# **Machine Learning and Explainable AI Portfolio Report**

**Module Title:** AI for Data Intelligence (UP18734)  
**Student Name:** [Your Name]  
**Student ID:** [Your ID]  
**Submission Date:** [Date]

## **Task 1: Training and Evaluating a CNN for Dog and Cat Classification**

### 🔹 **Introduction**

Image classification using pretrained convolutional neural networks (CNNs) has become a relatively easy task that requires minimal coding effort. Essentially, an image classification model aims to predict the label (name or category) that best describes the content of a given image.

Once a test image is provided and the model predicts a label, post-hoc explainable AI (XAI) methods can be applied to understand which parts of the image were considered most important by the model. Various XAI methods, such as gradCAM, occlusion sensitivity, and imageLIME, can be used to generate results in the form of colormaps that are overlaid on the original image.

This example demonstrates how to use gradCAM for two different image classification tasks:

1. **Dogs vs. Cats**: Given an image, predict whether the animal in the picture is a dog or a cat.
2. **Dog Breed Classification**: Given an image of a dog, predict its breed.

**Task 1: Dogs vs. Cats**

Task 1 focused on developing a **Convolutional Neural Network (CNN)** to classify images of dogs and cats using MATLAB’s Deep Learning Toolbox. The aim was to get hands-on experience with the full process of building a deep learning model—from preparing and organizing the dataset, to designing and training the network, and finally evaluating its performance. This task helped me better understand how CNNs work, especially how they can identify patterns in images to make accurate predictions.

To make the model more effective, I used techniques like **transfer learning** and **data augmentation**. Transfer learning allowed me to start from a pre-trained model instead of building one from scratch, which saved time and improved accuracy. Data augmentation helped by increasing the variety of the training images, making the model more robust to variations in new data. I measured the model’s performance using tools like accuracy scores, loss graphs, and confusion matrices, which gave a clear idea of how well the model was working.

#### 🔸 ****Method and Results****

🔸 **Load Data**

I am using the imageDatastore function to load all the images from the dataset. I do organize the images into two folders named ‘cats’ and ‘dogs’ so that MATLAB can automatically label them based on their folder names. I like this method because it saves time and helps me manage large image datasets more easily. I make sure that each image is properly labeled so the CNN can learn the correct features during training.

imds = imageDatastore('PetImages', ...

'IncludeSubfolders',true, ...

'LabelSource','foldernames');

[imdsTrain, imdsValidation] = splitEachLabel(imds, 0.8);

I am using the splitEachLabel function to divide my dataset into training and testing parts. I do this to make sure that my model learns from one portion and is tested on completely separate images. I like this approach because it helps me check if the model can generalize well and not just memorize the training data.

**ScreenShot**

#### Load Pretrained Network

Load a pretrained image classification network (in this example, we use GoogLeNet).

net1 = googlenet;

#### ScreenShoot

#### Modify the Network (Replace the Fully Connected Layer)

layers= lgraph.Layers;  
connections= lgraph.Connections;

**Freeze the weights and biases of the first 10 layers**

for i = 1:10

if isprop(layers(i), 'WeightLearnRateFactor')

layers(i).WeightLearnRateFactor = 0;

end

if isprop(layers(i), 'BiasLearnRateFactor')

