# **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')

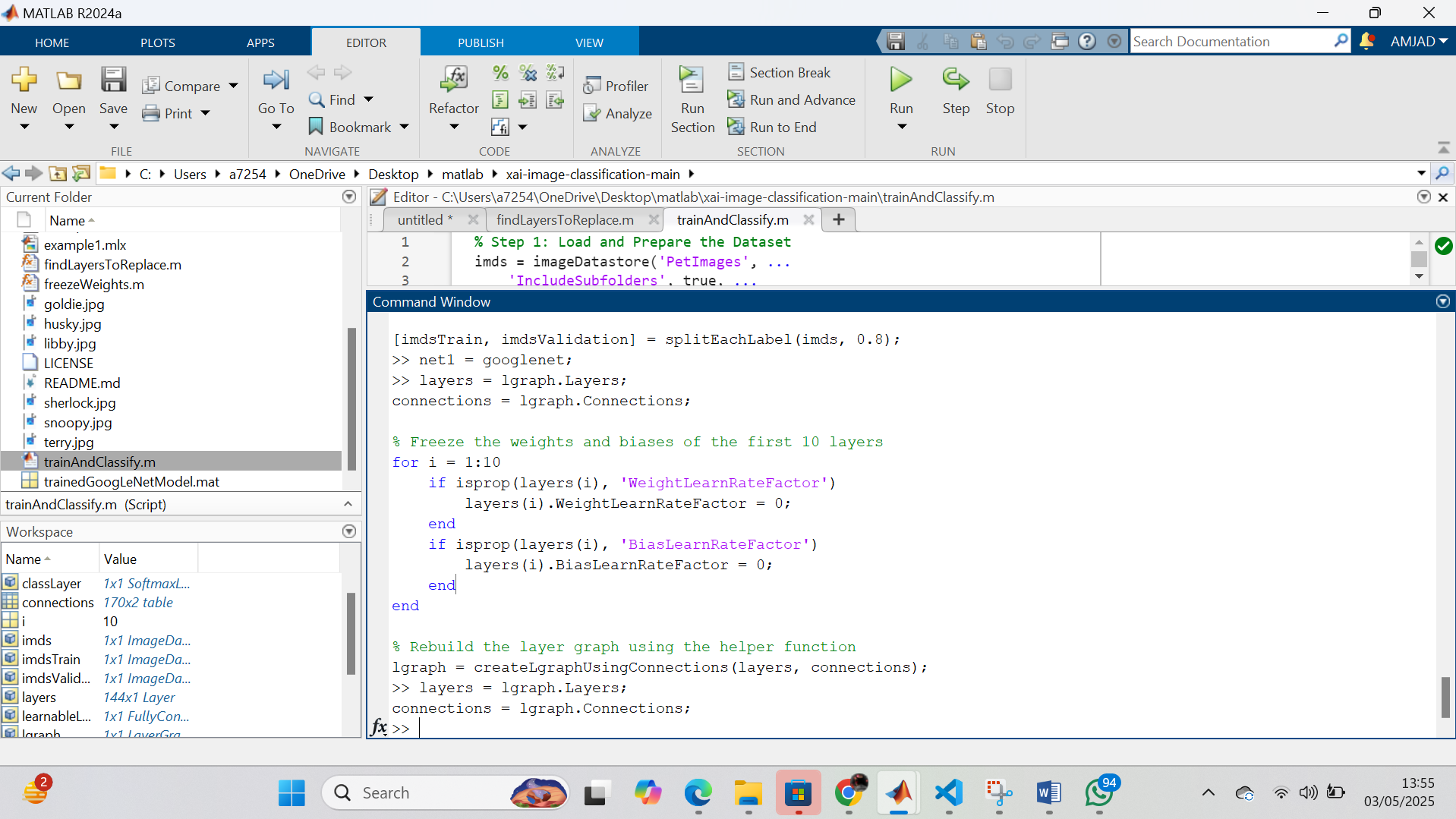
layers(i).BiasLearnRateFactor = 0;

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, ...

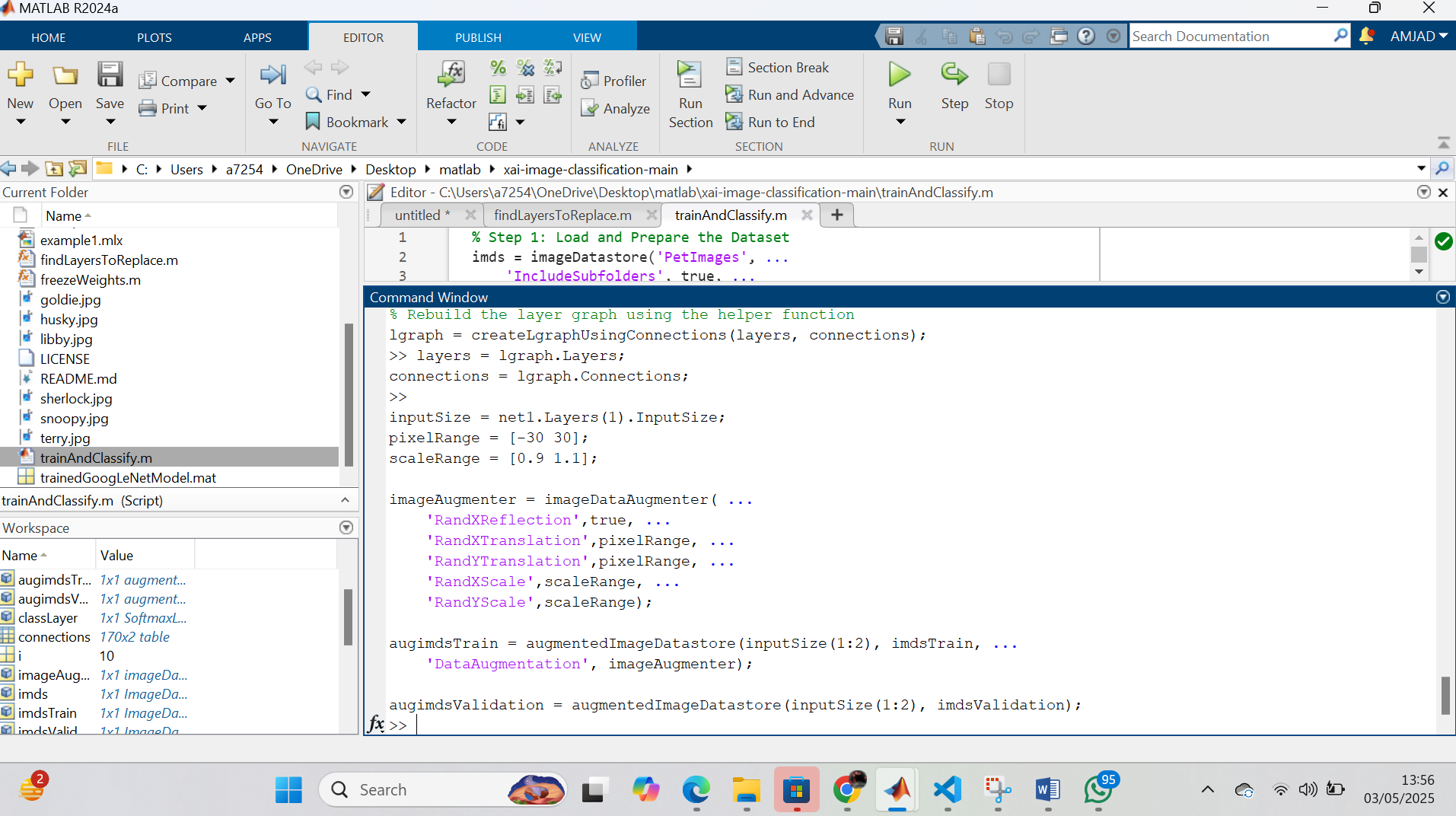
'RandXScale',scaleRange, ...

