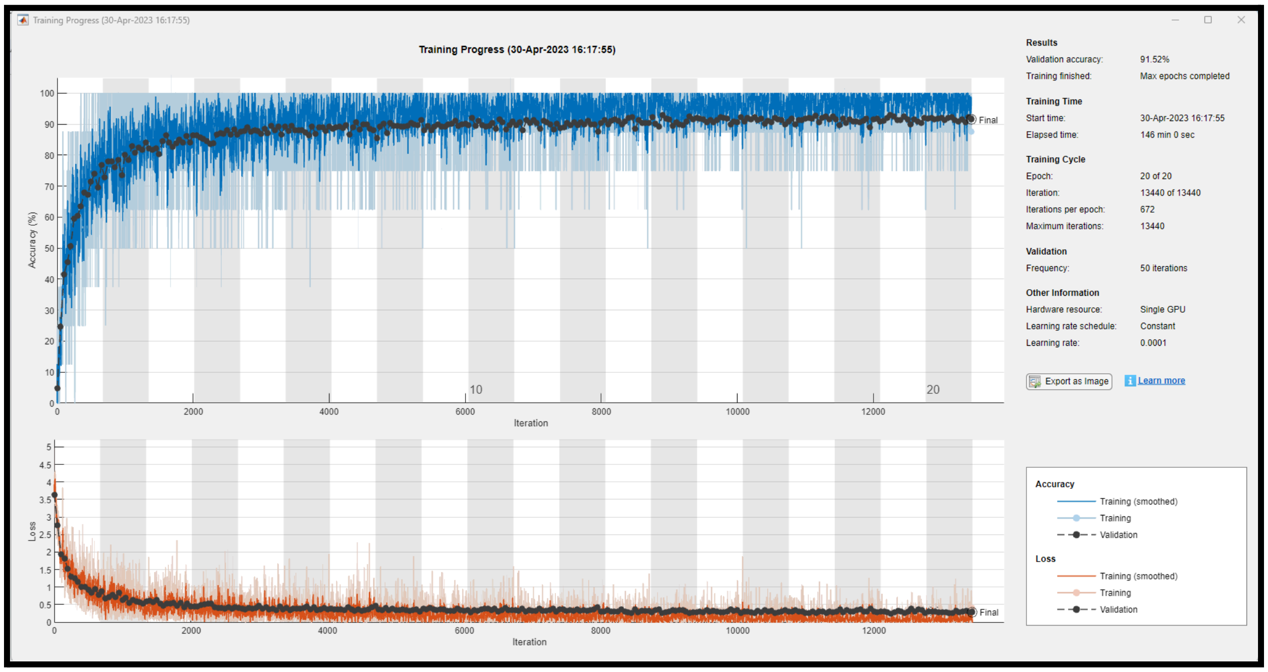


Confusion matrix with % success rate on each class:



Task 3: Implement ResNet-50 Architecture on dataset

The training is performed using the Adam optimizer with a mini-batch size of 8 and a maximum of 20 epochs. The execution is done on a GPU to speed up the training process. Finally, the trained model is saved for future use. The objective of this process is to improve the accuracy of the pre-trained model in recognizing images in a specific domain by fine-tuning it to the new task.



Results

Validation accuracy:	91.52%
Training finished:	Max epochs completed

Training Time

Start time:	30-Apr-2023 16:17:55
Elapsed time:	146 min 0 sec

Training Cycle

Epoch:	20 of 20
Iteration:	13440 of 13440
Iterations per epoch:	672
Maximum iterations:	13440

