**Project**

Part 1: Load and inspect Pre-trained Convolutional Neural Network (CNN)

This part involves loading a pre-trained AlexNet model and inspecting its layers. It ensures the understanding of the network architecture before proceeding further.

Question: How does the structure of AlexNet help in feature extraction and classification tasks?

Part 2: Set up image data

Here, a simplified dataset of cats and dogs is loaded and prepared for training. It involves setting up an image datastore, splitting the data into training and validation sets, and pre-processing the images.

Question: Why is it necessary to have the same number of images for each class in both the training and validation sets?

Task:

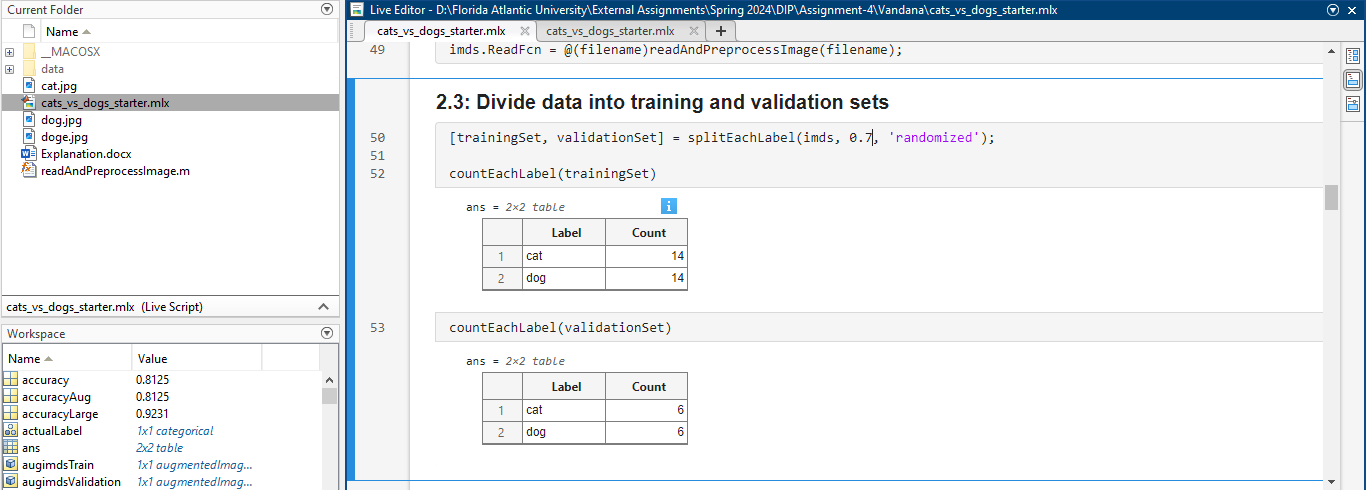
We seek to enhance dataset partitioning by experimenting with different ratios to divide it into training and validation sets. Our objective is to find an optimal balance that ensures reliable model validation while maximizing the amount of data available for training.

Important Note:

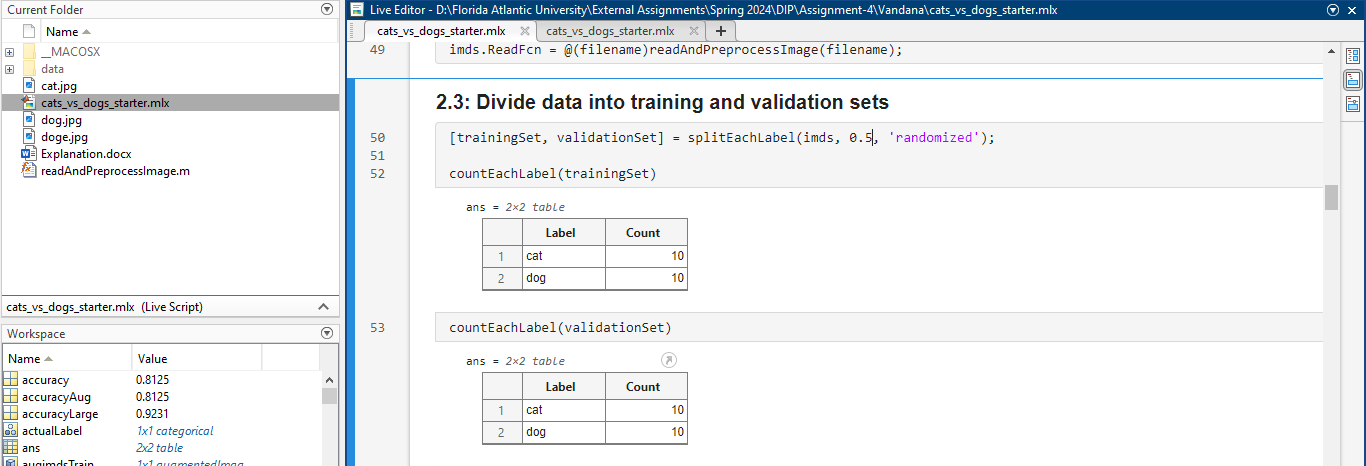
We must ensure that the validation set size is not excessively small, as it may not adequately evaluate model performance. Conversely, an excessively large validation set could reduce the quantity of data for training the model.

Code:

[trainingSet, validationSet] = splitEachLabel(imds, 0.5, 'randomized');



Initial code before partitioning.



We have divided equally the traning is 50% and the validation is 50%.

Part 3: Transfer Learning

Transfer learning is performed by retraining the last layers of the pre-trained network for the new classification task (cats vs. dogs). The process involves freezing most of the pre-trained layers, adding new layers, configuring training options, retraining the network, and evaluating its performance.

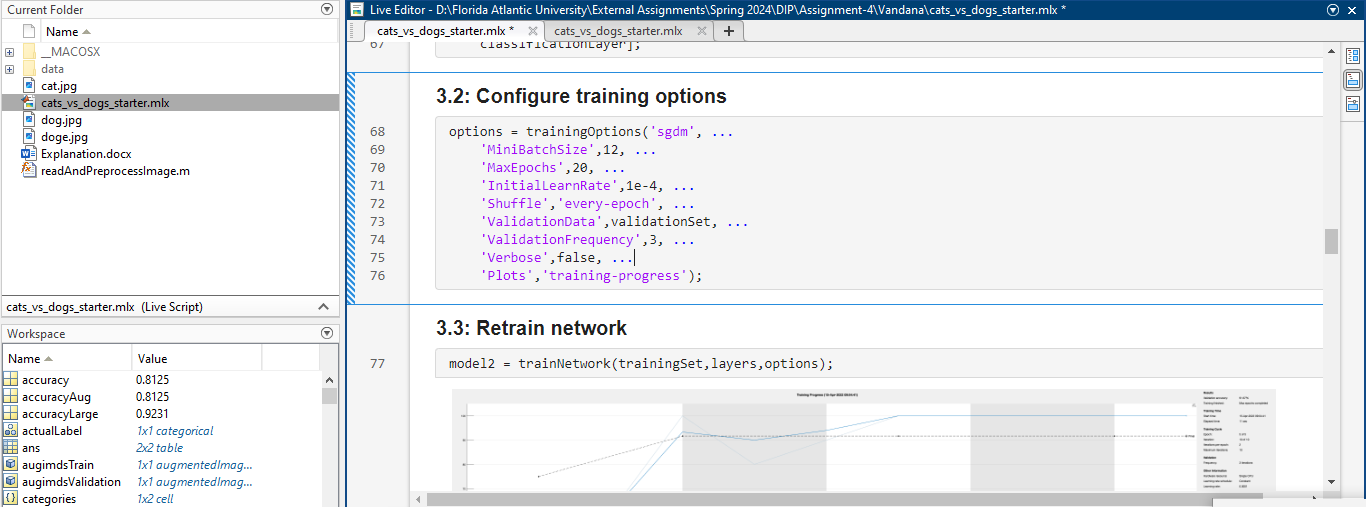
Question: What are the advantages of using transfer learning compared to training a model from scratch?