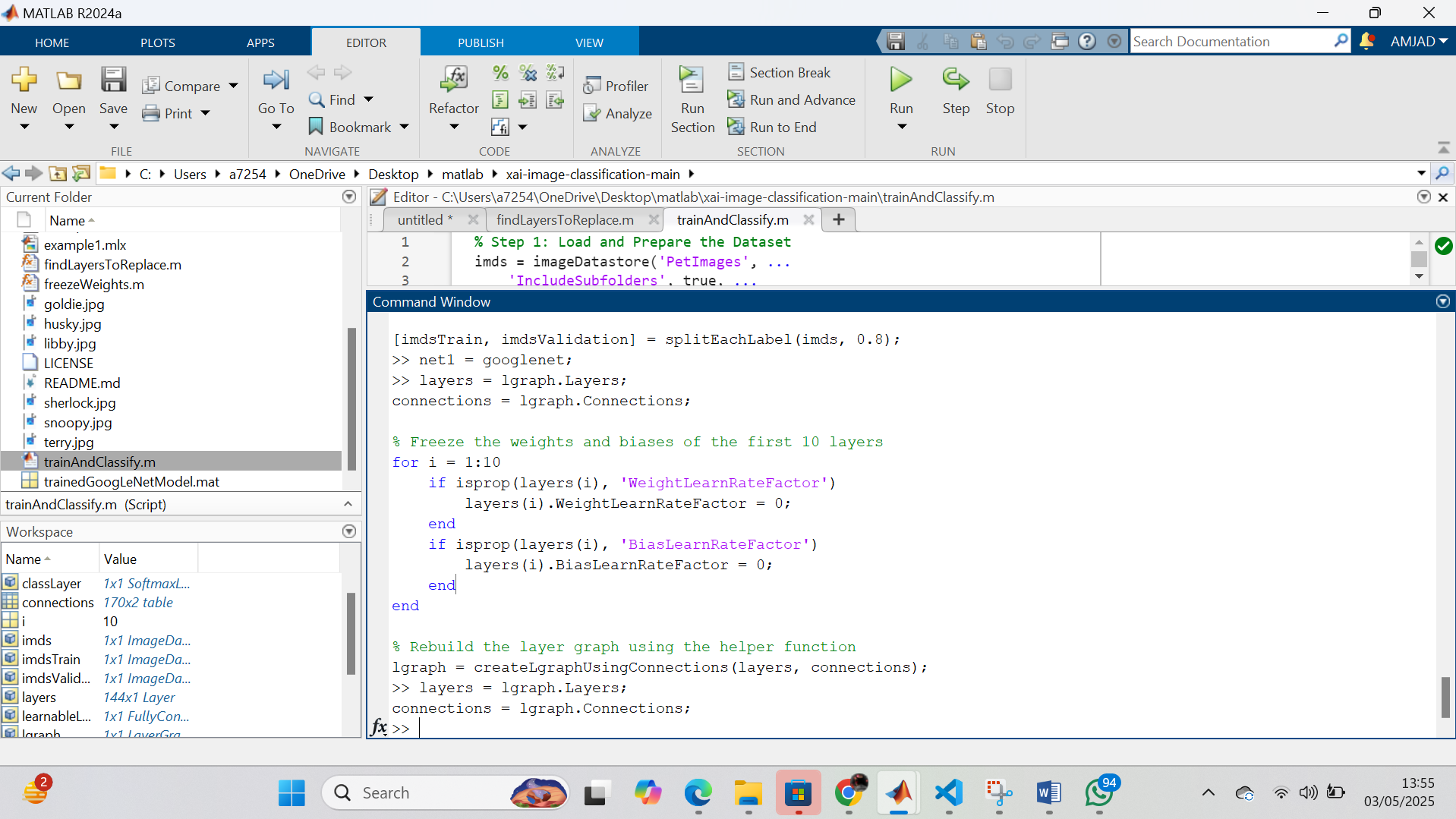
layers(i).BiasLearnRateFactor = 0;

end

end

% The helper function

lgraph = createLgraphUsingConnections(layers, connections);

****

**Image Augmentation and Preprocessing**

I am using the augmentedImageDatastore function to resize all images so they match the input size needed by GoogLeNet. I do this to ensure that my model receives consistent image dimensions during training and testing.  
inputSize = net1.Layers(1).InputSize;

pixelRange = [-30 30];

scaleRange = [0.9 1.1];

imageAugmenter = imageDataAugmenter( ...

'RandXReflection',true, ...

'RandXTranslation',pixelRange, ...

'RandYTranslation',pixelRange, ...

