ACS61011 Deep Learning Project

Animesh Sandhu

Registration No. 230235405

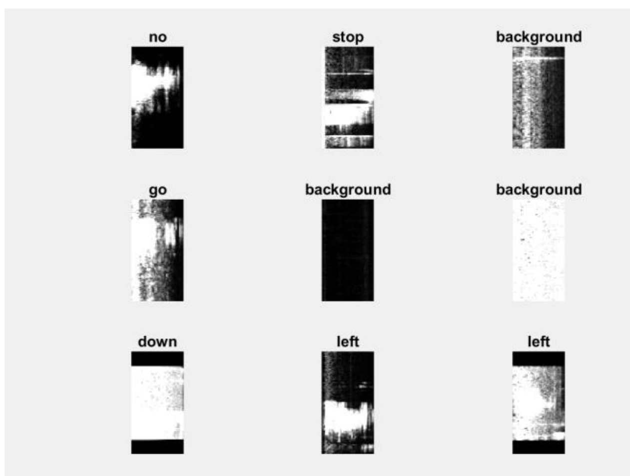
The assignment is to design, implement and evaluate a deep learning system.

INTRODUCTION

This project focuses on developing an end-to-end speech recognition system using deep learning techniques and spectrogram images as input data. The primary goal was to classify spoken words into one of twelve predefined categories. Initially, a baseline Convolutional Neural Network (CNN) was implemented and trained on preprocessed speech spectrograms. This model was iteratively improved using data augmentation strategies to simulate real-world variations, hyperparameter tuning to optimize performance, and ensemble learning to improve generalization.

To further enhance accuracy and robustness, transfer learning approaches using GoogLeNet and ResNet50 architectures were explored. These pre-trained models allowed for leveraging feature representations learned on large-scale image datasets and adapting them to the speech recognition task. Performance was assessed using accuracy metrics, confusion matrices, and training-validation plots across all experimental stages.

Baseline Model



- The raw speech data was preprocessed into grayscale spectrogram images of size 98x50, which formed the input to our model.
- The initial CNN architecture included:
    - An image input layer with shape [98x50x1]
    - Two convolutional layers, each followed by batch normalization, ReLU activation, and max pooling to reduce spatial dimensions.
    - A dropout layer with a rate of 0.4 to mitigate overfitting

    - A fully connected layer followed by a softmax and a classification layer for multi-class output

- The model was trained using the Adam optimizer with a learning rate of 0.001, a batch size of 32, and 50 epochs.
- On the validation set, the model achieved an accuracy of approximately 57.64%.
- A confusion matrix was generated to visualize misclassifications, and the training/validation progress was plotted.