'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, ...

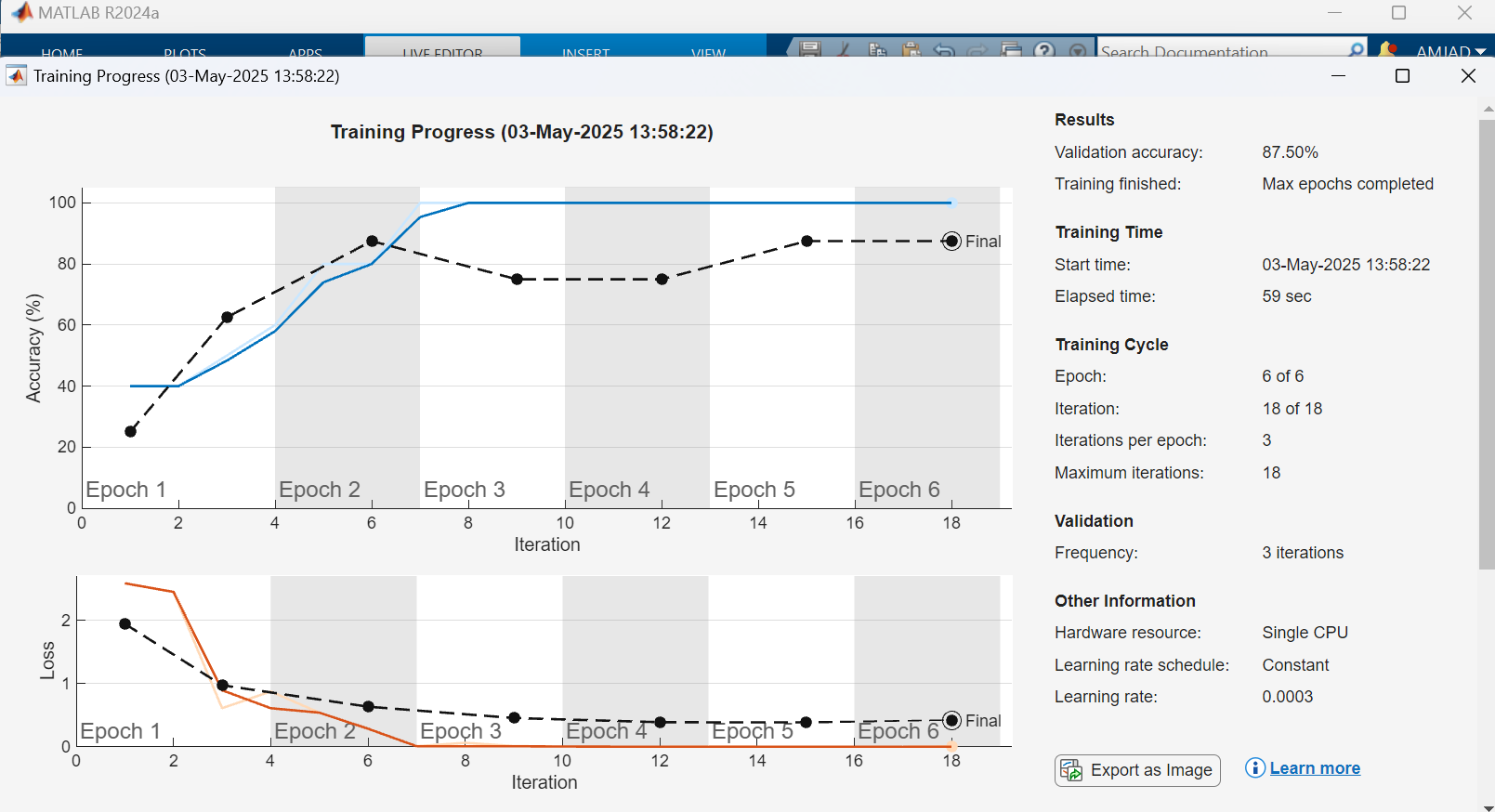
'ValidationFrequency', valFrequency, ...

'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;