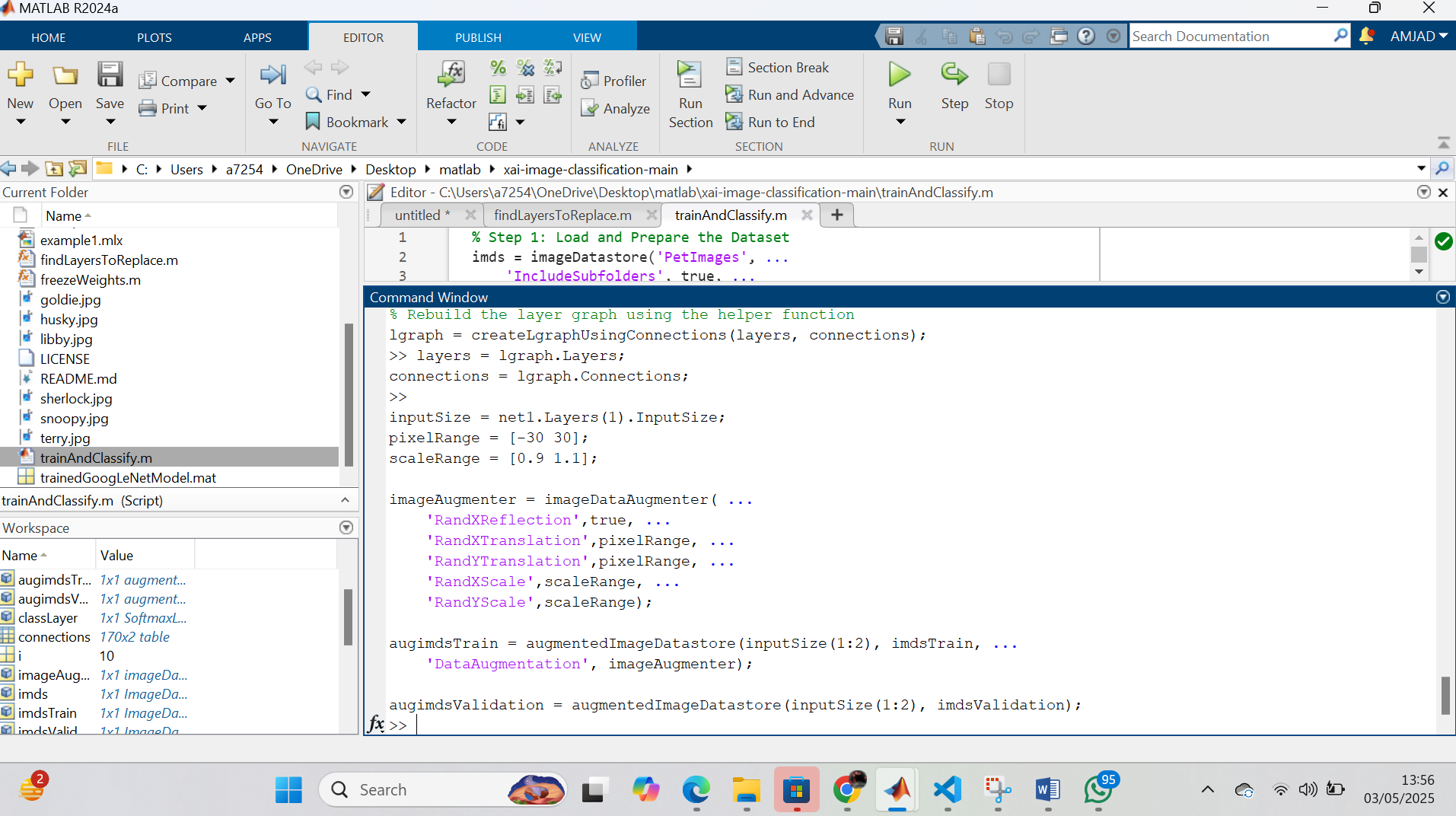
'RandXScale',scaleRange, ...

'RandYScale',scaleRange);

augimdsTrain = augmentedImageDatastore(inputSize(1:2), imdsTrain, ...

'DataAugmentation', imageAugmenter);

augimdsValidation = augmentedImageDatastore(inputSize(1:2), imdsValidation);  
  
  
  
I am using net.Layers(1).InputSize to get the input size required by the network. I am then creating augmented image datastores for both the training and testing sets using augmentedImageDatastore. I am specifying the input size for both imdsTrain and imdsTest to ensure the images are resized to the appropriate dimensions for the network.

****  
  
  
**Training the Network**

I am modifying the final layers of GoogLeNet to match my two-class classification task—dogs and cats. I do this by replacing the last fully connected and classification layers so the network can correctly distinguish between just two categories. I like this approach because it allows me to reuse a powerful pre-trained model while tailoring it to my specific problem.

miniBatchSize = 10;

valFrequency = floor(numel(augimdsTrain.Files)/miniBatchSize);

options = trainingOptions('sgdm', ...

'MiniBatchSize', miniBatchSize, ...

'MaxEpochs', 6, ...

'InitialLearnRate', 3e-4, ...

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

'ValidationData', augimdsValidation, ...