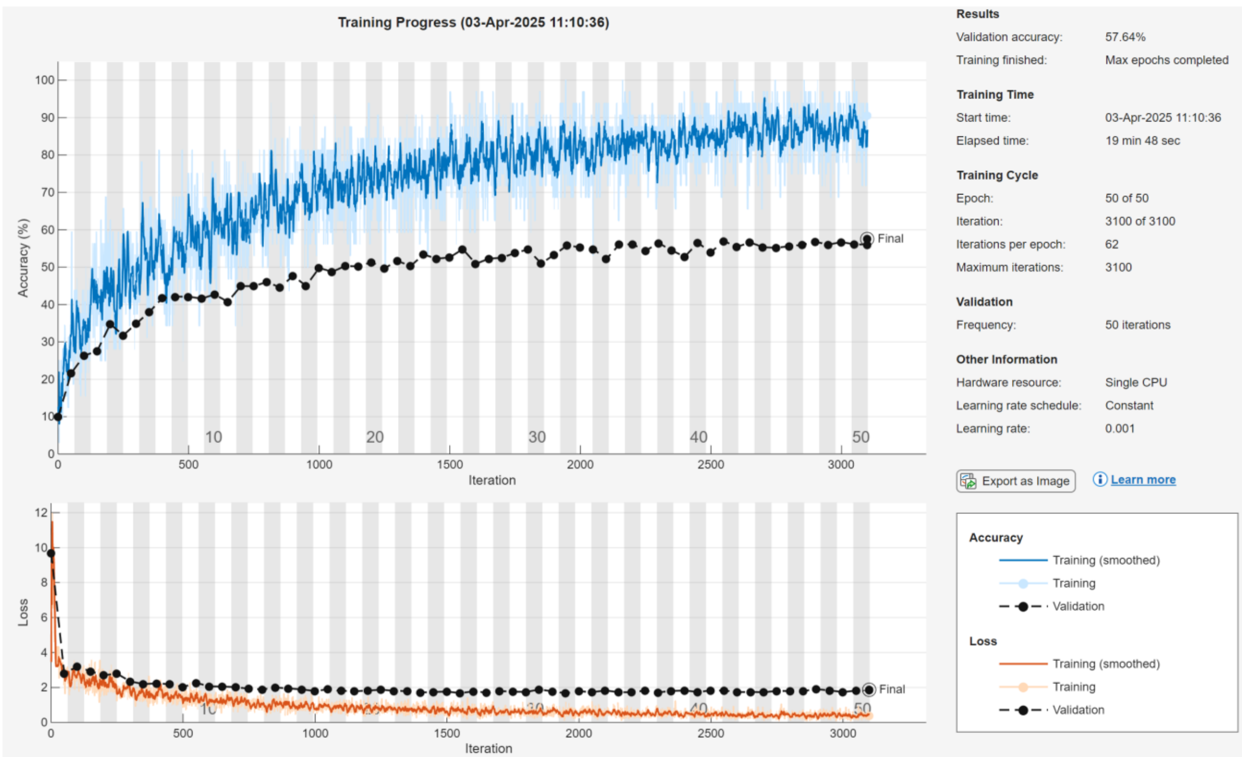


Figure 1. Baseline training plot

Data Augmentation

- To increase robustness, training data was augmented using MATLAB's imageDataAugmenter with the following transformations:
    - Random horizontal and vertical shifts (±3 pixels)
    - Random scaling in both axes (between 0.9 and 1.1)

- The same CNN architecture was retrained using the augmented dataset while keeping the validation set unchanged.
- As a result, the model showed improved generalization and achieved a higher validation accuracy of ~71.65%.
- The training/validation curves indicated smoother convergence with reduced overfitting.