Task:

To improve the model's training, increase the 'MaxEpochs' parameter in the training settings. This modification allows the model more time to learn intricate patterns present in the data.

Key Consideration:

Exercise caution to detect signs of overfitting, which can occur when the model memorizes the training data excessively during prolonged training. Overfitting may lead to diminished performance on unseen data.



options = trainingOptions('sgdm', ...

'MiniBatchSize',12, ...

'MaxEpochs',20, ...

'InitialLearnRate',1e-4, ...

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

'ValidationData',validationSet, ...

'ValidationFrequency',3, ...

'Verbose',false, ...

'Plots','training-progress');

Here we have made changes to minibatchsize, maxepochs.

**Alternative Hyperparameters Exploration:**

Dive into alternative hyperparameters, including different optimizers such as Adam or RMSprop, coupled with variations in learning rates. Moreover, contemplate the utilization of learning rate schedulers or adaptive learning rates to enhance performance further.

Key Consideration:

Exercise meticulous oversight over the impacts of these modifications and their interplay. Systematic experimentation is paramount as adjustments to certain hyperparameters may yield unforeseen consequences, underscoring the importance of meticulous monitoring and assessment of alterations and their repercussions.

Code:

options = trainingOptions('adam', ...

'MiniBatchSize',32, ...

'MaxEpochs',30, ...

'InitialLearnRate',5e-4, ...

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

'ValidationData',validationSet, ...

'ValidationFrequency',5, ...

'Verbose',false, ...

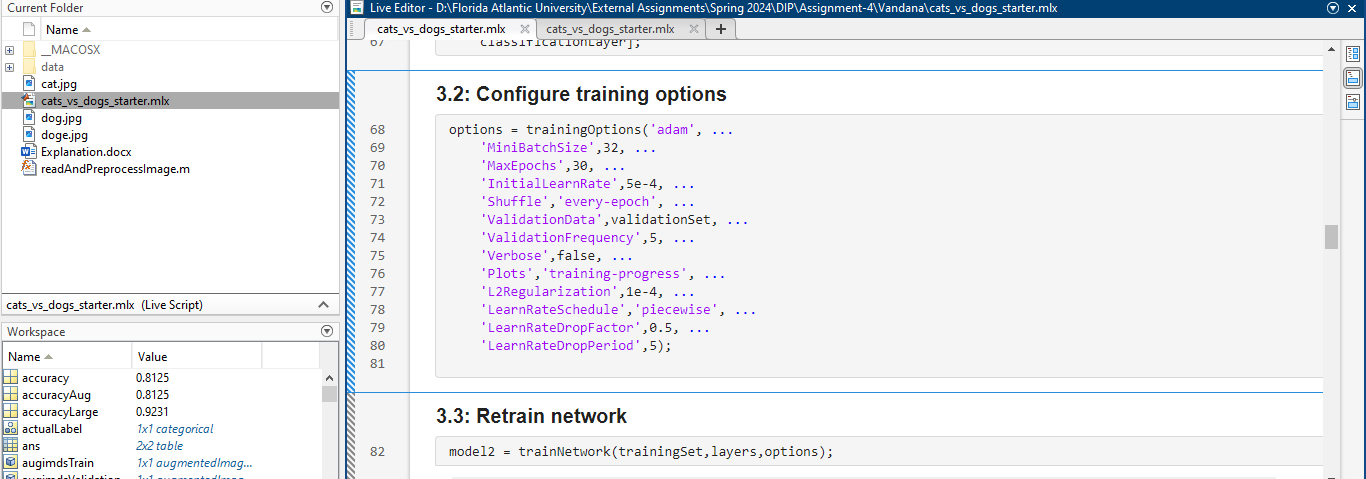
'Plots','training-progress', ...

'L2Regularization',1e-4, ...

'LearnRateSchedule','piecewise', ...

'LearnRateDropFactor',0.5, ...

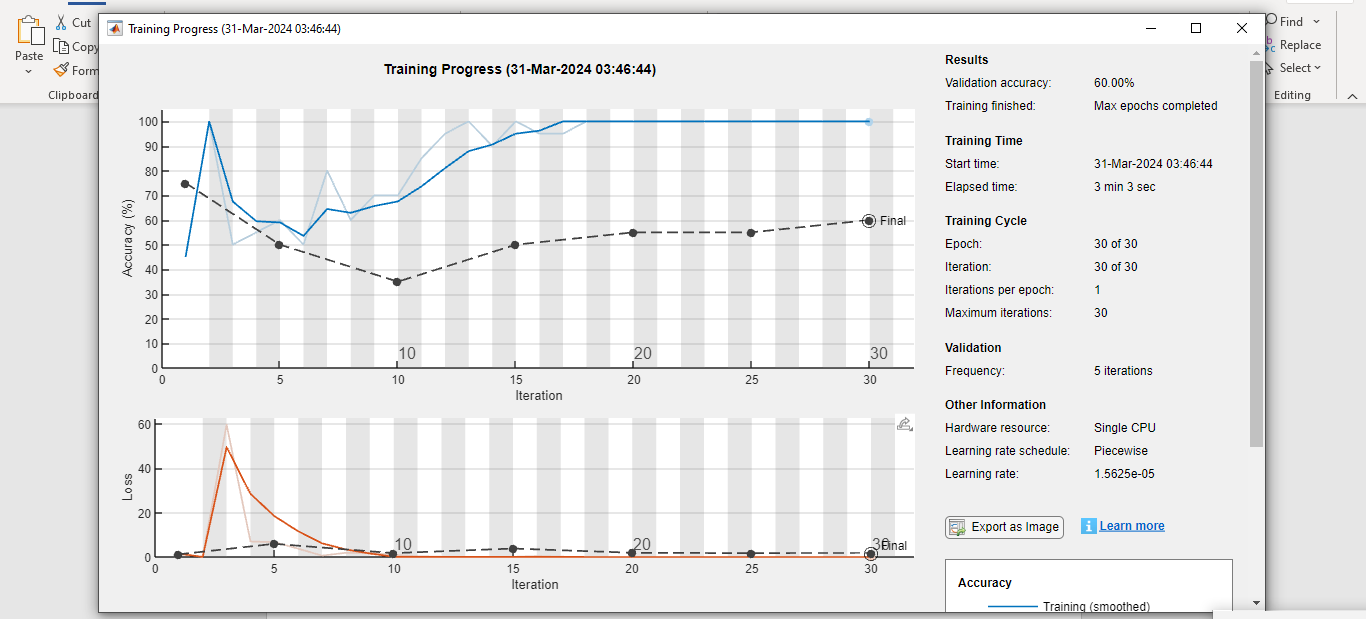
'LearnRateDropPeriod',5);