Validation

Frequency:	50 iterations

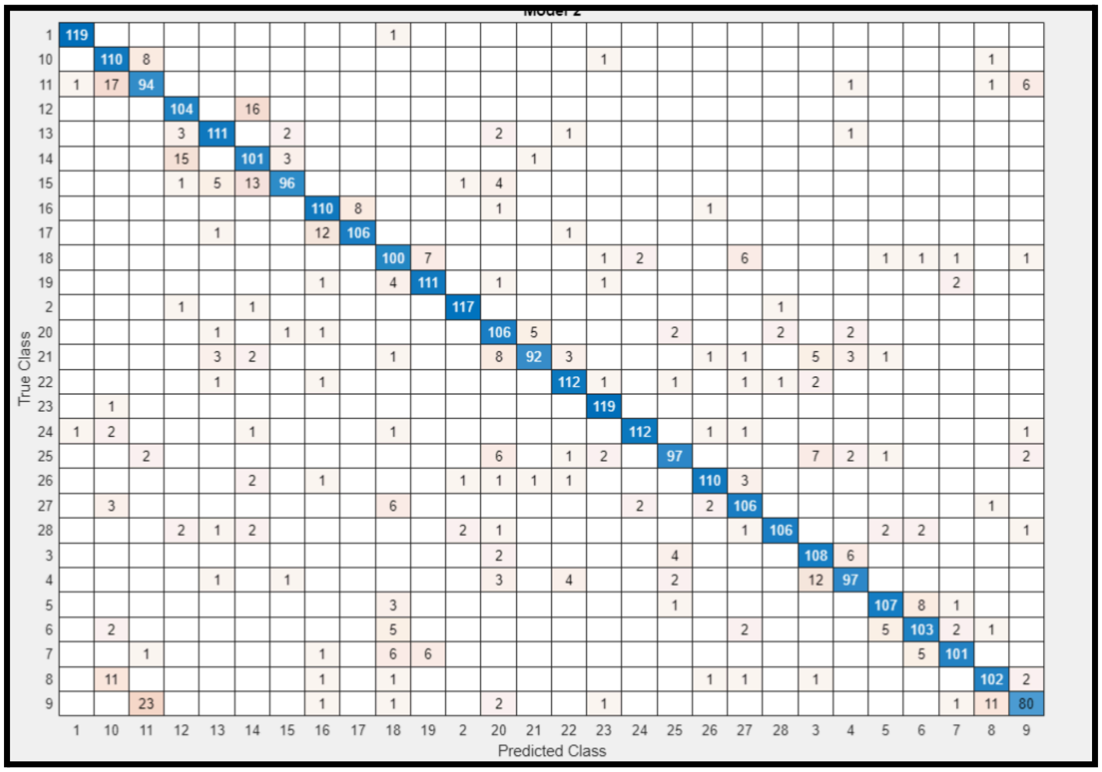
Other Information

Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.0001

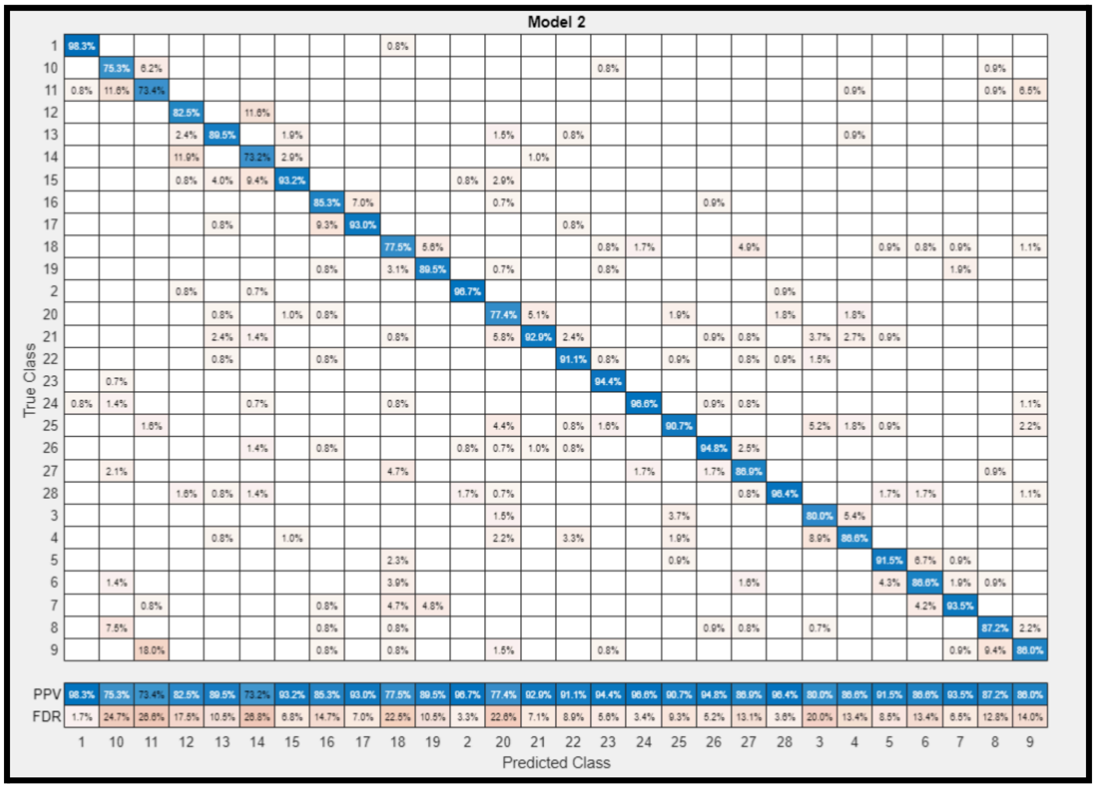
Results:

Validation accuracy: 91.52%
Elapsed time: 146 min 0 sec
Epoch: 20 of 20
Max Iterations: 13440
Iterations per epoch: 672
Frequency: 50 Iterations
Hardware resource: Single GPU
Learning rate: 0.0001