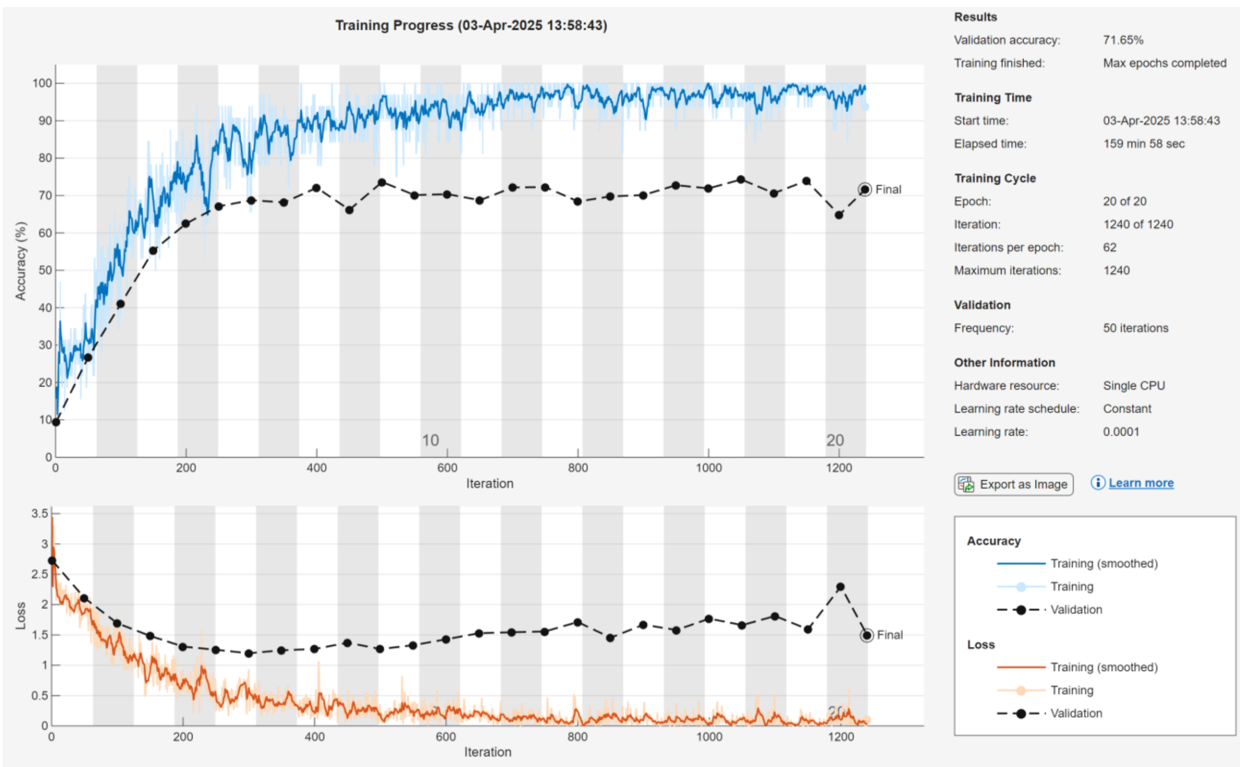


Figure 2: Training plot after augmentation

Hyperparameter Search

- A grid search approach was used to identify optimal values for two key hyperparameters:
  - Number of filters in convolutional layers: 32, 64
  - Number of convolutional blocks: 2, 3
- All configurations used consistent training parameters.

- The results showed that the configuration with 32 filters and 3 convolutional layers yielded the best validation accuracy of 66.52%.

Accuracy Table:

Filters	Layers	Accuracy (%)
32	2	56.11
32	3	66.52
64	2	57.05
64	3	65.67

Figure 5: Training/validation plot – Best hyperparameter config

Figure 6: Confusion matrix – Best config

Model Averaging (Ensemble Learning)

- To further boost model stability and reduce variance, an ensemble learning strategy using bagging was implemented.
- Three CNN models were trained independently on different random 80% subsets of the training data.
- Each model used the best architecture (32 filters, 3 conv layers) identified from hyperparameter tuning.
- Predictions were combined using majority voting (mode) across models.
- The ensemble achieved an improved validation accuracy of approximately 60.38%, highlighting the benefits of averaging across multiple learners.

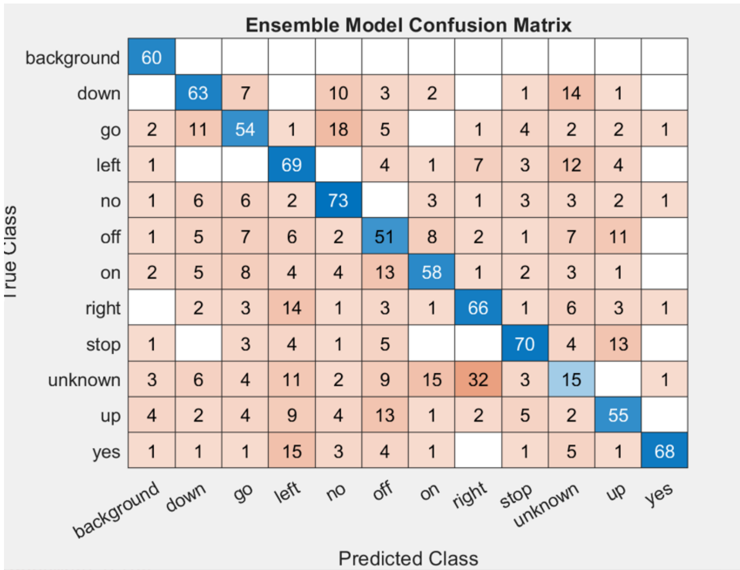


Figure 6: Confusion matrix – Ensemble prediction

```
Training Model 1/3...
Training Model 2/3...
Training Model 3/3...
All 3 models trained successfully!
Ensemble Validation Accuracy: 60.38%
```

Figure7: Ensemble model accuracy plot

Open-Ended Extension: Transfer Learning

GoogLeNet:

- GoogLeNet was imported and modified by replacing the final fully connected layer, softmax, and classification layer with custom layers for 12-class classification.
- Since GoogLeNet expects RGB input, grayscale spectrograms were replicated across three channels using gray2rgb.
- Images were resized to 224x224x3 to match the model's input requirement.
- Training used the Adam optimizer with a reduced learning rate (0.0001) and ran for 20 epochs.
- The model achieved 74.21% validation accuracy.