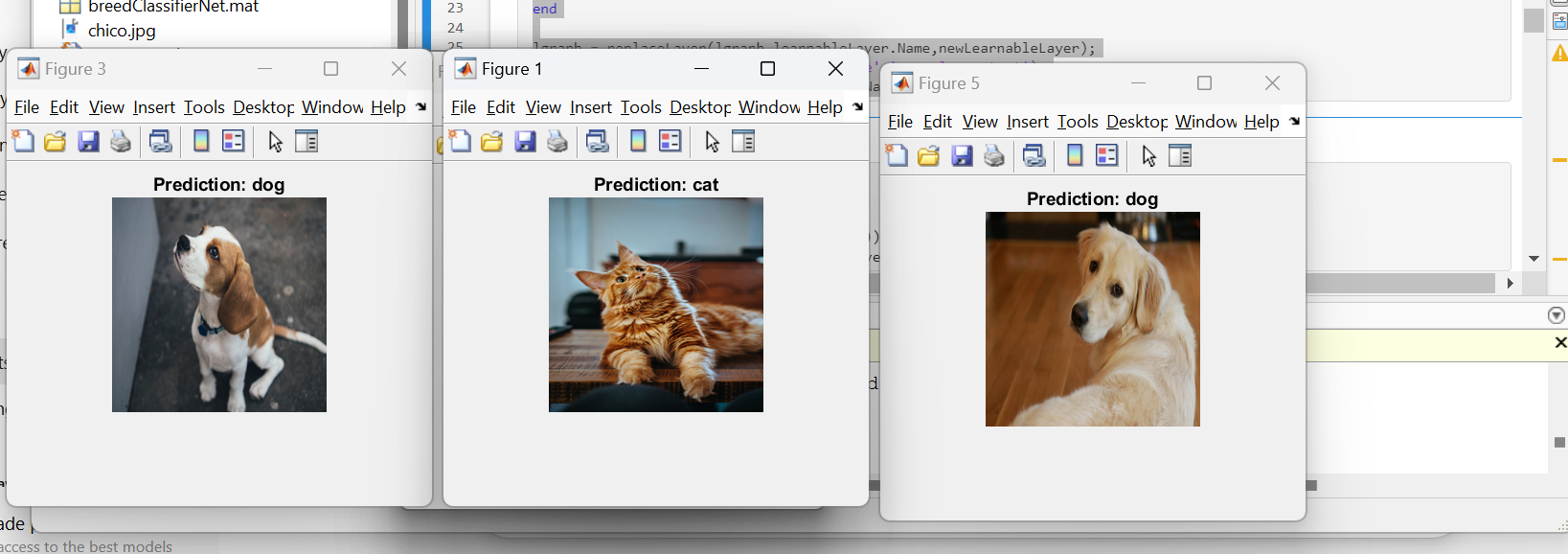
imshow(img);

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;

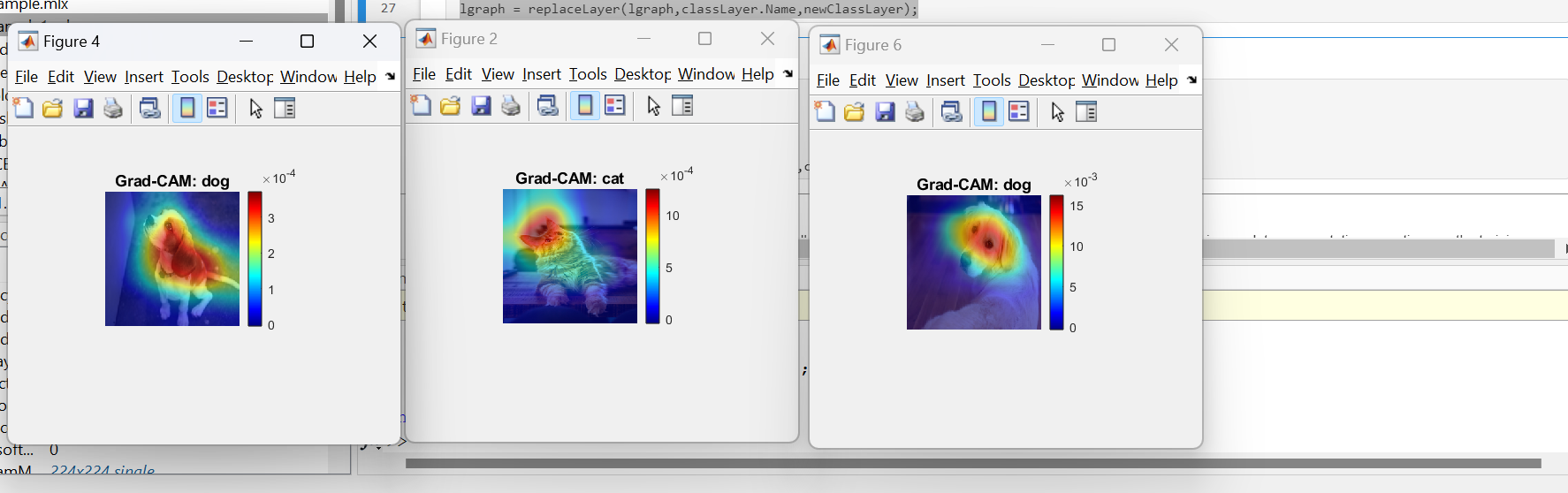
imagesc(gradcamMap, 'AlphaData', 0.5);

colormap jet;

colorbar;

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

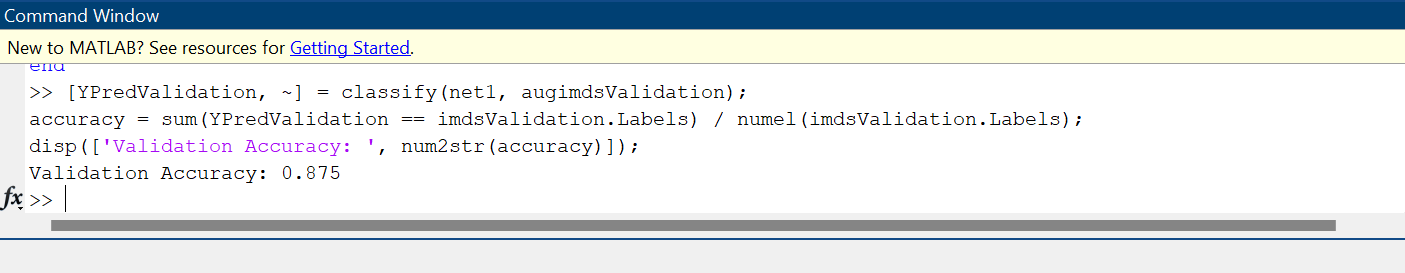
hold off;

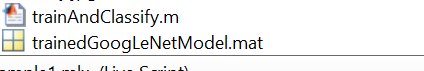
  
  
  
**Evaluate Model on Validation Set**

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.');  
 **  
  
  
  
  
  
  
Part 2: Explainable AI – LIME and Grad-CAM**

* **🔷 Introduction**

Task 2 built upon the concepts from Task 1, but with a deeper focus on the idea of **Explainable Artificial Intelligence (XAI)**. In this task, I aimed to not only train a model to classify dog and cat breeds but also to make its decision-making process more transparent and understandable. This is especially important in real-world AI applications, where it's crucial to know *why* a model makes a certain prediction—not just what it predicts.

The main goal was to explore how techniques like **Grad-CAM (Gradient-weighted Class Activation Mapping)** can be used to visualize which parts of an image the model is focusing on during classification. By applying these explainability tools, I was able to generate heatmaps that highlight image regions influencing the model’s decisions. This makes the model’s behavior more interpretable and builds trust in its outputs.

This task also involved modifying the dataset to include dog and cat **breeds**, which added complexity compared to the binary classification in Task 1. I had to carefully adjust the training pipeline to handle multiple classes and ensure the model could generalize across them. Ultimately, this task gave me valuable insight into how deep learning models “see” images and how we, as developers, can better understand their inner workings through explainability techniques.