Confusion matrix for ResNet-50

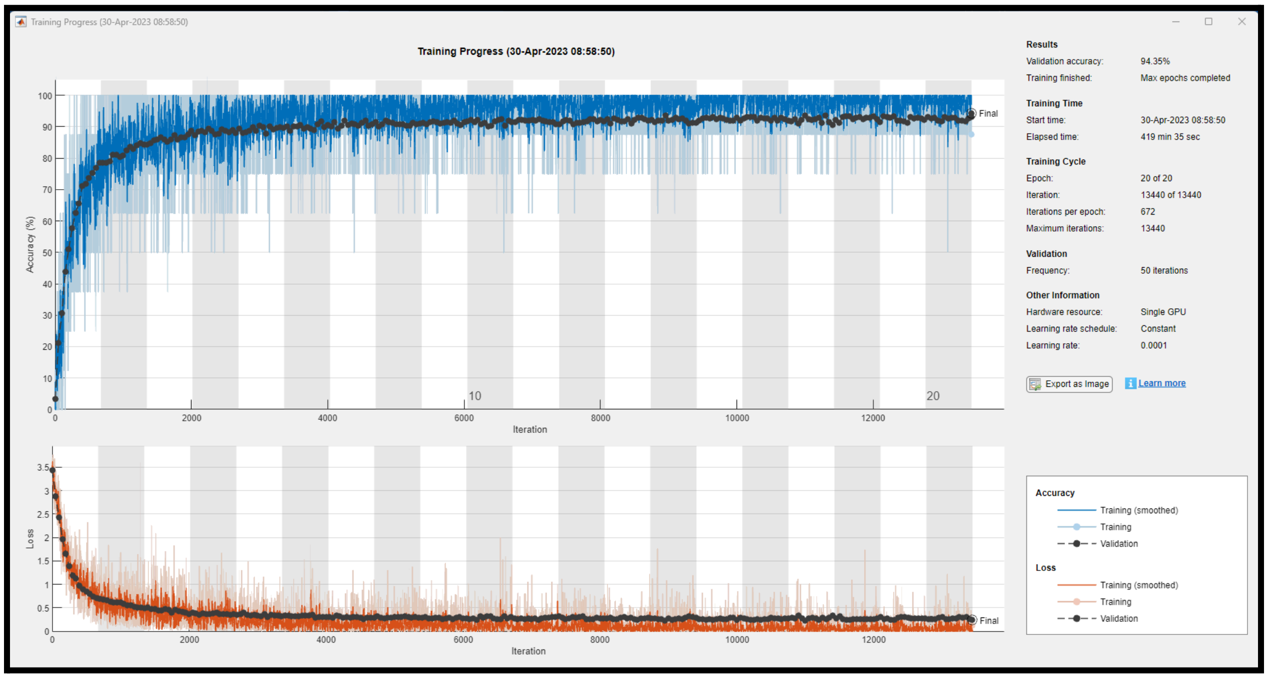


Confusion matrix with % success rate on each class:



Task 4: Implement Efficient B0 Architecture on dataset

The training is performed using the Adam optimizer with a mini-batch size of 8 and a maximum of 20 epochs. The execution is done on a GPU to speed up the training process. Finally, the trained model is saved for future use. The objective of this process is to improve the accuracy of the pre-trained model in recognizing images in a specific domain by fine-tuning it to the new task.



Results

Validation accuracy:	94.35%
Training finished:	Max epochs completed

Training Time

Start time:	30-Apr-2023 08:58:50
Elapsed time:	419 min 35 sec

Training Cycle

Epoch:	20 of 20
Iteration:	13440 of 13440
Iterations per epoch:	672
Maximum iterations:	13440