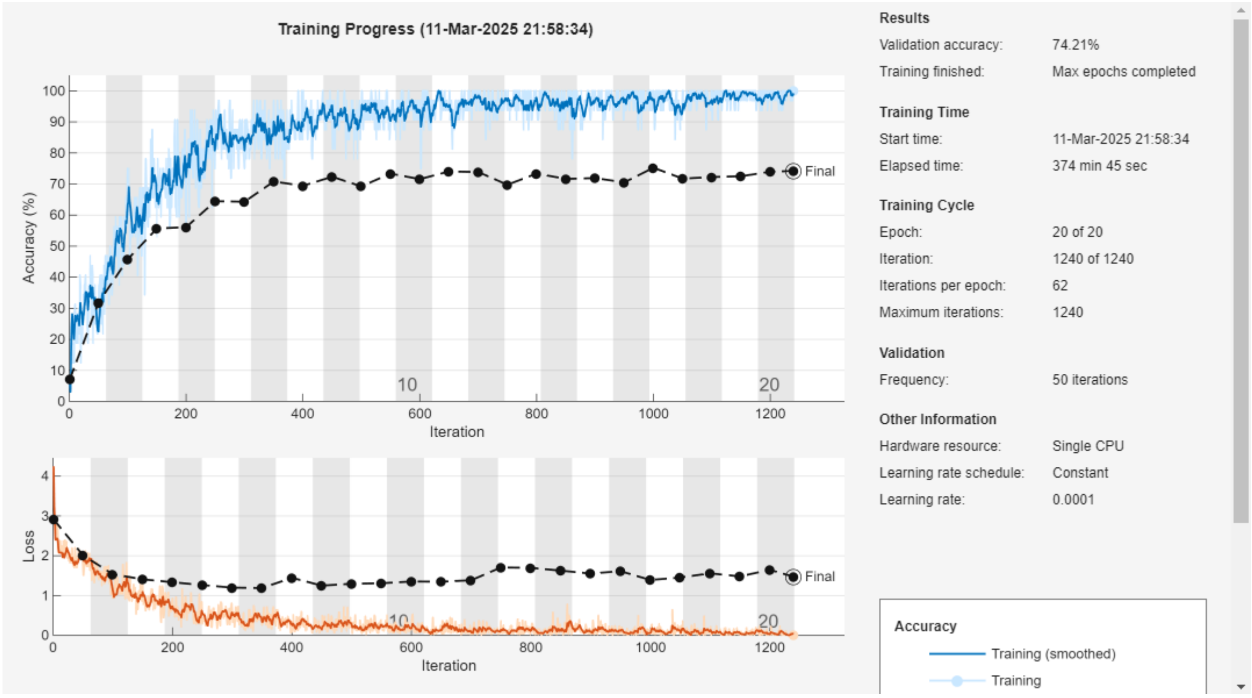


Figure 8: Googlenet

ResNet50:

- ResNet50 was used similarly, with final layers replaced to fit the 12-class task.
- Advanced augmentation was used including:
  - Image reflection, random rotation (±20 degrees), and larger scaling variation (0.7–1.3)
- A piecewise learning rate schedule was applied to improve learning dynamics.
- Trained over 50 epochs with an initial learning rate of 0.0005.
- Achieved ~71% validation accuracy, with strong generalization seen in the validation confusion matrix.