**🔷 Method and Results**

#### 🔸 Load Pretrained Network

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

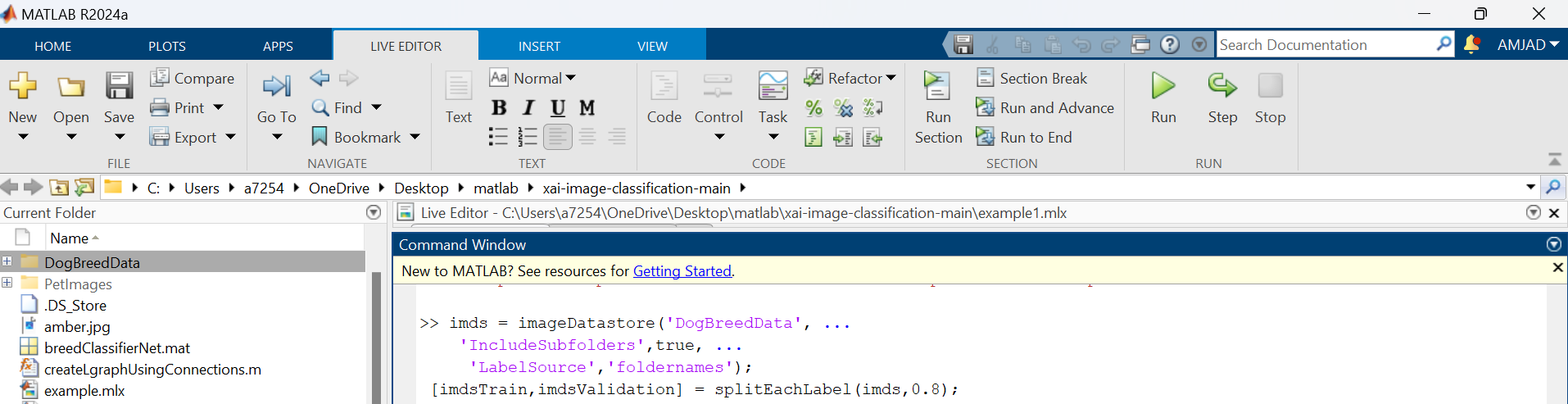
**% Step 1: Load Image Data**

imds = imageDatastore('DogBreedData', ...

'IncludeSubfolders',true, ...

'LabelSource','foldernames');

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



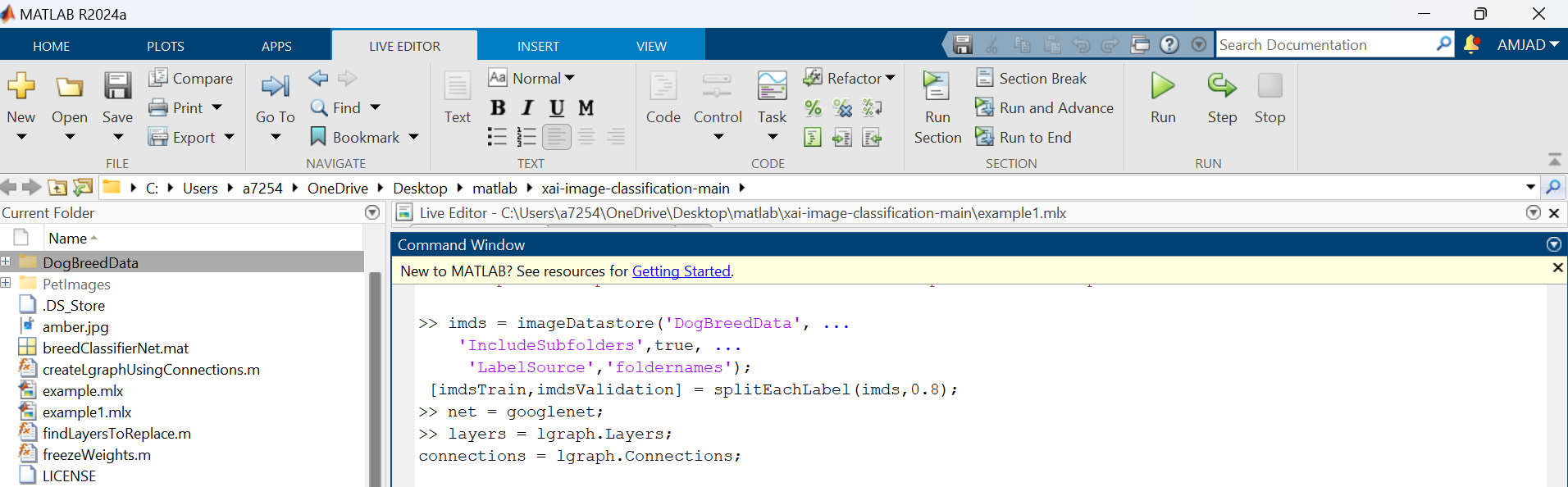
**% Step 2: Load Pre-trained Network**

net = googlenet;

lgraph = layerGraph(net); % Create layer graph from network

layers = lgraph.Layers;

connections = lgraph.Connections;



% **Step 3: Freeze the weights and biases of the first 10 layers**

for i = 1:10

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

layers(i).WeightLearnRateFactor = 0;