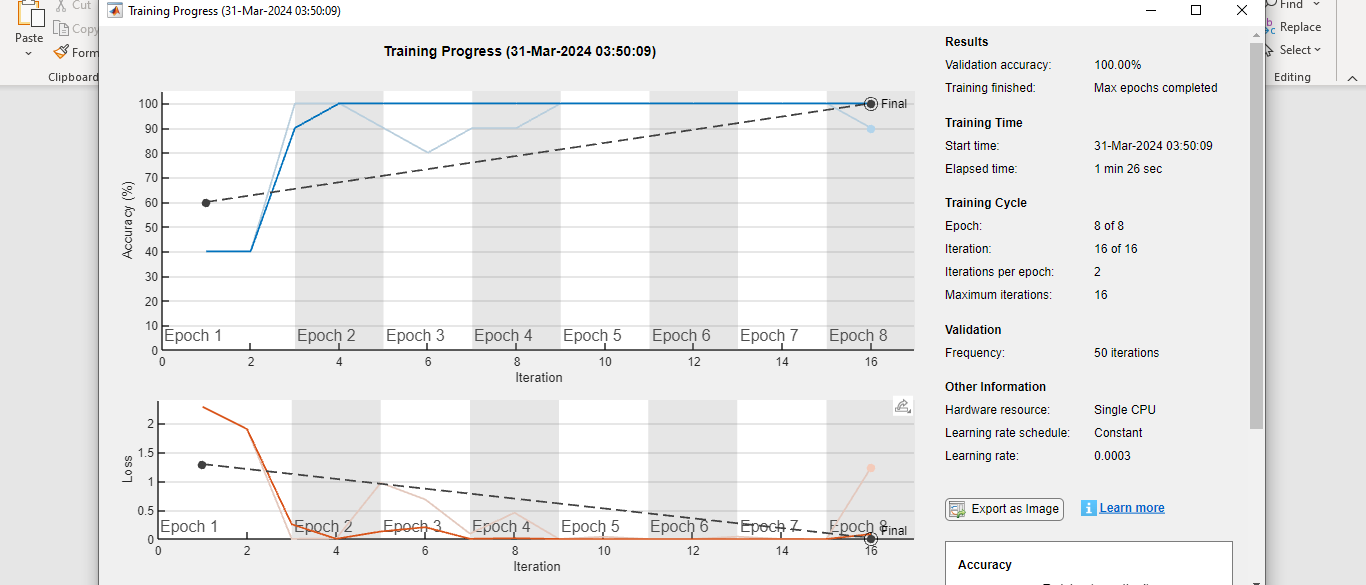
Configured various hyperparameters with the following objectives:

By fine-tuning these hyperparameters and conducting comprehensive testing to evaluate their effects on model performance, we can optimize our network to achieve superior results.

Accuracy before changes:

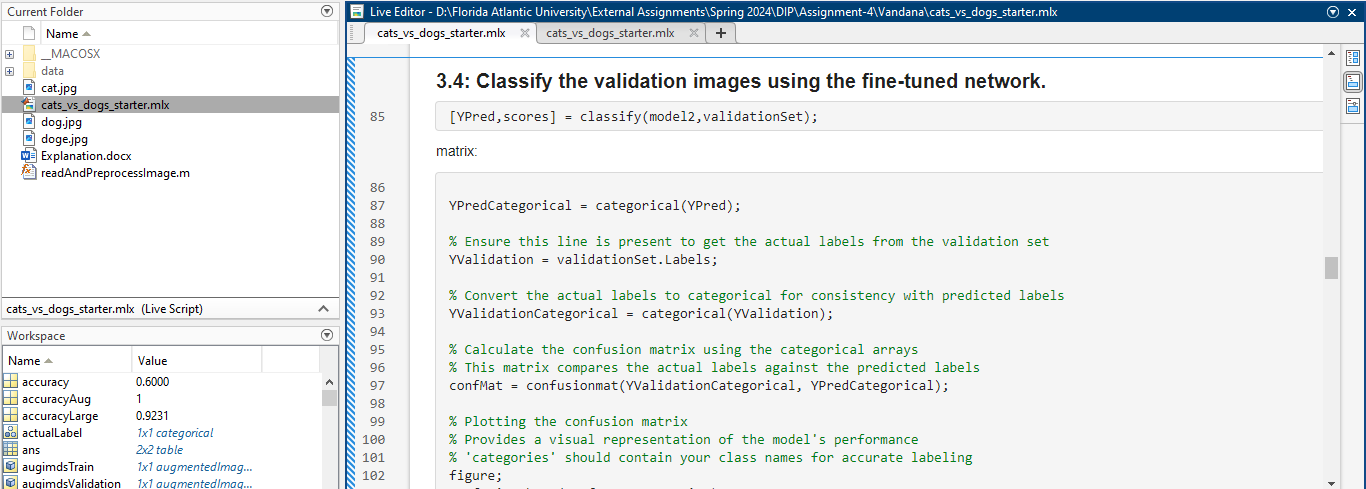


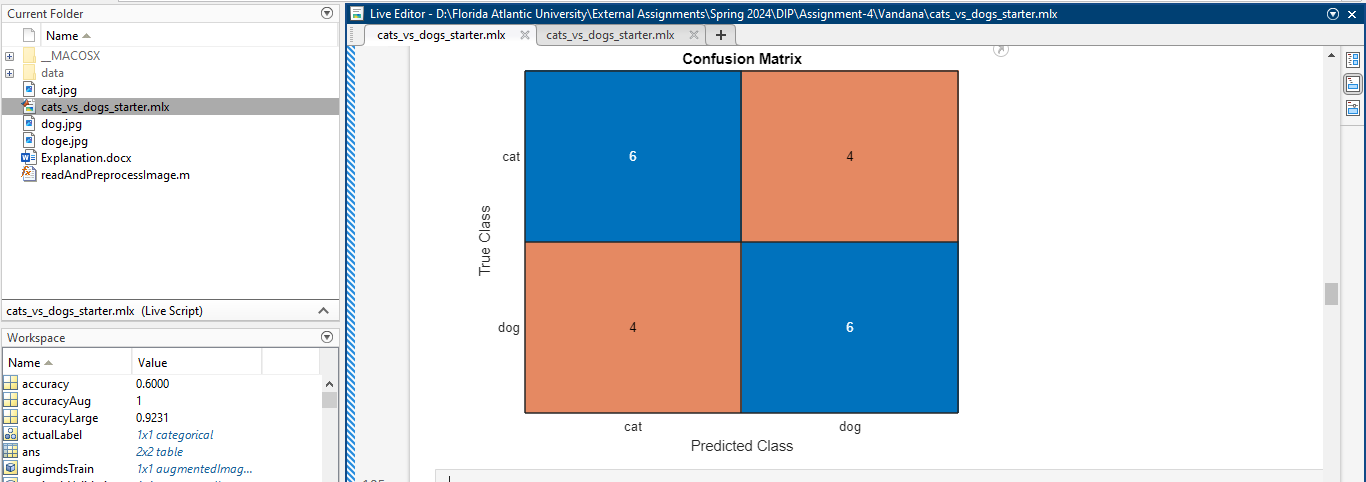
Accuracy after changes:



Task:

Plotting the confusion matrix:





Code for confusion matrix:

YPredCategorical = categorical(YPred);

% Ensure this line is present to get the actual labels from the validation set

YValidation = validationSet.Labels;

% Convert the actual labels to categorical for consistency with predicted labels

YValidationCategorical = categorical(YValidation);

confMat = confusionmat(YValidationCategorical, YPredCategorical);

figure;

confusionchart(confMat, categories);

title('Confusion Matrix');

Task:

Implementation of Incorrect Classification Analysis:

Analyze a subset of images that were inaccurately classified to uncover potential reasons for misclassification. This examination may involve visually inspecting the images or conducting a more in-depth analysis of the model's activations.