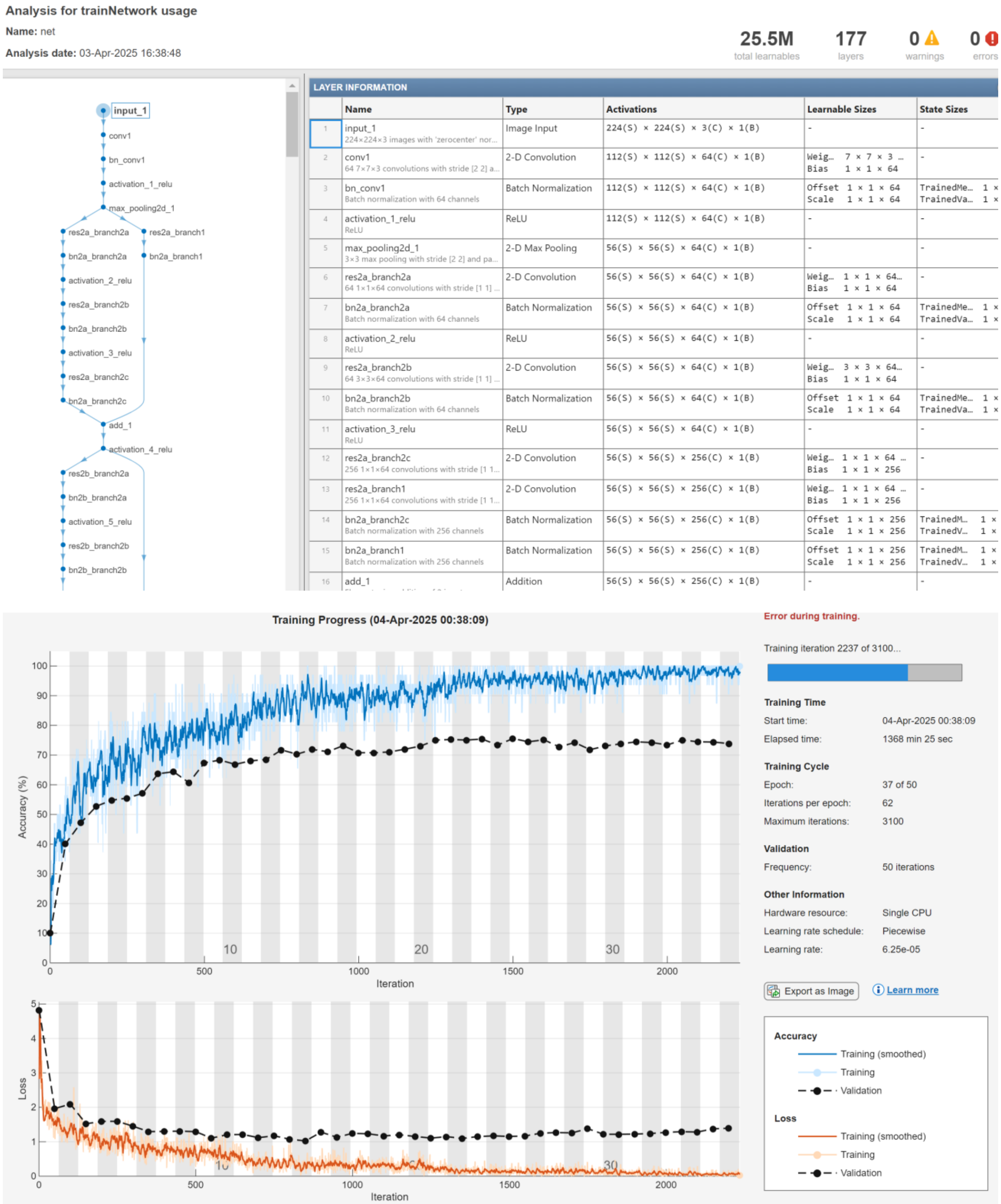


Figure 8: ResNet50 training/validation plot

Conclusion

This project demonstrated the application of deep learning for automated speech recognition using spectrogram images. Starting with a custom CNN baseline model, performance was enhanced progressively through data augmentation and hyperparameter tuning. Ensemble learning further improved robustness, while transfer learning using GoogLeNet and ResNet50 significantly boosted accuracy and generalization. The best performing models achieved ~71% validation accuracy, showing the effectiveness of deep CNNs and pre-trained networks in this domain.