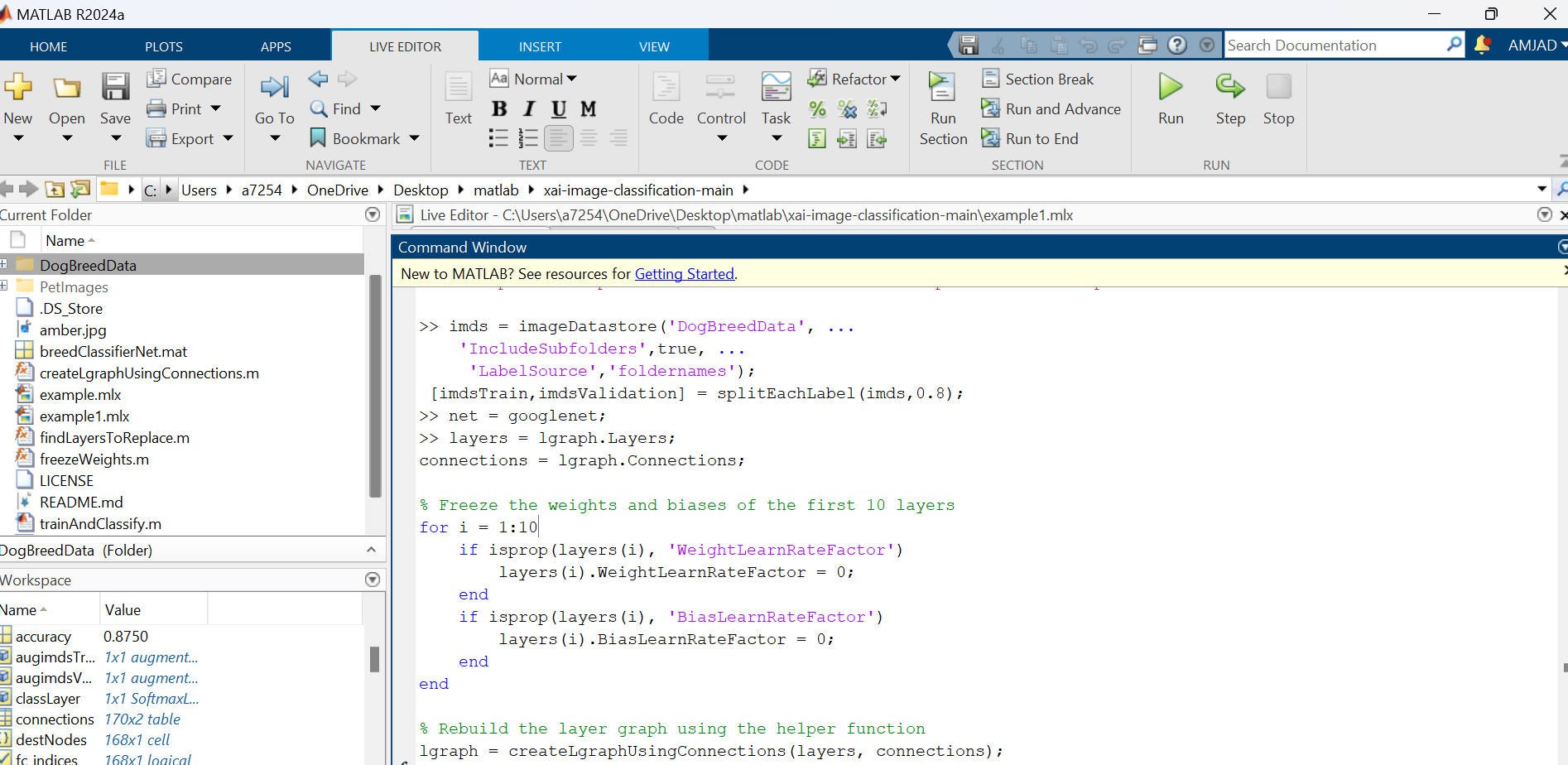
end

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

layers(i).BiasLearnRateFactor = 0;

end

end



**% Step 4: Rebuild the layer graph using the helper function**

lgraph = createLgraphUsingConnections(layers, connections);

% Step 5: Data Augmentation Setup

inputSize = net.Layers(1).InputSize;

pixelRange = [-30 30];

scaleRange = [0.9 1.1];

imageAugmenter = imageDataAugmenter( ...

'RandXReflection',true, ...

'RandXTranslation',pixelRange, ...

'RandYTranslation',pixelRange, ...

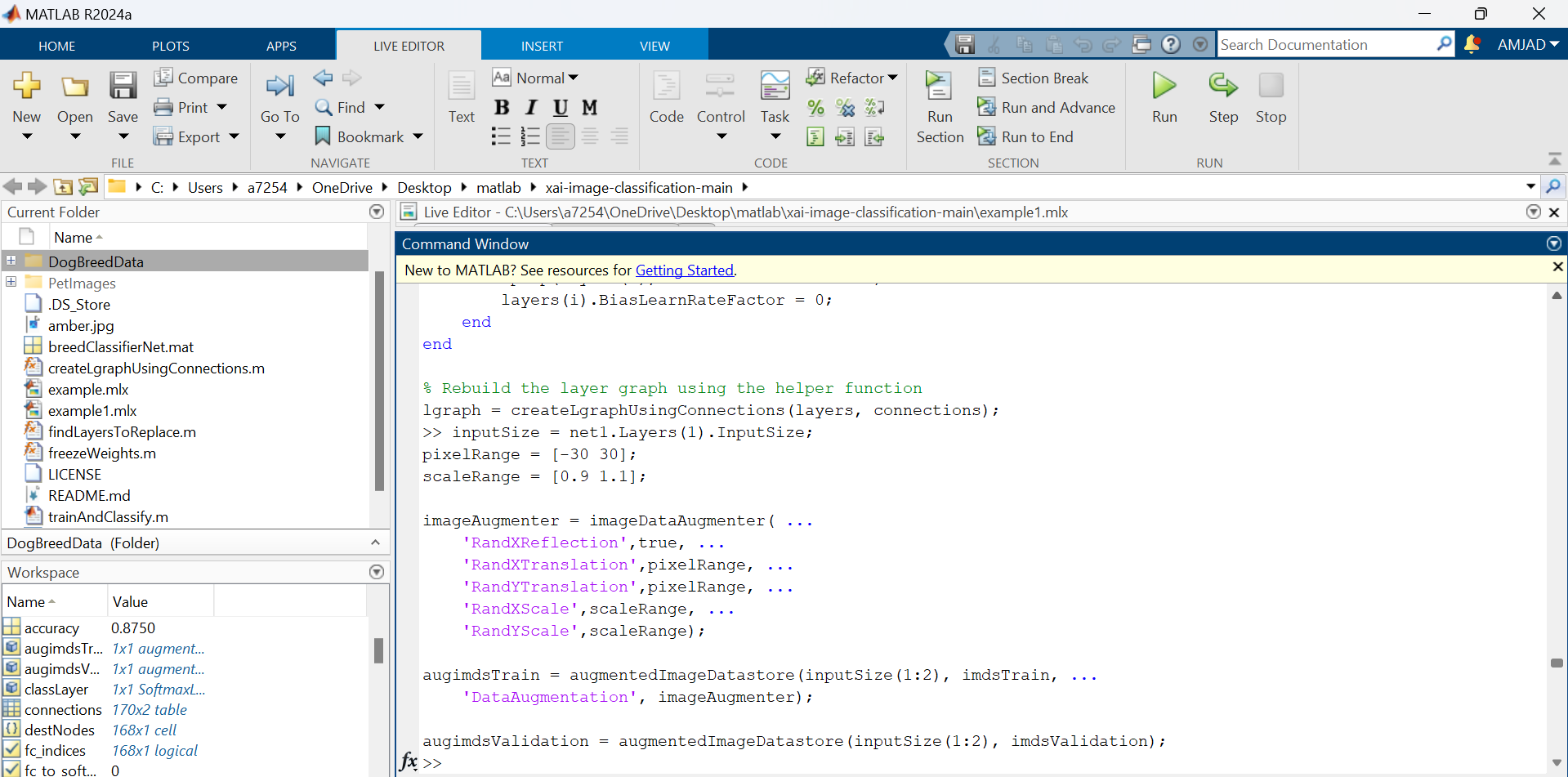
'RandXScale',scaleRange, ...