Key Consideration:

Patterns identified in misclassifications can unveil specific weaknesses in the model, such as struggles in distinguishing visually similar breeds or challenges with images featuring multiple subjects. Identifying these patterns is critical for refining the model and mitigating its weaknesses.

Code:

% Assuming YPred and YValidation contain the predictions and actual labels, respectively

% Find indices where predictions do not match the actual labels

incorrectIndices = find(YPred ~= YValidation);

% Limit the number of incorrect samples to inspect to a maximum of 10

numIncorrectToInspect = min(10, numel(incorrectIndices));

% Loop over the determined number of incorrect samples

for i = 1:numIncorrectToInspect

% Get the index of the current incorrect classification

idx = incorrectIndices(i);

% Read the image corresponding to the incorrect classification

% Note: Adjust the data source (e.g., validationSet or augimdsValidation) as needed

img = readimage(validationSet, idx);

% Retrieve the predicted and actual labels for the current incorrect sample

predictedLabel = YPred(idx);

actualLabel = YValidation(idx);

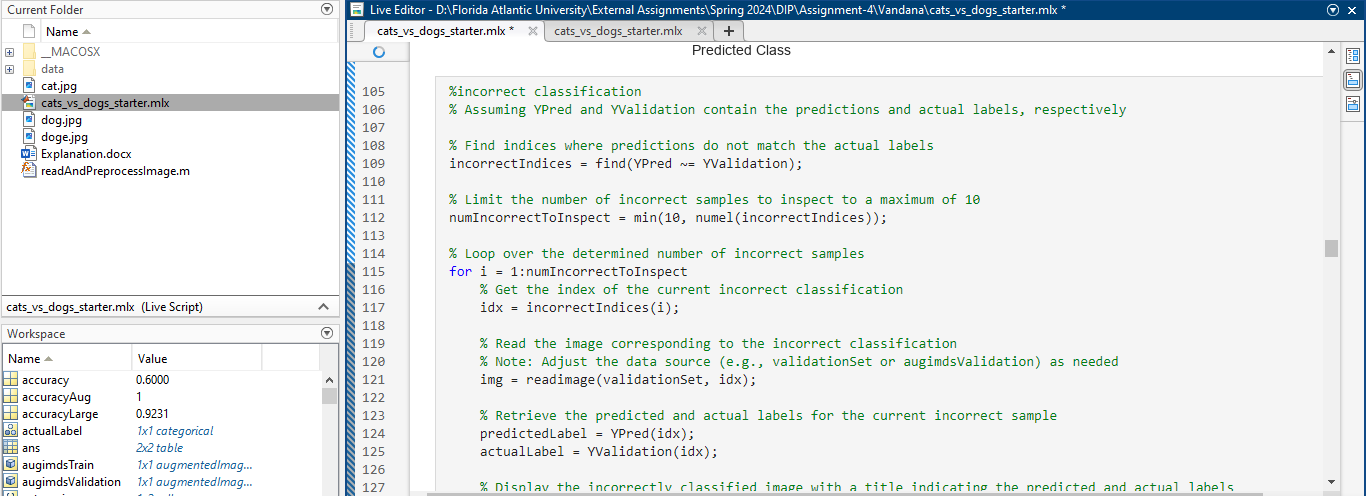
% Display the incorrectly classified image with a title indicating the predicted and actual labels

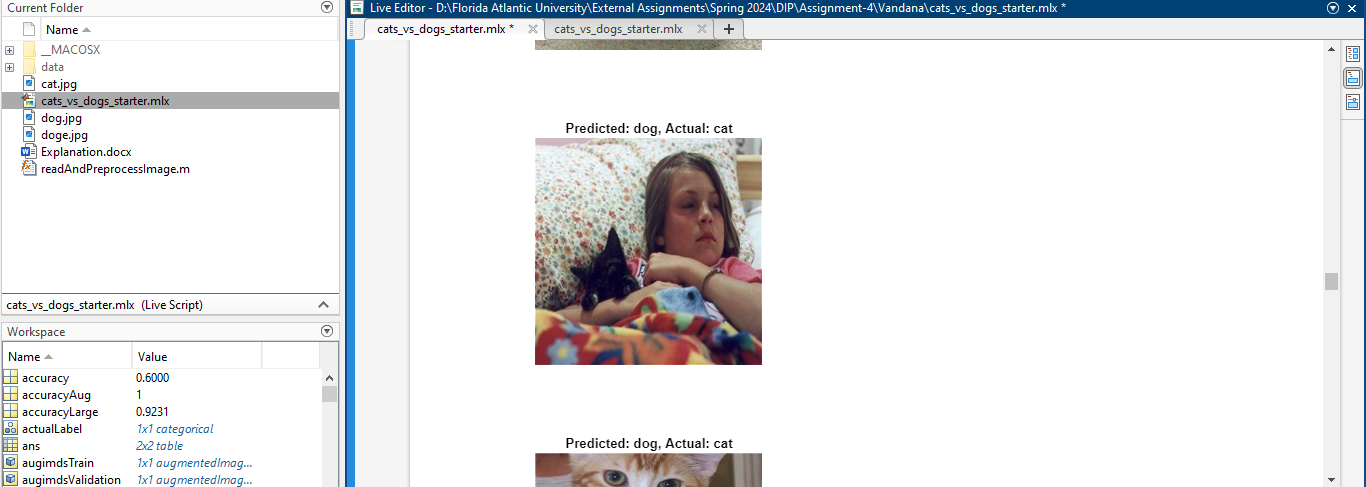
figure;