Evaluation was supported with visual tools such as training curves and confusion matrices, validating the consistency and effectiveness of the developed system.

Appendix : full code

```
clear all;


% define the random number seed for repeatable results

rng(1,'twister');


%% Load Speech Data


% create an image data store from the raw images

imdsTrain = imageDatastore('speechImageData\TrainData',...

"IncludeSubfolders",true,"LabelSource","foldernames")


% create an image validation data store from the validation images

imdsVal = imageDatastore('speechImageData\ValData',...

"IncludeSubfolders",true,"LabelSource","foldernames")


%% figure;

perm = randperm(numel(imdsTrain.Files), 9);

for i = 1:9
```

```
    subplot(3,3,i);
    img = readimage(imdsTrain, perm(i));
    imshow(img);
    title(string(imdsTrain.Labels(perm(i))));
end
%% % Define augmentation
imageAugmenter = imageDataAugmenter( ...
    'RandXTranslation', [-3 3], ... % Random shifts in X-direction
    'RandYTranslation', [-3 3], ... % Random shifts in Y-direction
    'RandXScale', [0.9 1.1], ... % Random scaling in X-direction
    'RandYScale', [0.9 1.1]); % Random scaling in Y-direction

% Apply augmentation to training data
imageSize = [98 50 1];
augimdsTrain = augmentedImageDatastore(imageSize, imdsTrain, ...
    'DataAugmentation', imageAugmenter);

% Keep validation data unchanged
augimdsVal = imdsVal;

%% % Define improved CNN architecture
layers = [
    imageInputLayer([98 50 1])

    convolution2dLayer(3, 32, 'Padding', 'same') % Increased filters (16 → 32)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)
```

```
    convolution2dLayer(3, 64, 'Padding', 'same') % Increased filters (32 → 64)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)

    dropoutLayer(0.4) % Increased dropout to prevent overfitting

    fullyConnectedLayer(12)
    softmaxLayer
    classificationLayer];

% Retrain the CNN with augmentation
options = trainingOptions('adam', ...
    'MaxEpochs', 50, ...
    'MiniBatchSize', 32, ...
    'InitialLearnRate', 0.001, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', augimdsVal, ...
    'Plots', 'training-progress', ...
    'Verbose', false);

net = trainNetwork(augimdsTrain, layers, options);
%% % Define possible configurations
filterSizes = [32, 64]; % Number of filters in Conv layers
numConvLayers = [2, 3]; % Number of Conv blocks

% Store results
results = [];
```

```
for f = 1:length(filterSizes)
  for l = 1:length(numConvLayers)

    % Define CNN dynamically
    layers = [
      imageInputLayer([98 50 1])];

    for i = 1:numConvLayers(l) % Loop over conv layers
      layers = [layers;
        convolution2dLayer(3, filterSizes(f), 'Padding', 'same')
        batchNormalizationLayer
        reluLayer
        maxPooling2dLayer(2, 'Stride', 2)];
    end

    layers = [layers;
      dropoutLayer(0.4)
      fullyConnectedLayer(12)
      softmaxLayer
      classificationLayer];

    % Training options
    options = trainingOptions('adam', ...
      'MaxEpochs', 50, ...
      'MiniBatchSize', 32, ...
      'InitialLearnRate', 0.001, ...
      'Shuffle', 'every-epoch', ...
      'ValidationData', augimdsVal, ...
      'Verbose', false);
```

```
        % Train the model
        net = trainNetwork(augimdsTrain, layers, options);

        % Evaluate
        YPred = classify(net, imdsVal);
        YValidation = imdsVal.Labels;
        accuracy = sum(YPred == YValidation) / numel(YValidation);

        % Store results
        results = [results; filterSizes(f), numConvLayers(l), accuracy*100];
        fprintf('Filters: %d, Layers: %d, Accuracy: %.2f%%\n', filterSizes(f), numConvLayers(l), accuracy*100);
    end
end

% Display results table
disp(array2table(results, 'VariableNames', {'Filters', 'Layers', 'Accuracy'}));
%% % Number of ensemble models
numModels = 3;

% Store trained networks
nets = cell(1, numModels);

% Define the best CNN architecture (32 filters, 3 layers)
layers = [
    imageInputLayer([98 50 1])

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