Validation

Frequency:	50 iterations

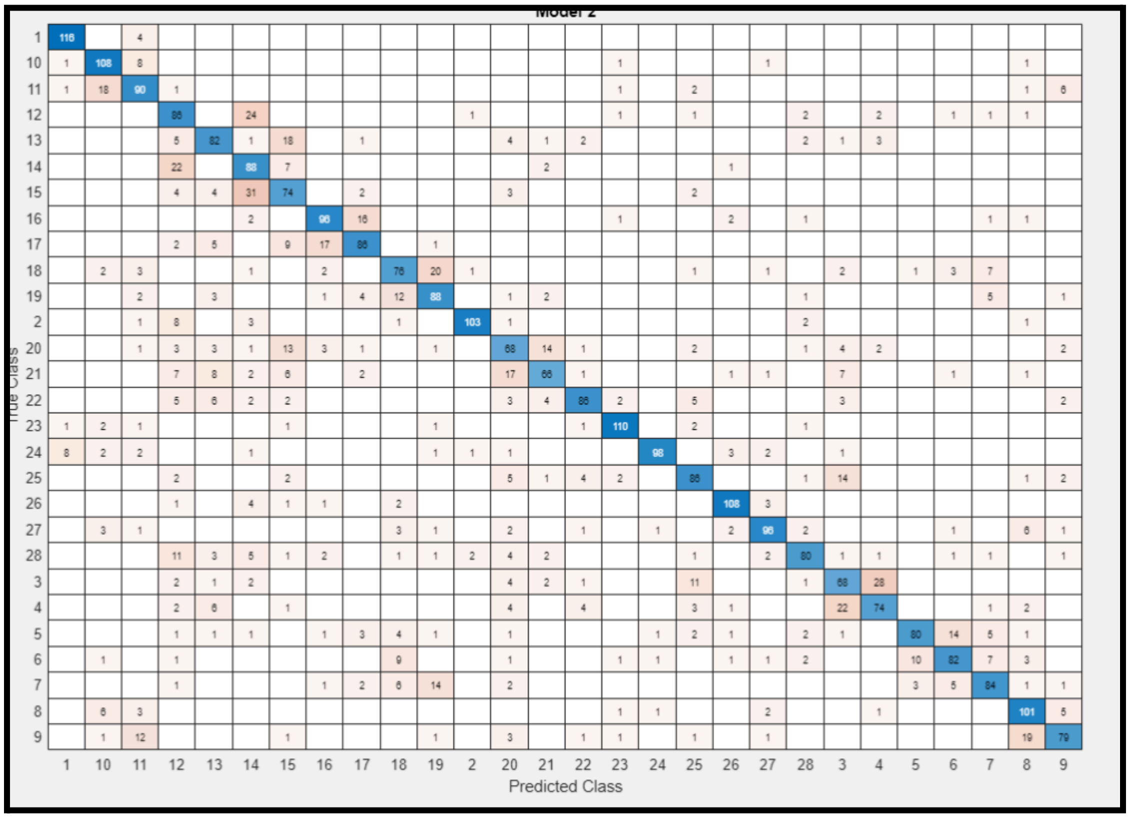
Other Information

Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.0001

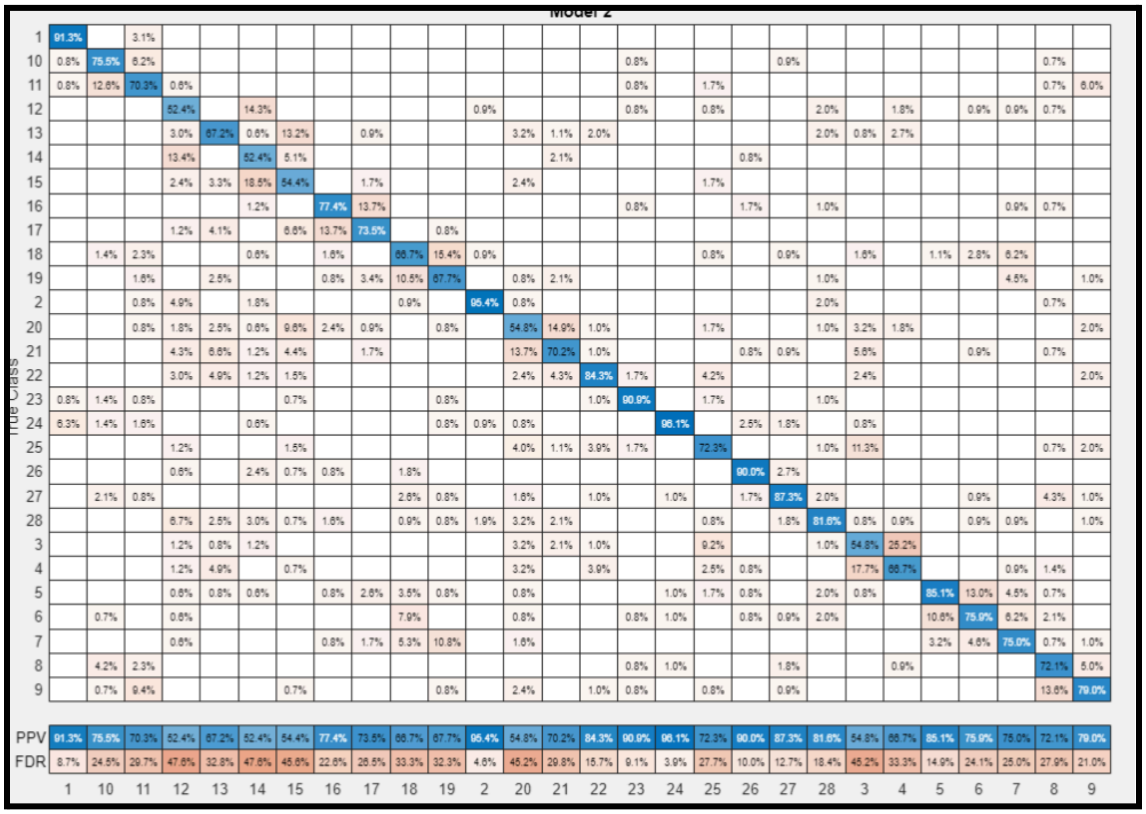
Results:

Validation accuracy: 94.35 %
Elapsed time: 419 min 35 sec
Epoch: 20 of 20
Max Iterations: 13440
Iterations per epoch: 672
Frequency: 50 Iterations
Hardware resource: Single GPU
Learning rate: 0.0001

Confusion matrix for Efficient B0



Confusion matrix with % success rate on each class:



4.2 Data Analysis

In this section of the report, we are going to analyze the time complexity of each algorithm versus the accuracy that it provided on the selected dataset. Moreover, there will be graphs, charts, and tables indicating the details. Finally, there will be a percent success rate for each individual class inside the dataset as well.

Graph: time it took to complete the training

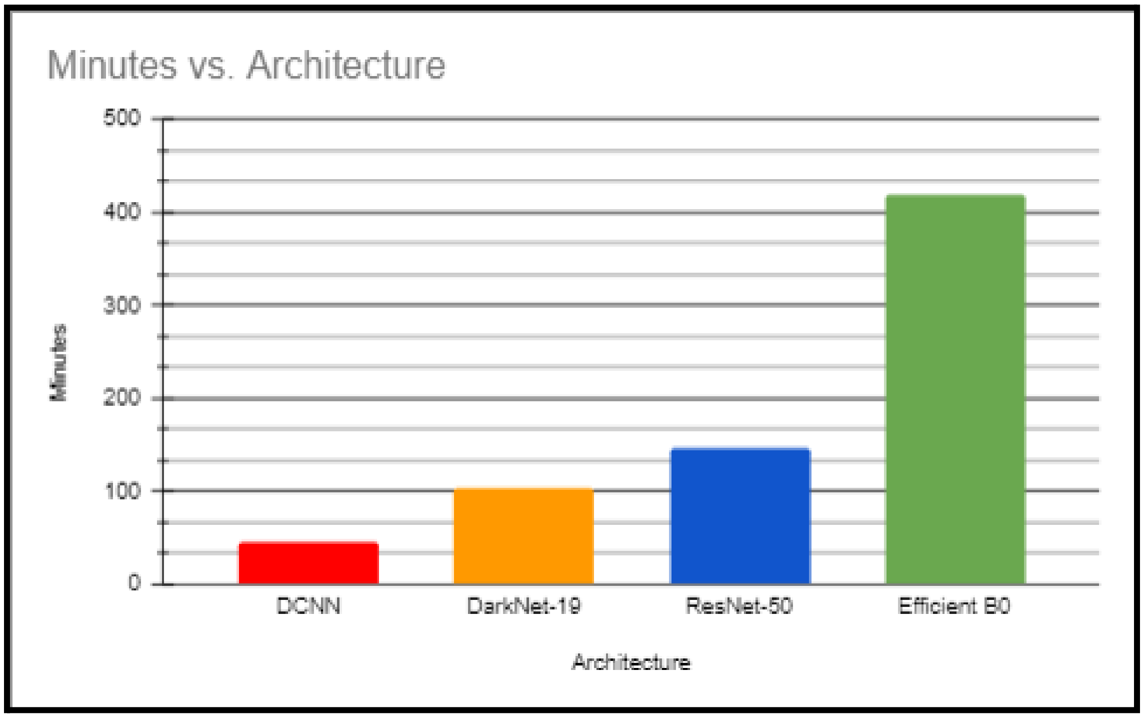


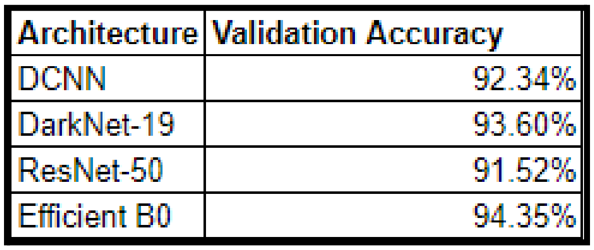
Chart 4.2.1: Time complexity

As we can observe on the graph above, DCNN architecture that was implemented is performing under the least amount of minutes by 46 mins. On the other hand, Efficient B0 is performing the worst on the same dataset architecture by over 400 minutes.