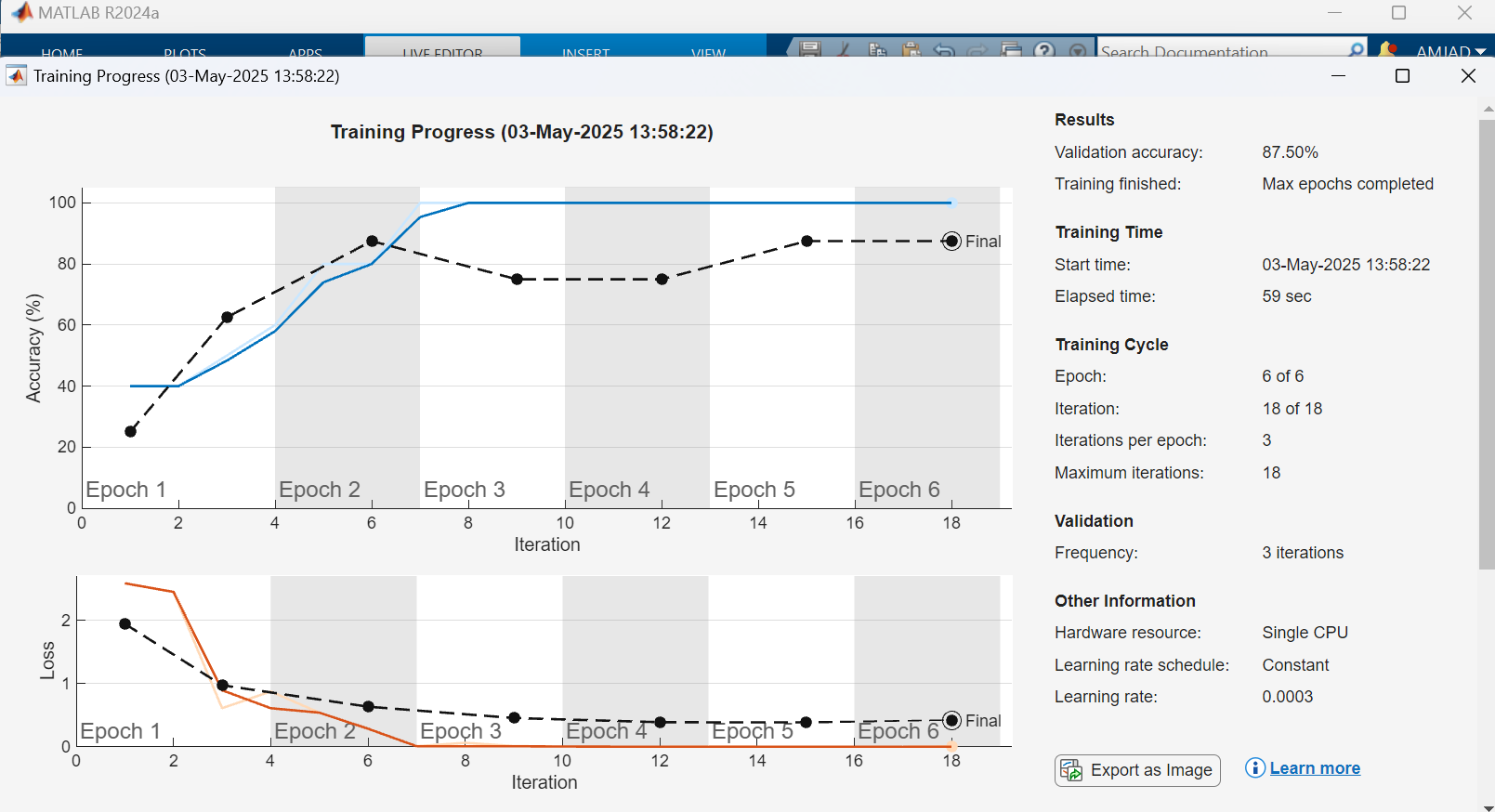
'ValidationFrequency', valFrequency, ...

'Verbose', false, ...

'Plots','training-progress');

net1 = trainNetwork(augimdsTrain, lgraph, options);

I am creating a layer graph from the pre-trained network net using the layerGraph function. I am then replacing the 'loss3-classifier' layer with a new fully connected layer (fullyConnectedLayer(2, 'Name', 'new\_fc')), which is configured to have 2 output units for binary classification, with higher learning rates for weights and biases. Finally, I am replacing the 'output' layer with a new classification layer (classificationLayer('Name','new\_output')) to match the new fully connected layer and complete the network modification.