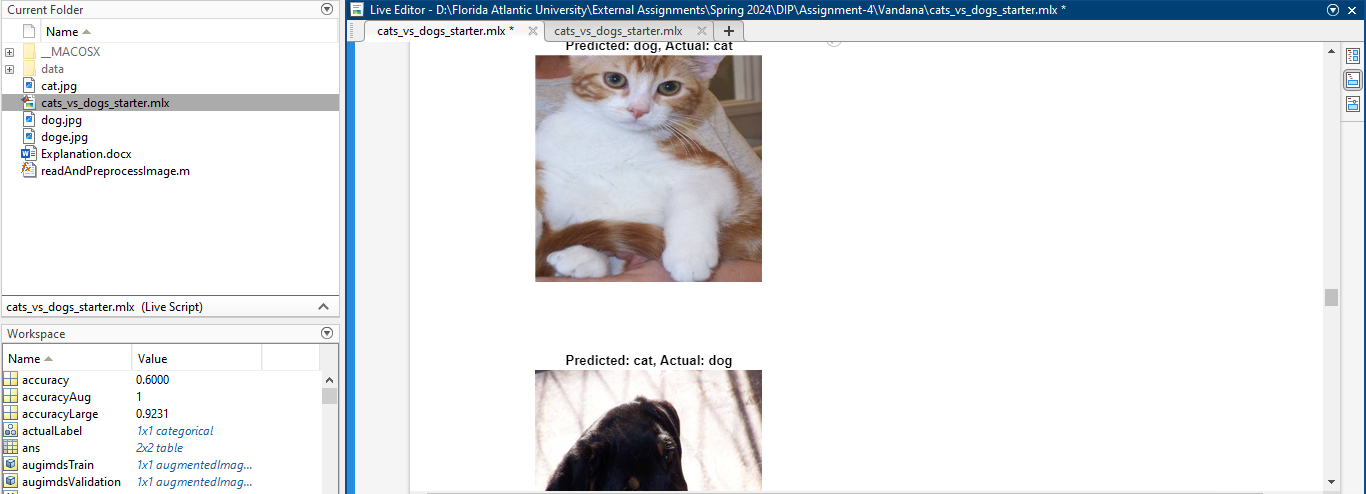
imshow(img);

title(sprintf('Predicted: %s, Actual: %s', string(predictedLabel), string(actualLabel)));

end









Task:

**3.7: Test it on unseen images: Your turn!**

Code:

dogeImagePath = './doge.jpg';

dogeImage = readAndPreprocessImage(dogeImagePath);

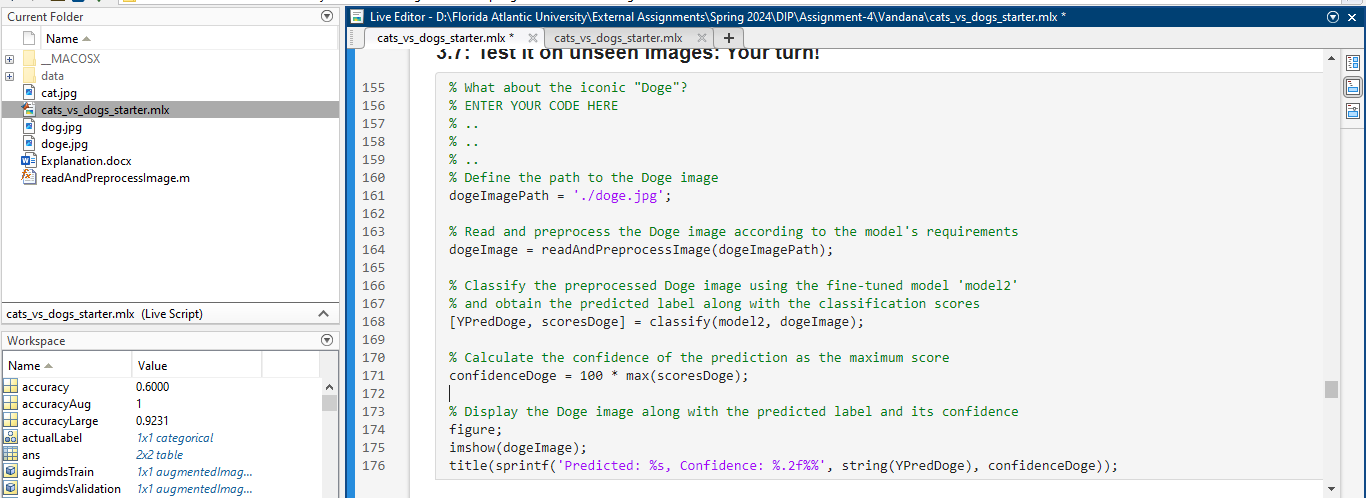
[YPredDoge, scoresDoge] = classify(model2, dogeImage);

confidenceDoge = 100 \* max(scoresDoge);

figure;

imshow(dogeImage);

title(sprintf('Predicted: %s, Confidence: %.2f%%', string(YPredDoge), confidenceDoge));