```
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)

    convolution2dLayer(3, 32, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)

    convolution2dLayer(3, 32, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)

    dropoutLayer(0.4)

    fullyConnectedLayer(12)
    softmaxLayer
    classificationLayer];

% Training options
options = trainingOptions('adam', ...
    'MaxEpochs', 50, ...
    'MiniBatchSize', 32, ...
    'InitialLearnRate', 0.001, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', augimdsVal, ...
    'Verbose', false);

% Train 3 CNN models with different random subsets
```

```
for i = 1:numModels
   fprintf('Training Model %d/%d...\n', i, numModels);

   % Create a new training subset (random sampling)
   subsetIdx = randperm(numel(imdsTrain.Files), round(0.8 * numel(imdsTrain.Files)));
   imdsSubset = subset(imdsTrain, subsetIdx);

   % Train the model
   nets{i} = trainNetwork(imdsSubset, layers, options);
end

fprintf('All %d models trained successfully!\n', numModels);
%% % Get predictions from all models
YPred1 = classify(nets{1}, imdsVal);
YPred2 = classify(nets{2}, imdsVal);
YPred3 = classify(nets{3}, imdsVal);

% Convert categorical labels to arrays
YPred = [YPred1, YPred2, YPred3];

% Majority voting (mode of predictions)
finalPredictions = mode(YPred, 2);

% Get actual validation labels
YValidation = imdsVal.Labels;

% Calculate final ensemble accuracy
ensembleAccuracy = sum(finalPredictions == YValidation) / numel(YValidation);
fprintf('Ensemble Validation Accuracy: %.2f%%\n', ensembleAccuracy * 100);
```

```
% Display Confusion Matrix
figure;
confusionchart(YValidation, finalPredictions);
title('Ensemble Model Confusion Matrix');
%% %% Step 1: Load and Modify GoogLeNet
net = googlenet;
inputSize = net.Layers(1).InputSize; % Get input size

% Convert to Layer Graph
lgraph = layerGraph(net);

% Display all layers (to confirm names)
disp({lgraph.Layers.Name}');

%% Step 2: Modify Final Layers for 12 Classes
% Create new layers with unique names to avoid conflicts
newFC = fullyConnectedLayer(12, 'Name', 'new_fc', 'WeightLearnRateFactor', 10, 'BiasLearnRateFactor', 10);
newSoftmax = softmaxLayer('Name', 'new_softmax');
newClassOutput = classificationLayer('Name', 'new_output');

% Replace old layers with new ones
lgraph = replaceLayer(lgraph, "loss3-classifier", newFC);
lgraph = replaceLayer(lgraph, "prob", newSoftmax);
lgraph = replaceLayer(lgraph, "output", newClassOutput);

%% Step 3: Load & Augment Training Data
% Load the dataset
dataDir = fullfile('speechImageData');
```

```
imdsTrain = imageDatastore(fullfile(dataDir, 'TrainData'), 'IncludeSubfolders', true, 'LabelSource',
'foldernames');

imdsVal = imageDatastore(fullfile(dataDir, 'ValData'), 'IncludeSubfolders', true, 'LabelSource', 'foldernames');


% Define augmentation

imageAugmenter = imageDataAugmenter( ...

   'RandXTranslation', [-3 3], ...

   'RandYTranslation', [-3 3], ...

   'RandXScale', [0.9 1.1], ...

   'RandYScale', [0.9 1.1]);


% Apply augmentation to training images

imageSize = [inputSize(1) inputSize(2) 3]; % GoogLeNet requires RGB images

% Function to Convert Grayscale to RGB

convertToRGB = @(img) cat(3, img, img, img);


% Apply transformation during image loading

augimdsTrain = augmentedImageDatastore([224 224 3], imdsTrain, ...

   'DataAugmentation', imageAugmenter, 'ColorPreprocessing', 'gray2rgb');


augimdsVal = augmentedImageDatastore([224 224 3], imdsVal, ...