****  
  
  
  
**Classify an Image and Display the Result**

% List of image filenames

imageFiles = ["amber.jpg", "snoopy.jpg", "sherlock.jpg"];

% Loop through each image

for i = 1:length(imageFiles)

% Read and resize image

img = imread(imageFiles(i));

img = imresize(img, inputSize(1:2));

% Display image

figure;

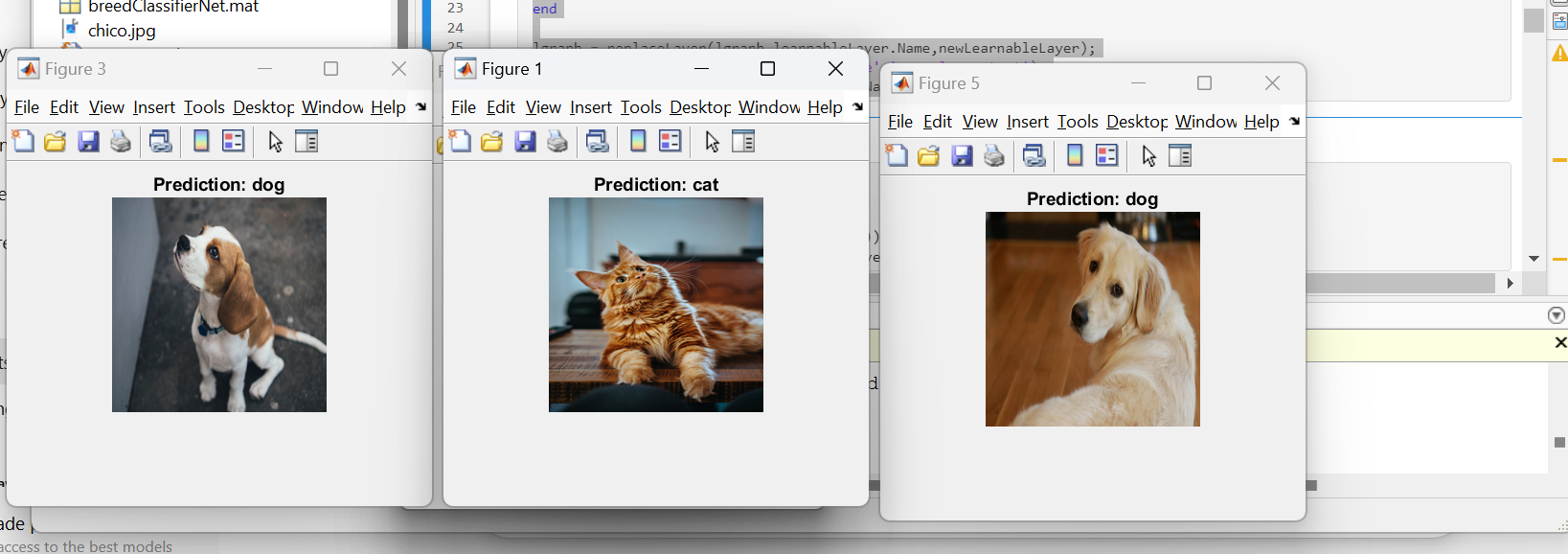
imshow(img);

title("Original Image: " + imageFiles(i));

% Predict class

[YPred, scores] = classify(net1, img);

% Display predicted label

title("Prediction: " + string(YPred));  
  
  
  
  
  
**Apply Grad-CAM for Visualization**% Generate and overlay Grad-CAM map

gradcamMap = gradCAM(net1, img, YPred);

figure;

imshow(img);

hold on;