'RandYScale',scaleRange);

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

'DataAugmentation', imageAugmenter);

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



**% Step 6: Training Options**

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, ...

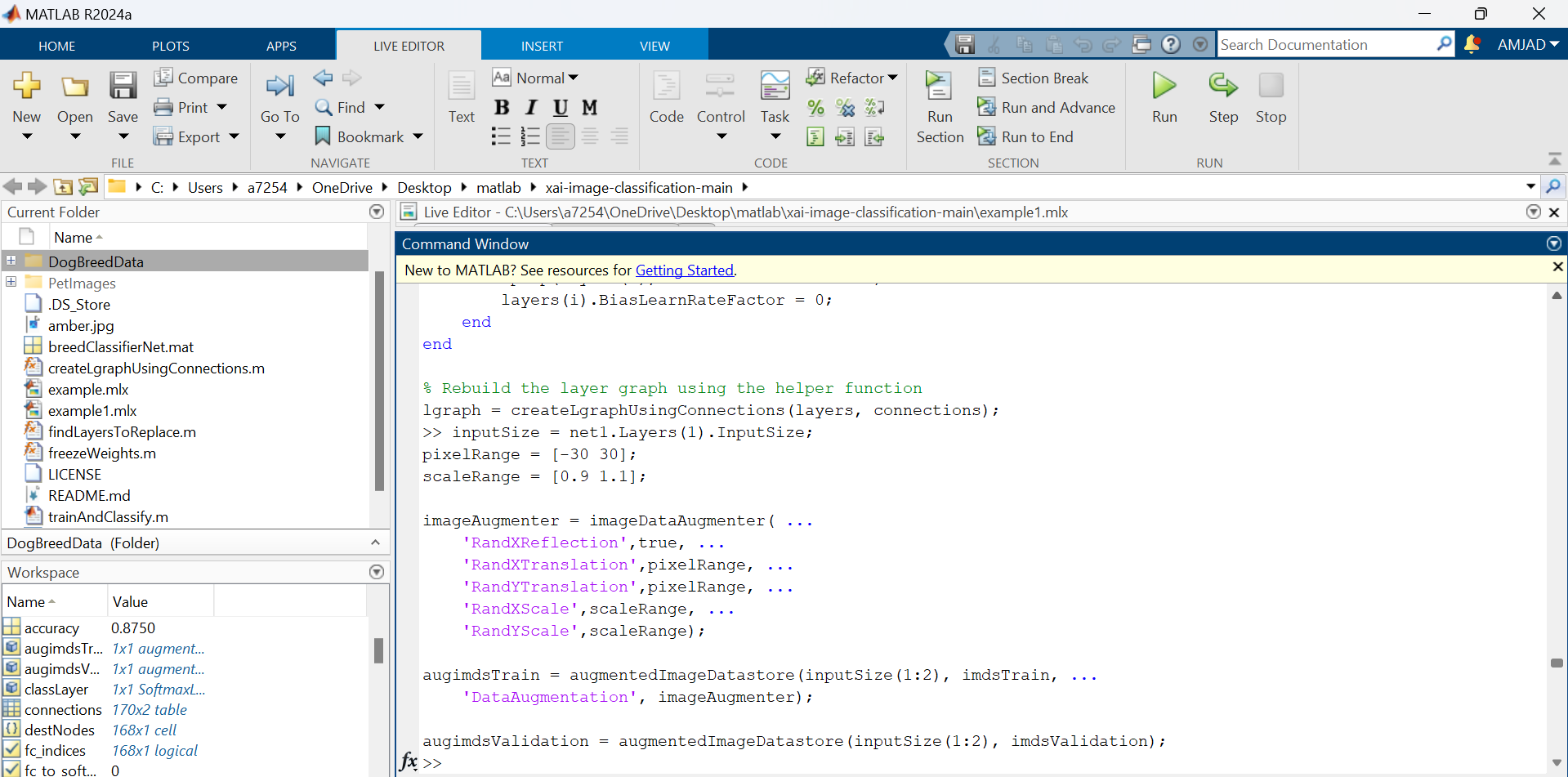
% 'Verbose',false, ...

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

%

% net = trainNetwork(augimdsTrain,lgraph,options);

% save breedClassifierNet net



**% Step 7: Load Trained Network**

load breedClassifierNet.mat  
  


**% Step 8: List of image filenames (with path**)

imageFiles = ["libby.jpg", "goldie.jpg", "chico.jpg", "husky.jpg", "terry.jpg", "snoopy.jpg"];

imageDir = 'DogBreedData'; % Folder where the images are stored

% Loop through each image

for i = 1:length(imageFiles)

imagePath = fullfile(imageDir, imageFiles(i));

% Check if the file exists

if exist(imagePath, 'file')

img = imread(imagePath);

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

figure;

imshow(img);