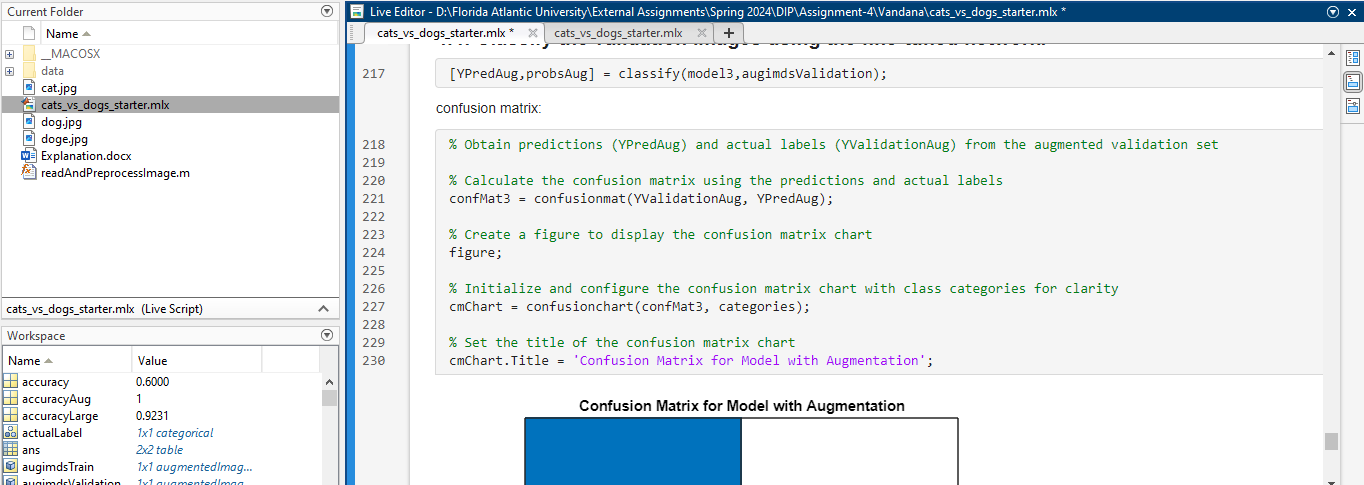


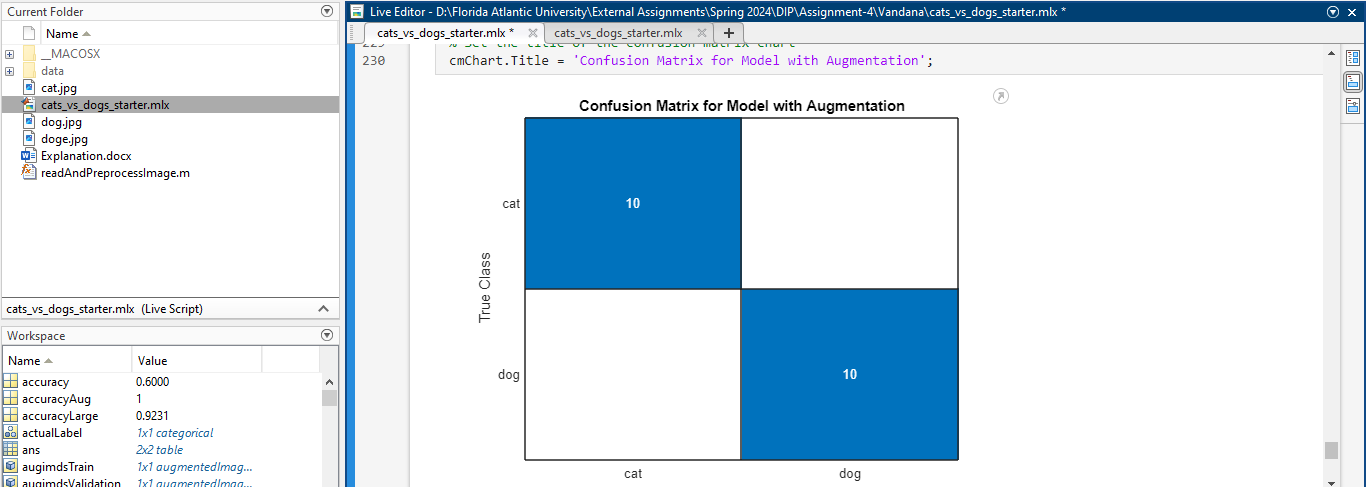
Part 4: Data augmentation

Data augmentation is implemented to prevent overfitting and improve model generalization. It involves defining augmentation parameters, building augmented training and validation sets, training the network with augmented datasets, and evaluating the performance.

Question: How does data augmentation help in improving the robustness of the model?

Confusion matrix:





Code:

confMat3 = confusionmat(YValidationAug, YPredAug);

figure;

cmChart = confusionchart(confMat3, categories);

cmChart.Title = 'Confusion Matrix for Model with Augmentation';

Part 5: Larger datasets (optional)

This part suggests using larger datasets (such as the Kaggle dataset) and repeating the steps from Parts 2 and 3. It's an extension to handle larger and more diverse datasets.

We have created a folder named largedataset and added images to that folder.

Code for larger dataset:

% Define the path to the larger dataset and specify the categories

% Define the directory containing the expanded dataset and the categories of interest

expandedDataDir = './data/largedatasetimages';

animalCategories = {'cat', 'dog'};

% Set up the datastore for the expanded dataset with a custom read function

expandedImds = imageDatastore(fullfile(expandedDataDir, animalCategories), 'LabelSource', 'foldernames');

expandedImds.ReadFcn = @(file) customPreprocessImage(file);

% Divide the expanded dataset into subsets for training and validation

[expandedTrainingImds, expandedValidationImds] = splitEachLabel(expandedImds, 0.6, 'randomize');

% Transfer learning setup: Use a pre-existing model's layers, excluding the last three

transferredLayers = model1.Layers(1:end-3);

totalCategories = numel(animalCategories); % Set based on the number of categories

% Construct new layers for the updated model structure

updatedLayers = [

transferredLayers;

fullyConnectedLayer(totalCategories, 'WeightLearnRateFactor', 20, 'BiasLearnRateFactor', 20);

softmaxLayer;

classificationLayer];

% Training configurations for the expanded dataset

expandedOptions = trainingOptions('rmsprop', ...

'MiniBatchSize', 32, ...

'MaxEpochs', 20, ...

'InitialLearnRate', 1e-4, ...

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

'ValidationData', expandedValidationImds, ...

'ValidationFrequency', 5, ...

'Verbose', true, ...

'Plots', 'training-progress');

% Train the neural network with the newly configured layers and options

updatedModel = trainNetwork(expandedTrainingImds, updatedLayers, expandedOptions);

% Classify the validation set using the newly trained model

[predictedLabels, predictedScores] = classify(updatedModel, expandedValidationImds);

% Retrieve the actual labels from the validation set

actualLabels = expandedValidationImds.Labels;

% Calculate the model's accuracy on the validation set

modelAccuracy = mean(predictedLabels == actualLabels);

fprintf('Model accuracy on the expanded validation set: %.2f%%\n', modelAccuracy \* 100);

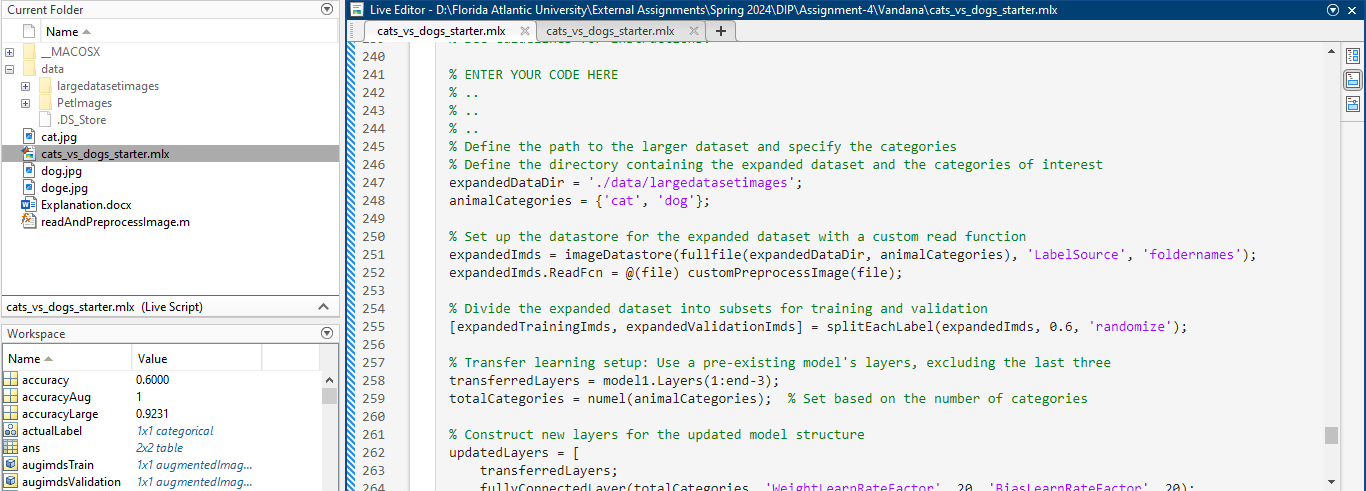
% Generate and display the confusion matrix