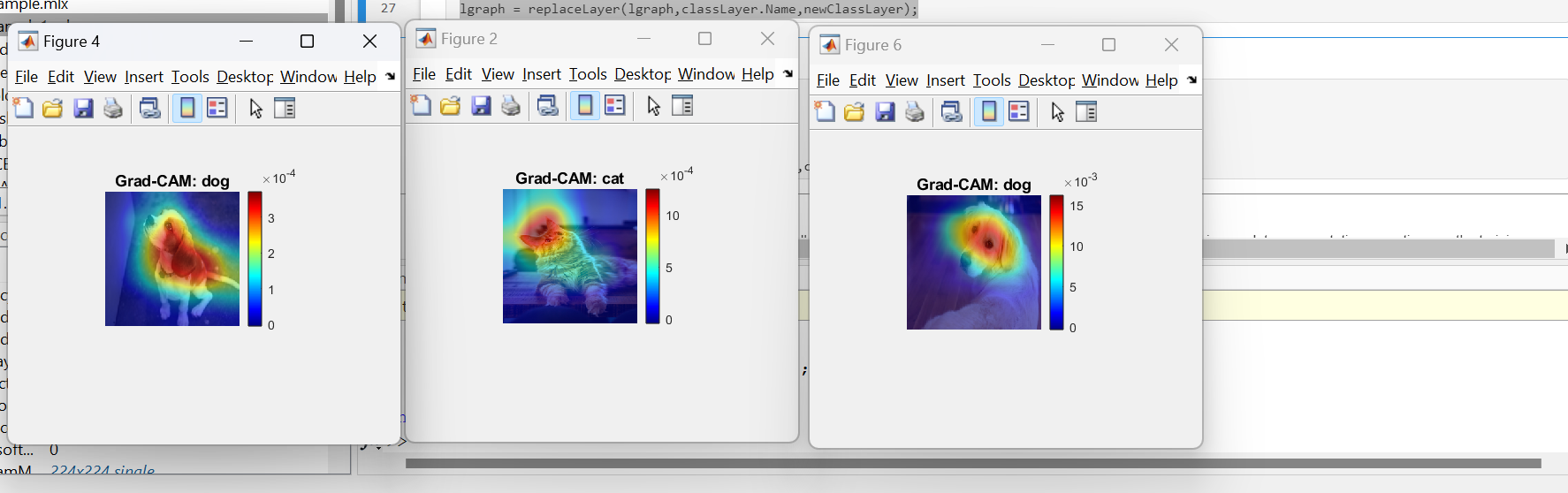
imagesc(gradcamMap, 'AlphaData', 0.5);

colormap jet;

colorbar;

title("Grad-CAM: " + string(YPred));

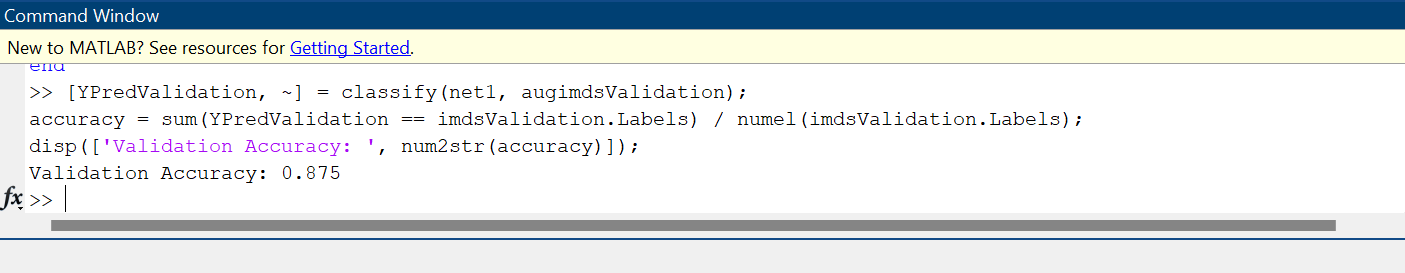
hold off;

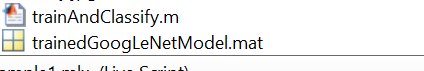
  
  
  
**Evaluate Model on Validation Set**

I am evaluating the trained model by using the test data to see how well it performs on unseen images. I use the classify function to get predictions and then I am plotting the confusion matrix to visually check the accuracy of predictions and where the model might be making mistakes.

[YPredValidation, ~] = classify(net1, augimdsValidation);

accuracy = sum(YPredValidation == imdsValidation.Labels) / numel(imdsValidation.Labels);

disp(['Validation Accuracy: ', num2str(accuracy)]);  
  
I am using the classify function to predict the labels for the test set (augimdsTest) using the trained network (trainedNet). I am then extracting the true labels from the test set (imdsTest.Labels). To calculate the accuracy, I am comparing the predicted labels (YPred) with the true labels (YTest) and calculating the ratio of correct predictions. Finally, I am using the confusionchart function to visualize the performance of the model by displaying a confusion matrix, which shows the true versus predicted labels for the test set.  
  
  
  
  
**Save the trained model for future use**save('trainedGoogLeNetModel.mat', 'net1');

disp('Model has been saved.');  
 ****