   'ColorPreprocessing', 'gray2rgb');



%% Step 4: Define Training Options

options = trainingOptions('adam', ...

   'MaxEpochs', 20, ... % Faster training

   'MiniBatchSize', 32, ...

   'InitialLearnRate', 0.0001, ... % Small learning rate for fine-tuning
```

```
    'Shuffle', 'every-epoch', ...
    'ValidationData', augimdsVal, ...
    'Plots', 'training-progress', ...
    'Verbose', false);

%% Step 5: Train the Transfer Learning Model
netTransfer = trainNetwork(augimdsTrain, lgraph, options);

%% % Load ResNet50
net = resnet50;
inputSize = net.Layers(1).InputSize; % Get input size

% Convert to Layer Graph
lgraph = layerGraph(net);

% Display all layers (to confirm names)
disp({lgraph.Layers.Name}');

% Find the correct names of the layers to replace
analyzeNetwork(net);
%% % Create new layers with unique names
newFC = fullyConnectedLayer(12, 'Name', 'new_fc', 'WeightLearnRateFactor', 10, 'BiasLearnRateFactor', 10);
newSoftmax = softmaxLayer('Name', 'new_softmax');
newClassOutput = classificationLayer('Name', 'new_output');

% Replace old layers
lgraph = replaceLayer(lgraph, "fc1000", newFC);  % Fully Connected
lgraph = replaceLayer(lgraph, "fc1000_softmax", newSoftmax);  % Softmax
lgraph = replaceLayer(lgraph, "ClassificationLayer_fc1000", newClassOutput);  % Classification
```

```
%% % Advanced Data Augmentation
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection', true, ... % Flip images horizontally
    'RandRotation', [-20 20], ... % Increase rotation range
    'RandScale', [0.7 1.3], ... % Increase scale variation
    'RandXTranslation', [-10 10], ...
    'RandYTranslation', [-10 10]);



% Convert Grayscale to RGB
convertToRGB = @(img) cat(3, img, img, img);

% Apply augmentation to training images
augimdsTrain = augmentedImageDatastore([224 224 3], imdsTrain, ...
    'DataAugmentation', imageAugmenter, 'ColorPreprocessing', 'gray2rgb');

augimdsVal = augmentedImageDatastore([224 224 3], imdsVal, ...
    'ColorPreprocessing', 'gray2rgb');

disp('Data Augmentation Applied');

%%
 options = trainingOptions('adam', ...
    'MaxEpochs', 50, ... % Increase from 30 to 50
    'MiniBatchSize', 32, ...
    'InitialLearnRate', 0.0005, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.5, ...
    'LearnRateDropPeriod', 10, ...
```

```
    'Shuffle', 'every-epoch', ...

    'ValidationData', augimdsVal, ...

    'Plots', 'training-progress', ...

    'Verbose', false, ...

    'ExecutionEnvironment', 'cpu');


 %% % Train ResNet50

netTransfer = trainNetwork(augimdsTrain, lgraph, options);

%% % Predict on validation data

YPred = classify(netTransfer, imdsVal);

YValidation = imdsVal.Labels;


% Calculate accuracy

transferAccuracy = sum(YPred == YValidation) / numel(YValidation);

fprintf('ResNet50 Validation Accuracy: %.2f%%\n', transferAccuracy * 100);


% Display confusion matrix

figure;

confusionchart(YValidation, YPred);

title('ResNet50 Confusion Matrix');


%% YPred = classify(net, imdsVal);

YValidation = imdsVal.Labels;

confusionchart(YValidation, YPred);

title('Baseline CNN Confusion Matrix');

%% % After training, get accuracy/loss data

trainingInfo = net.TrainingHistory;

figure;

plot([trainingInfo.TrainingAccuracy], 'LineWidth', 2); hold on;
```

```
plot([trainingInfo.ValidationAccuracy], 'LineWidth', 2);

legend('Training Accuracy', 'Validation Accuracy');

xlabel('Iteration');

ylabel('Accuracy');

title('Training Curve - Best Hyperparameter Configuration');
```