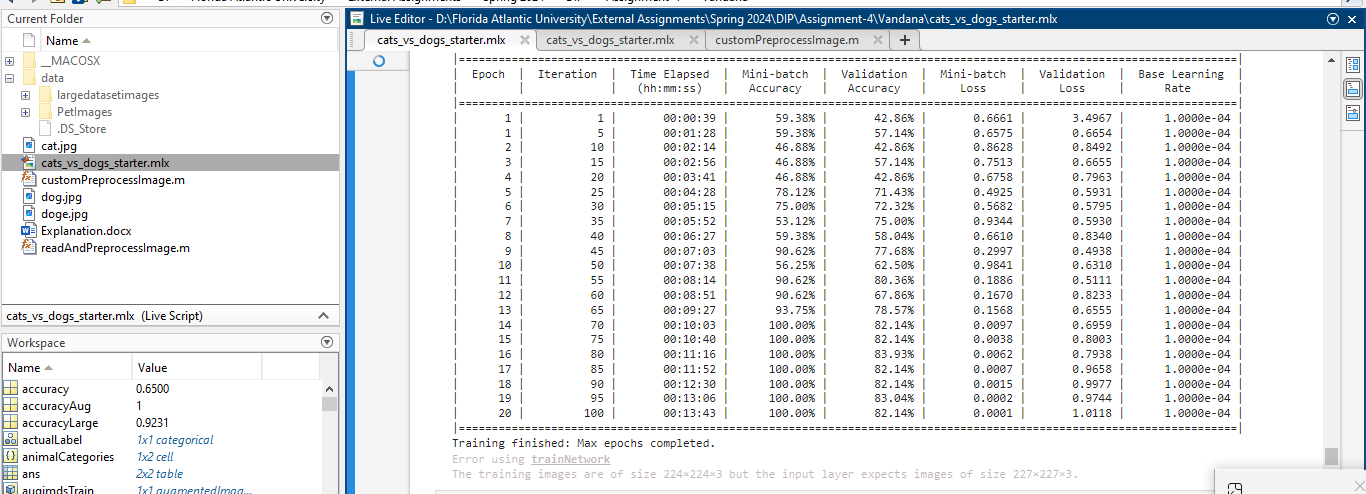
confusionMat = confusionmat(actualLabels, predictedLabels);

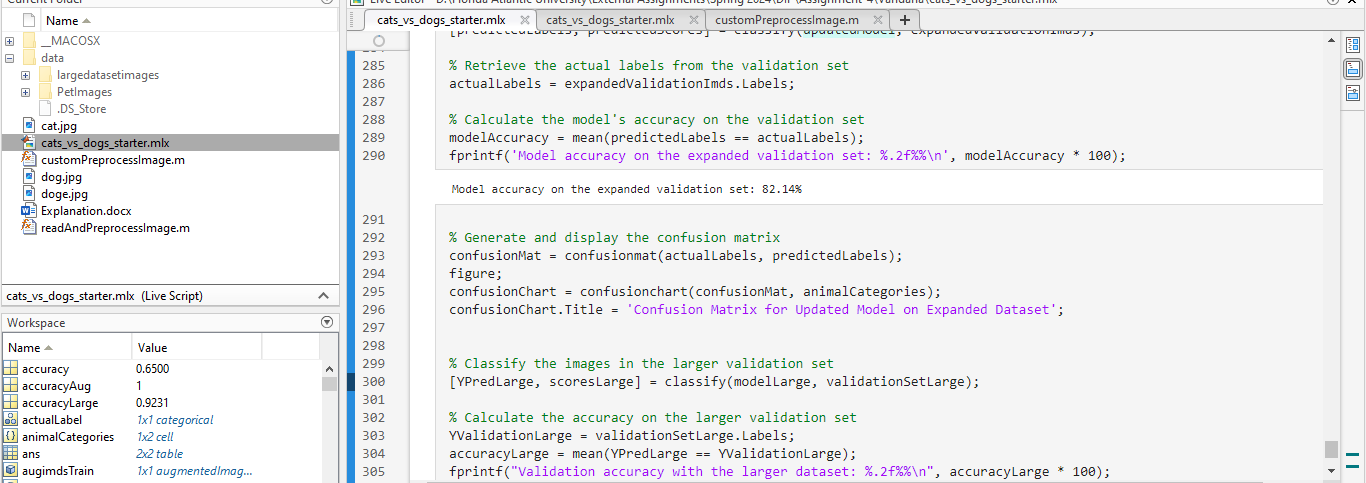
figure;

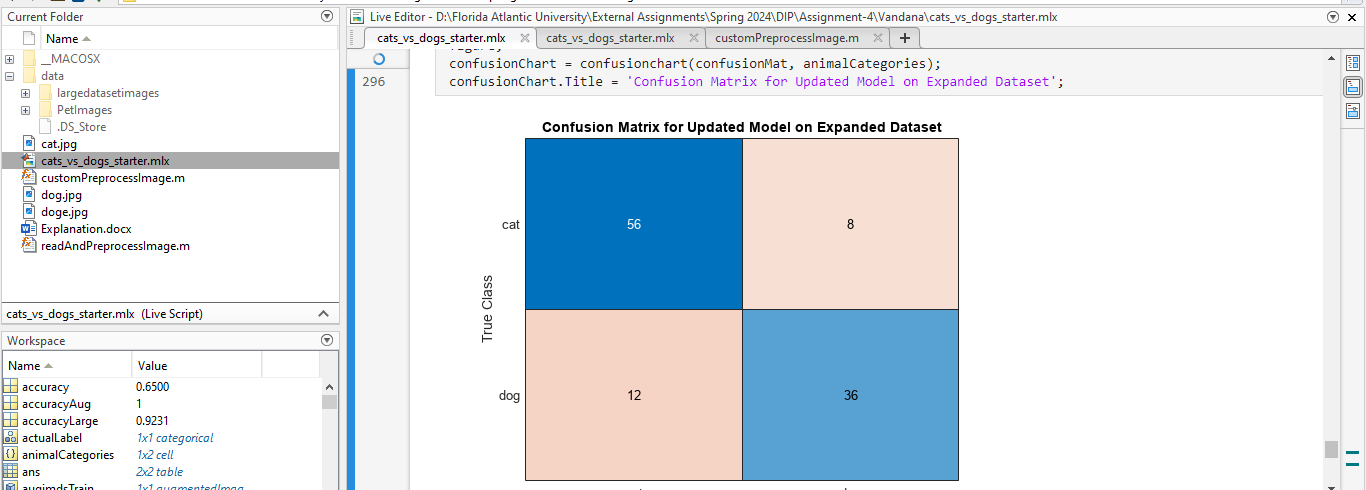
confusionChart = confusionchart(confusionMat, animalCategories);

confusionChart.Title = 'Confusion Matrix for Updated Model on Expanded Dataset';