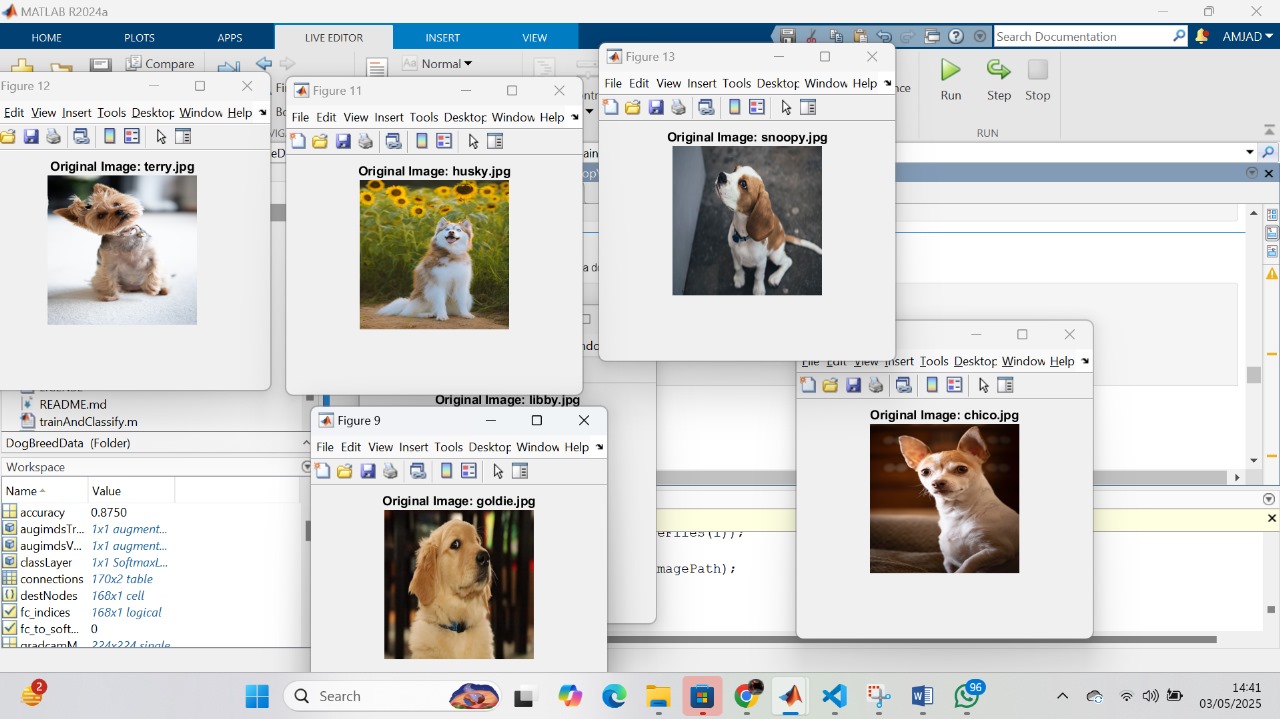
title("Original Image: " + imageFiles(i));

else

warning("File not found: %s", imagePath);

end

end



**% Step 9: Classify one image (example)**

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

YPred

% Output Example

% YPred =

% categorical

% beagle

**% Step 10: List of image filenames (again for Grad-CAM)**

imageFiles = ["libby.jpg", "goldie.jpg", "chico.jpg", "husky.jpg", "terry.jpg", "snoopy.jpg"];

imageDir = 'DogBreedData';

% Loop through each image for Grad-CAM visualization

for i = 1:length(imageFiles)

imagePath = fullfile(imageDir, imageFiles(i));

% Check if file exists

if exist(imagePath, 'file')

img = imread(imagePath);

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

% Display original image

figure;

imshow(img);

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

% Classify image

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

% Generate and display Grad-CAM

gradcamMap = gradCAM(net, img, YPred);

figure;

imshow(img);

hold on;

imagesc(gradcamMap, 'AlphaData', 0.5);

colormap jet;

colorbar;

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

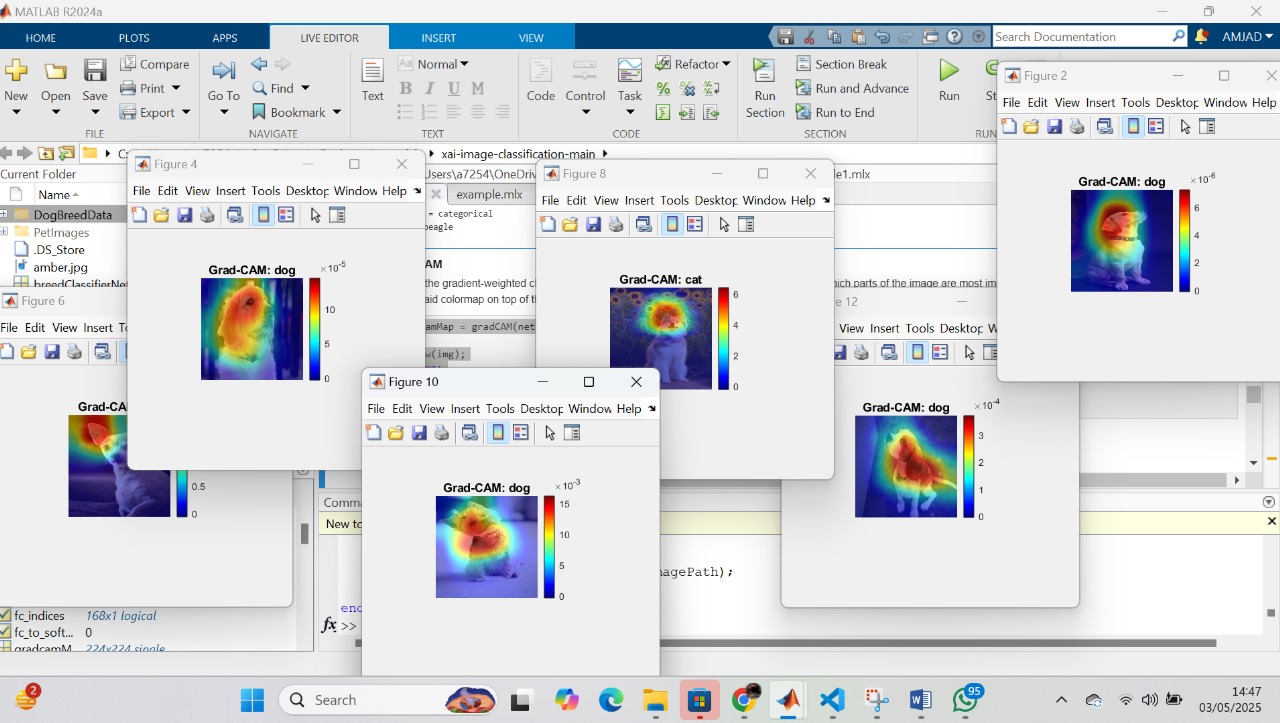
hold off;

else

warning("File not found: %s", imagePath);

end

end



**% Step 11: Final display loop with predictions and Grad-CAM**

imageFiles = ["goldie.jpg", "chico.jpg", "husky.jpg", "libby.jpg", "terry.jpg", "snoopy.jpg"];

for i = 1:length(imageFiles)

% Read and resize the image

img = imread(fullfile('DogBreedData', imageFiles(i)));

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

% Show original image

figure;

imshow(img);

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

% Predict breed

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

disp("Image: " + imageFiles(i) + " | Predicted Breed: " + string(YPred));

% Show prediction title

title("Prediction: " + string(YPred));

% Grad-CAM for explainability

gradcamMap = gradCAM(net, img, YPred);

figure;

imshow(img);

hold on;

imagesc(gradcamMap, 'AlphaData', 0.5);