Darknet19 and Resnet50 are approximately close to each other, between 100-140 minutes for their architecture.

Therefore, we can conclude that the DCNN architecture implemented is the best on the time scaling graph.

Graph: Validation Accuracy



Architecture	Validation Accuracy
DCNN	92.34%
DarkNet-19	93.60%
ResNet-50	91.52%
Efficient B0	94.35%

Chart 4.2.2: Validation Accuracy

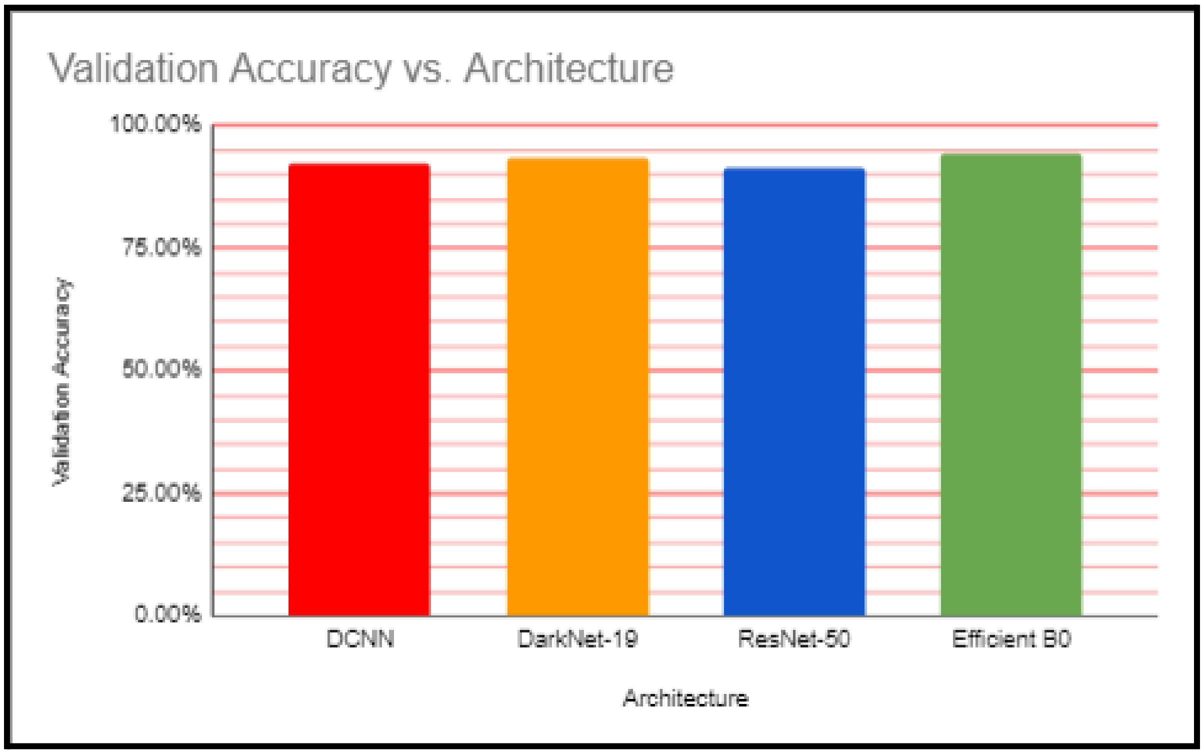


Chart 4.2.3: Validation Accuracy

As we can observe on the graph above, the Efficient B0 architecture that was implemented is performing the best for the resulting validation accuracy from the given training and testing dataset. However, we must emphasize that the rest of the architecture is also in a very close margin by 1-3% difference among them. The lowest performance is coming from ResNet-50 Architecture by a 91.52% validation accuracy, which is still a fantastic result.

Therefore, Efficient B0 Architecture is the best performing, and ResNet-50 is the worst performing architecture in this experiment.

Findings of architecture versus each class created for the dataset

The dataset consists of Handwritten Arabic characters, and it's important to remember that the alphabet has 28 total characters. Therefore, we first have to put each character into its corresponding class before we train.

The Validation accuracy of each architecture designed for the dataset for each class is given in the chart below. Moreover, there is a table generated for it to better visualize the results.