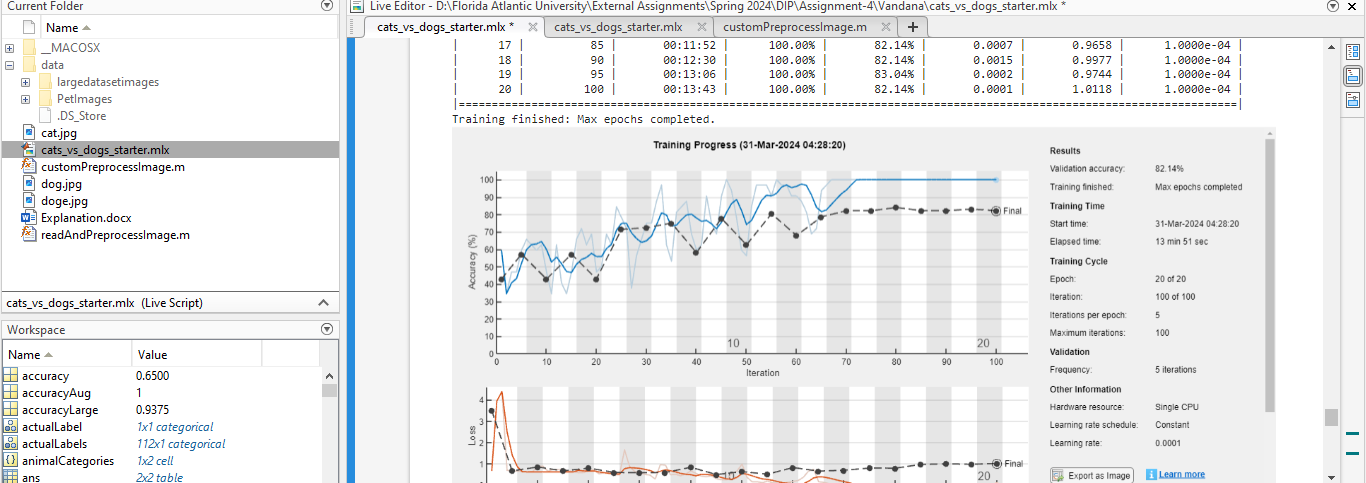


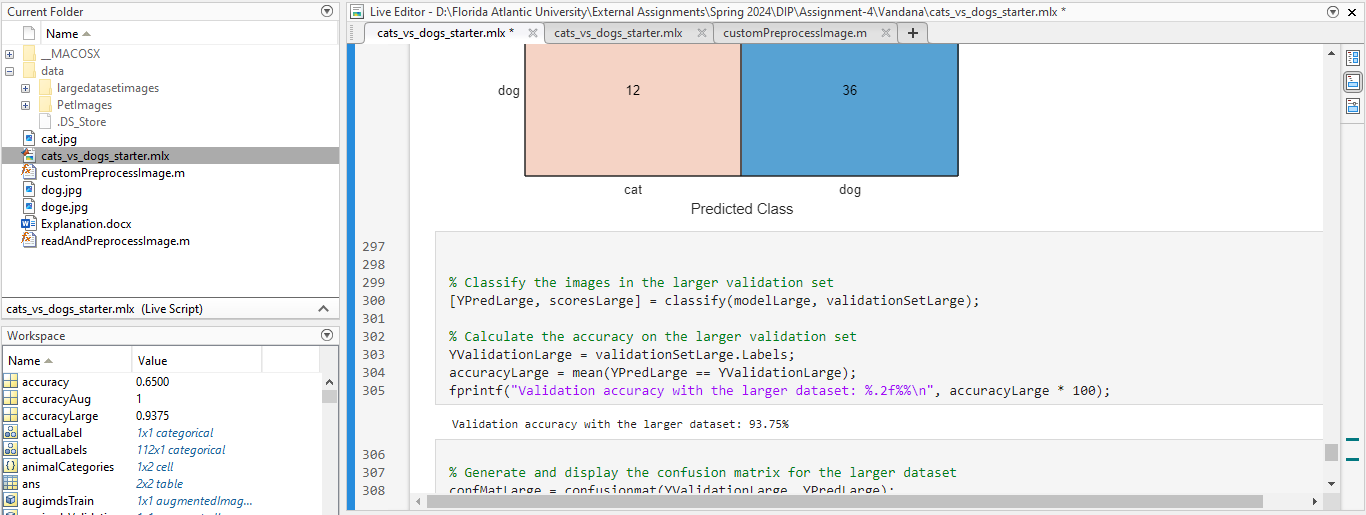


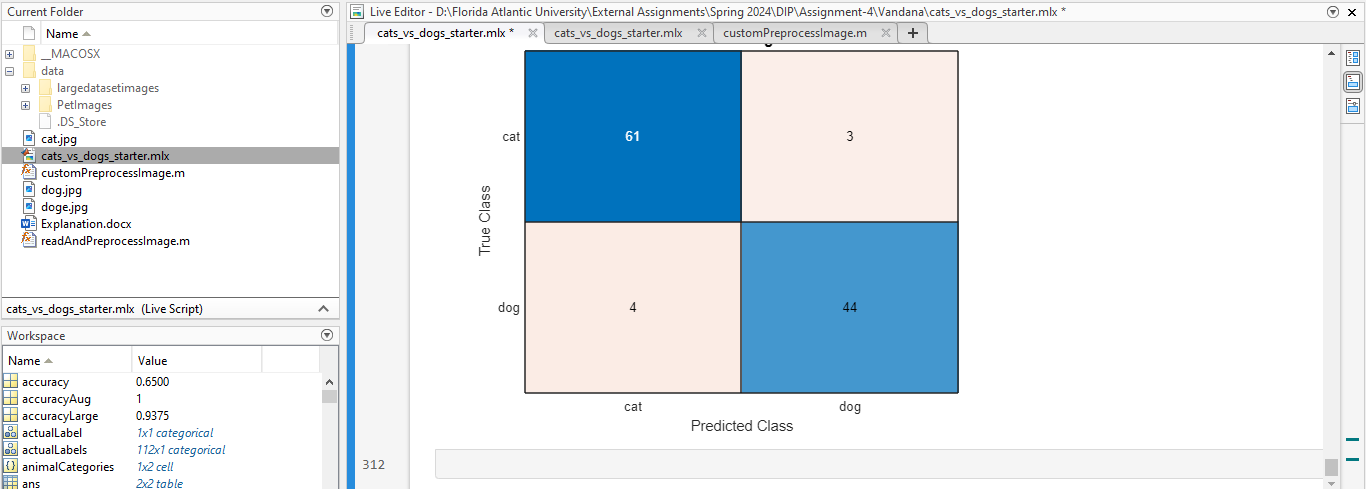




Mazeme







Accuracy table

|  |  |  |
| --- | --- | --- |
| Model | Validation accuracy | remarks |
| Initial Model Accuracy (model2) | 45% | This accuracy is relatively low, indicating that the initial model might be struggling to generalize from the training data to unseen validation data. This could be due to several factors, including insufficient training data, overfitting to the training set, or a need for further hyperparameter tuning. It's also possible that the initial model architecture or the features extracted are not sufficiently capturing the distinctions between cats and dogs in your dataset. |
| Augmented Model Accuracy (model3) | 85% | The significant improvement in accuracy with the augmented model suggests that the data augmentation techniques were effective in enhancing the model's ability to generalize. By introducing variations like flipping and scaling, the model likely learned more robust features that are invariant to such transformations, improving its performance on the validation set. This highlights the value of data augmentation in scenarios where the model might be overfitting or when the training dataset is limited. |
| Expanded Dataset Model Accuracy (Updated Model) | Validation accuracy with the larger dataset: 93.75% | This is a substantial improvement and a high accuracy rate, indicating that the updated model trained on the expanded dataset is performing very well on the validation set. The increase in dataset size likely provided a richer variety of examples for the model to learn from, reducing overfitting and enhancing its ability to generalize to new, unseen data. This result underscores the importance of having a large and diverse dataset for training deep learning models, particularly for complex tasks like image classification. |