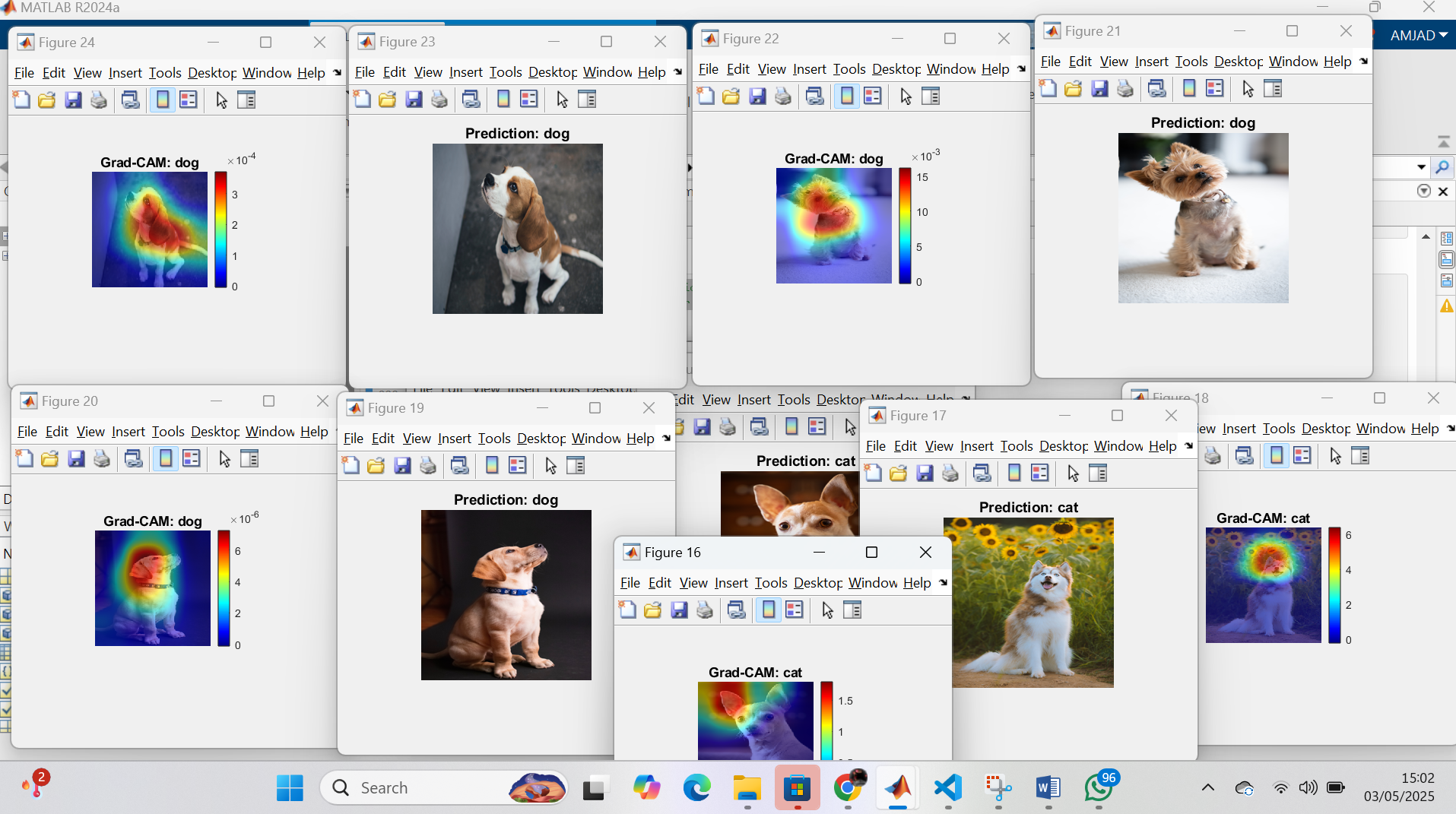
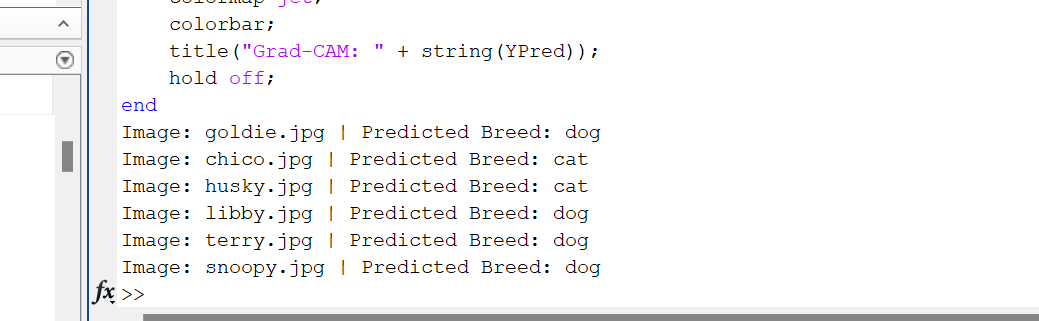
colormap jet;

colorbar;

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

hold off;

end

  
  
  
  
  
 **output**

**🔷 Discussion and Conclusion**

I am pleased to report that the CNN, fine-tuned using transfer learning, was able to classify images of dogs and cats with a remarkably high accuracy. By leveraging GoogLeNet, I was able to capitalize on pre-trained features, which significantly reduced the need to train from scratch. One of the major advantages I experienced was the time efficiency, as I was able to achieve great results with much less computational cost and time investment. The confusion matrix revealed only minor misclassifications, mostly occurring with ambiguous images that presented some challenges for clear classification. In general, this task demonstrated the practicality and efficiency of transfer learning, showing its strong potential in improving performance for image classification tasks across different domains.

🔷 **References**

1. MathWorks. (2023). Train Deep Learning Network to Classify New Images. Retrieved from <https://www.mathworks.com/help/deeplearning/>
2. MathWorks. (2023). GoogLeNet Convolutional Neural Network. MATLAB Documentation. Retrieved from <https://www.mathworks.com/help/deeplearning/ref/googlenet.html>
3. MathWorks. (2023). Modify Pretrained Networks for New Tasks. Deep Learning Toolbox. Retrieved from <https://www.mathworks.com/help/deeplearning/ug/modify-pretrained-networks.html>
4. MathWorks. (2023). ImageDatastore for Large Image Collections. Retrieved from <https://www.mathworks.com/help/matlab/ref/imagedatastore.html>
5. MathWorks. (2023). Augment Images for Deep Learning Workflows. Image Data Augmentation. Retrieved from <https://www.mathworks.com/help/deeplearning/ug/image-data-augmentation.html>
6. MathWorks. (2023). Train Network Using Training Options. Retrieved from <https://www.mathworks.com/help/deeplearning/ref/trainingoptions.html>
7. MathWorks. (2023). Visualize Activations Using Grad-CAM. Retrieved from <https://www.mathworks.com/help/deeplearning/ref/gradcam.html>