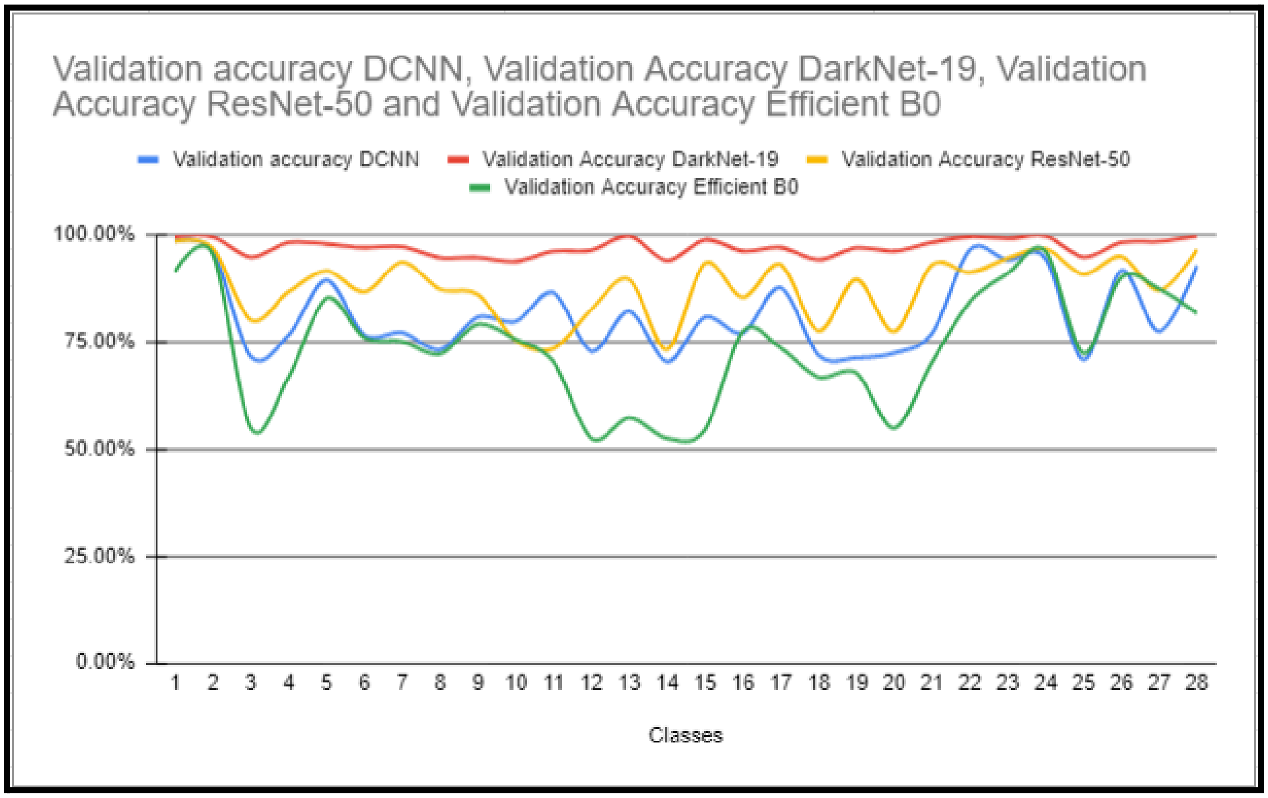


Chart 4.2.4: Validation Accuracy For classes

From the chart above it could be seen that the best validation accuracy across all classes is under DarkNet-19 Architecture. The next one up is ResNet-50, DCNN, and finally the worst one across all classes with huge inconsistency is Efficient B0. The validation accuracy measurement is completed via a classifier tool.

5. CONCLUSIONS

As a conclusion, the experiment was successfully conducted across four different data classifier architectures, DCNN, DarkNet-19, ResNet-50, and Efficient B0. The dataset used in this project consisted of more than 3500 training images in 32x32 size, black and white. The dataset has been organized into 28 classes according to their character in the alphabet. The classes one by one went through a training process across all architectures listed in the project. The results of the training process is as follows: Efficient B0 had the highest validation accuracy of 94.35%, following with DarkNet-19 at validation accuracy of 93.60%, then DCNN architecture that was developed by the student that included total of 6 convolution layers, 6 reluLayers, 5 max pooling layers, 1 global Average Pooling Layer, 1 fully Connected Layer , 1 softmax Layer at validation accuracy of 92.34%, and finally ResNet-50 architecture with a 91.52% validation accuracy.

Furthermore, the time complexity of each architecture played an important role in this experiment. The least time taken for the training process was by DCNN architecture, then DarkNet-19, ResNet-50, and Efficient B0 being the worst with a 419 min training time. This concludes that the DCNN architecture designed has the least time consumption for a very good validation accuracy of 92.35%.

Overall, this project was a learning experience for me to understand how each architecture affects different aspect of the learning process.

6. REFERENCES

Lewis, Nick. "Inside the BREACH attack: How to avoid HTTPS traffic exploits." *TechTarget*, https://www.techtarget.com/searchsecurity/tip/Inside-the-BREACH-attack-How-to-avoid-HTTPS-traffic-exploits. Accessed 28 April 2023.

"9 Man In the Middle Attack Prevention Methods to Use Now." *Cheap SSL Certificates. Buy SSL/HTTPS Certificate $3.98*, 16 November 2021, https://cheapsslsecurity.com/blog/man-in-the-middle-attack-prevention/. Accessed 28 April 2023.

Sen, Kaushik. "What is an Attack Vector? 16 Common Attack Vectors in 2023." *UpGuard*, https://www.upguard.com/blog/attack-vector. Accessed 28 April 2023.

"What is MITM (Man in the Middle) Attack | Imperva." *Imperva, Inc.*, https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/. Accessed 28 April 2023.

APPENDIX A: DCNN Architecture

```
clc;close all;clear;
digitDatasetPath =
fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
imdsTrain = imageDatastore(digitDatasetPath, ...
   'IncludeSubfolders',true,'LabelSource','foldernames');
[imdsTrain,imdsValidation] = splitEachLabel(imdsTrain,.8,'randomized');
digitDatasetPath2 =
fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TesImages');
imdsTest = imageDatastore(digitDatasetPath2, ...
   'IncludeSubfolders',true,'LabelSource','foldernames');
num_images=length(imdsTest.Labels);
perm=randperm(num_images,6);

   net=load('updatedNet.mat');
   net=net.net;

   lgraph = layerGraph(net);
   clear net;
   numClasses = numel(categories(imdsTrain.Labels));

newFCLayer =
fullyConnectedLayer(numClasses,'Name','NewFc','WeightLearnRateFactor',10,'BiasL
earnRateFactor',10);
lgraph = replaceLayer(lgraph,'fc',newFCLayer);
newClassLayer = softmaxLayer('Name','NewSoftmax');
lgraph = replaceLayer(lgraph,'softmax',newClassLayer);
newClassLayer1 = classificationLayer('Name','classification');
lgraph = addLayers(lgraph,newClassLayer1);
lgraph = replaceLayer(lgraph,'classification',newClassLayer1);
lgraph = connectLayers(lgraph,'NewSoftmax','classification');

   augmenter = imageDataAugmenter( ...
       'RandRotation',[-5 5],'RandXReflection',1,...
       'RandYReflection',1,'RandXShear',[-0.05 0.05],'RandYShear',[-0.05
0.05]);
 auimds = augmentedImageDatastore([224
224],imdsTrain,'DataAugmentation',augmenter,'ColorPreprocessing','gray2rgb');
   auimdsVali = augmentedImageDatastore([224
224],imdsValidation,'DataAugmentation',augmenter,'ColorPreprocessing','gray2rgb
');
   options = trainingOptions('adam',...
       'MaxEpochs',20,'MiniBatchSize',8,...
       'ExecutionEnvironment','gpu',...
       'Shuffle','every-epoch', ...
       'InitialLearnRate',1e-4, ...
       'ValidationData',auimdsVali,...
       'Verbose',false, ...
```

```
    'Plots','training-progress');

% Training
UpdatedNovelCNModel = trainNetwork(auimds,lgraph,options);
save('UpdatedNovelCNModel','UpdatedNovelCNModel');
```

APPENDIX B: DCNN Architecture Layers

```
layers = [
    imageInputLayer([224 224 3],"Name","imageinput","Normalization","none")
    convolution2dLayer([3 3],32,"Name","conv","Padding","same")
    reluLayer("Name","relu")
    maxPooling2dLayer([3 3],"Name","maxpool","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],64,"Name","conv_1","Padding","same")
    reluLayer("Name","relu_1")
    maxPooling2dLayer([3 3],"Name","maxpool_1","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],128,"Name","conv_2","Padding","same")
    reluLayer("Name","relu_2")
    maxPooling2dLayer([3 3],"Name","maxpool_2","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],256,"Name","conv_3","Padding","same")
    reluLayer("Name","relu_3")
    maxPooling2dLayer([3 3],"Name","maxpool_3","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],512,"Name","conv_4","Padding","same")
    reluLayer("Name","relu_4")
    maxPooling2dLayer([3 3],"Name","maxpool_4","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],1024,"Name","conv_5","Padding","same")
    reluLayer("Name","relu_5")
    globalAveragePooling2dLayer("Name","gapool")
    fullyConnectedLayer(10,"Name","fc")
    softmaxLayer("Name","softmax")];
```

APPENDIX C: DarkNet-19 Architecture

```
clc;close all;clear;
%%cross validation
%load images
digitDatasetPath =
fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
imdsTrain = imageDatastore(digitDatasetPath, ...
   'IncludeSubfolders',true,'LabelSource','foldernames');
[imdsTrain,imdsValidation] = splitEachLabel(imdsTrain,.8,'randomized');
digitDatasetPath2 =
fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
imdsTest = imageDatastore(digitDatasetPath2, ...
   'IncludeSubfolders',true,'LabelSource','foldernames');
% Determine the split up
total_split=countEachLabel(imdsTrain)
% Number of Images
num_images=length(imdsTest.Labels);
%%K-fold Validation
% Number of folds
num_folds=2;
% Loop for each fold
for fold_idx=1:num_folds

   fprintf('Processing %d among %d folds \n',fold_idx,num_folds);

  % Test Indices for current fold
   test_idx=fold_idx:num_folds:num_images;
   % Test cases for current fold
   imdsTest = subset(imdsTest,test_idx);
   countEachLabel(imdsTest)
   % Train indices for current fold
   train_idx=setdiff(1:length(imdsTrain.Files),test_idx);

   % Train cases for current fold
   imdsTrain = subset(imdsTrain,train_idx);
   countEachLabel(imdsTrain)

   net=darknet19;
    % Replacing the last layers with new layers
   layersTransfer = net.Layers(1:end-4);
   clear net;
% Number of categories
   numClasses = numel(categories(imdsTrain.Labels));
layers = [
   layersTransfer
   globalAveragePooling2dLayer
   fullyConnectedLayer(numClasses)
   softmaxLayer
   classificationLayer];

   % Data Augumentation
   augmenter = imageDataAugmenter( ...
       'RandRotation',[-5 5],'RandXReflection',1,...
       'RandYReflection',1,'RandXShear',[-0.05 0.05],'RandYShear',[-0.05
0.05]);

   % Resizing all training images to [224 224] for ResNet architecture
   auimds = augmentedImageDatastore([256
256],imdsTrain,'DataAugmentation',augmenter,'ColorPreprocessing','gray2rgb');
   auimdsVali = augmentedImageDatastore([256
256],imdsValidation,'DataAugmentation',augmenter,'ColorPreprocessing','gray2rgb
');
   options = trainingOptions('adam',...
       'MaxEpochs',20,'MiniBatchSize',8,...
       'ExecutionEnvironment','gpu',...
       'Shuffle','every-epoch', ...
       'InitialLearnRate',1e-4, ...
       'ValidationData',auimdsVali,...
       'Verbose',false, ...
       'Plots','training-progress');
    % Training
   netTransferDark19 = trainNetwork(auimds,layers,options);
   save('netTransferDark19','netTransferDark19')

end
```

APPENDIX D: ResNet-50 Architecture

```
clc;close all;clear;
%%cross validation
%load images
digitDatasetPath =
fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
imdsTrain = imageDatastore(digitDatasetPath, ...
   'IncludeSubfolders',true,'LabelSource','foldernames');
[imdsTrain,imdsValidation] = splitEachLabel(imdsTrain,.8,'randomized');
digitDatasetPath2 =
fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
imdsTest = imageDatastore(digitDatasetPath2, ...
   'IncludeSubfolders',true,'LabelSource','foldernames');
% Determine the split up
total_split=countEachLabel(imdsTrain)
% Number of Images
num_images=length(imdsTest.Labels);
% Number of folds
num_folds=2;
% Loop for each fold
for fold_idx=1:num_folds

   fprintf('Processing %d among %d folds \n',fold_idx,num_folds);

  % Test Indices for current fold
   test_idx=fold_idx:num_folds:num_images;
   % Test cases for current fold
   imdsTest = subset(imdsTest,test_idx);
   countEachLabel(imdsTest)
   % Train indices for current fold
   train_idx=setdiff(1:length(imdsTrain.Files),test_idx);

   % Train cases for current fold
   imdsTrain = subset(imdsTrain,train_idx);
   countEachLabel(imdsTrain)
   % ResNet Architecture
   net=resnet50;
   lgraph = layerGraph(net);
   clear net;

   % Number of categories
   numClasses = numel(categories(imdsTrain.Labels));

   % New Learnable Layer
   newLearnableLayer = fullyConnectedLayer(numClasses, ...
       'Name','new_fc', ...
       'WeightLearnRateFactor',10, ...
```

```
        'BiasLearnRateFactor',10);

    % Replacing the last layers with new layers
    lgraph = replaceLayer(lgraph,'fc1000',newLearnableLayer);
    newsoftmaxLayer = softmaxLayer('Name','new_softmax');
    lgraph = replaceLayer(lgraph,'fc1000_softmax',newsoftmaxLayer);
    newClassLayer = classificationLayer('Name','new_classoutput');
    lgraph = replaceLayer(lgraph,'ClassificationLayer_fc1000',newClassLayer);

    % Data Augumentation
    augmenter = imageDataAugmenter( ...
        'RandRotation',[-5 5],'RandXReflection',1,...
        'RandYReflection',1,'RandXShear',[-0.05 0.05],'RandYShear',[-0.05
0.05]);

    % Resizing all training images to [224 224] for ResNet architecture
    auimds = augmentedImageDatastore([224
224],imdsTrain,'DataAugmentation',augmenter,'ColorPreprocessing','gray2rgb');
    auimdsVali = augmentedImageDatastore([224
224],imdsValidation,'DataAugmentation',augmenter,'ColorPreprocessing','gray2rgb
');
    options = trainingOptions('adam',...
        'MaxEpochs',20,'MiniBatchSize',8,...
        'ExecutionEnvironment','auto',...
        'Shuffle','every-epoch', ...
        'InitialLearnRate',1e-4, ...
        'ValidationData',auimdsVali,...
        'Verbose',false, ...
        'Plots','training-progress');


    % Training
    netTransferRes50 = trainNetwork(auimds,lgraph,options);
    save('netTransferRes50','netTransferRes50')

end
```

APPENDIX E: Efficient B0 Architecture

```
clc;close all;clear;
%%cross validation
%load images
digitDatasetPath =
fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
imdsTrain = imageDatastore(digitDatasetPath, ...
   'IncludeSubfolders',true,'LabelSource','foldernames');
[imdsTrain,imdsValidation] = splitEachLabel(imdsTrain,.8,'randomized');
digitDatasetPath2 =
fullfile('C:\Users\puyaf\OneDrive\Desktop\Codes\Ds\TrainedImages');
imdsTest = imageDatastore(digitDatasetPath2, ...
   'IncludeSubfolders',true,'LabelSource','foldernames');
% Determine the split up
total_split=countEachLabel(imdsTrain)
% Number of Images
num_images=length(imdsTest.Labels);

%%K-fold Validation
% Number of folds
num_folds=2;
% Loop for each fold
for fold_idx=1:num_folds

   fprintf('Processing %d among %d folds \n',fold_idx,num_folds);

  % Test Indices for current fold
   test_idx=fold_idx:num_folds:num_images;
   % Test cases for current fold
   imdsTest = subset(imdsTest,test_idx);
   countEachLabel(imdsTest)
   % Train indices for current fold
   train_idx=setdiff(1:length(imdsTrain.Files),test_idx);

   % Train cases for current fold
   imdsTrain = subset(imdsTrain,train_idx);
   countEachLabel(imdsTrain)

   net=efficientnetb0;
   lgraph = layerGraph(net);
   clear net;

   % Number of categories
   numClasses = numel(categories(imdsTrain.Labels));

   % New Learnable Layer
   newLearnableLayer = fullyConnectedLayer(numClasses, ...
       'Name','new_fc', ...
```

```
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);

   % Replacing the last layers with new layers
   lgraph = 
replaceLayer(lgraph,'efficientnet-b0|model|head|dense|MatMul',newLearnableLayer
);
   newsoftmaxLayer = softmaxLayer('Name','new_softmax');
   lgraph = replaceLayer(lgraph,'Softmax',newsoftmaxLayer);
   newClassLayer = classificationLayer('Name','new_classoutput');
   lgraph = replaceLayer(lgraph,'classification',newClassLayer);

   % Data Augumentation
   augmenter = imageDataAugmenter( ...
        'RandRotation',[-5 5],'RandXReflection',1,...
        'RandYReflection',1,'RandXShear',[-0.05 0.05],'RandYShear',[-0.05
0.05]);

   % Resizing all training images to [224 224] for ResNet architecture
   auimds = augmentedImageDatastore([224
224],imdsTrain,'DataAugmentation',augmenter,'ColorPreprocessing','gray2rgb');
   auimdsVali = augmentedImageDatastore([224
224],imdsValidation,'DataAugmentation',augmenter,'ColorPreprocessing','gray2rgb
');
   options = trainingOptions('adam',...
        'MaxEpochs',20,'MiniBatchSize',8,...
        'ExecutionEnvironment','gpu',...
        'Shuffle','every-epoch', ...
        'InitialLearnRate',1e-4, ...
        'ValidationData',auimdsVali,...
        'Verbose',false, ...
        'Plots','training-progress');


   % Training
   netTransferEfficient = trainNetwork(auimds,lgraph,options);
   save('netTransferEfficient','netTransferEfficient')

end
```