To generate a **ground truth mask** for the image you uploaded using the **Image Labeler app in MATLAB**, follow these steps:

✅ Steps to Label Image and Export Ground Truth Mask

1. **Open MATLAB**.
2. **Launch Image Labeler:**

```
imageLabeler
```

4. **Load the Image:**
   - In the Image Labeler UI, click "**Add Images**" and select your image file (e.g., `7b6d61fd-de7e-4e5f-8c89-e6d950657ecf.jpg`).
5. **Create Label Definitions:**
   - Click "**Label**" > "**Add ROI Label**".
   - Name your label (e.g., `building`, `vehicle`, `door`, etc.).
   - Choose an ROI type: usually **polygon** for precise masks.
6. **Draw ROI (Region of Interest):**
   - Select the label you created.
   - Use the mouse to manually draw around each object in the image.
   - You can add multiple regions with the same label or define additional labels.
7. **Export the Ground Truth Mask:**
   - After labeling is complete, go to "**Export Labels**" > "**To Workspace**".
   - It exports a variable of type `groundTruth`.
8. **Generate Pixel Masks (optional but common):**

```
% Convert the labeled ground truth to a pixel-wise mask
data = groundTruthData; % your variable name from export
pixelLabelData = pixelLabelTrainingData(data);

% Save mask images to disk
outputFolder = fullfile(pwd, 'masks');
if ~exist(outputFolder, 'dir')
    mkdir(outputFolder);
end
for i = 1:numel(pixelLabelData)
    imwrite(uint8(pixelLabelData{i,2}), fullfile(outputFolder, ['mask_' num2str(i) '.png']));
end
```

Here's a complete **MATLAB script** to automate the process of:

1. Opening the image in the **Image Labeler**.
2. Letting you label objects.
3. Exporting the label to the workspace.
4. Converting it to a **binary mask** or **multi-class mask**.
5. Saving the mask as an image file.

✅ Step-by-Step Script: `label_and_export_mask.m`

```
% Step 1: Launch the Image Labeler
imageLabeler('7b6d61fd-de7e-4e5f-8c89-e6d950657ecf.jpg');

% --- STOP HERE: Label the image in the UI manually ---
% Once you are done labeling, export the groundTruth object to workspace
(name it e.g. `gTruth`)
% Then run the following code to extract and save the mask
```

✅ Step 2: Extract and Save Ground Truth Mask

```
% Step 2: After labeling, extract pixel label data
pxds = pixelLabelTrainingData(gTruth);  % gTruth must be in workspace

% Optional: display label names and IDs
classes = gTruth.LabelDefinitions.Name;
disp("Classes labeled:");
disp(classes);

% Step 3: Save mask as image
outputFolder = fullfile(pwd, 'exported_masks');
if ~exist(outputFolder, 'dir')
    mkdir(outputFolder);
end

for i = 1:height(pxds)
    I = pxds{i, 2};  % pixel label matrix
    maskFilename = fullfile(outputFolder, ['mask_' num2str(i) '.png']);

    % Optional: convert to binary mask (for single class)
    if numel(classes) == 1
        binaryMask = uint8(I == 1);
        imwrite(binaryMask, maskFilename);
    else
        % Multi-class mask
        imwrite(uint8(I), maskFilename);
    end
end

disp("✅ Mask(s) saved to folder: " + outputFolder);
```

🖼 Output:

- Folder `exported_masks/` will contain:
  - `mask_1.png`, `mask_2.png`, etc., depending on number of images labeled.
- Each pixel in the mask image:
  - `0` = background
  - `1, 2, ...` = different classes based on your label definitions

Great! Here's an enhanced **MATLAB script** that supports **multiple images**, **custom label names**, and optionally exports the masks as `.mat` files as well as `.png` images.

✅ Full Script: Batch Image Labeling and Mask Export

⚠ **Before running this script**, make sure you:

- Have labeled your images using the **Image Labeler**
- Exported the result to the workspace as a `groundTruth` object (e.g., named `gTruth`)

📜 `generate_masks_from_groundTruth.m`

```
% ---------- CONFIGURATION ----------
outputFolder = fullfile(pwd, 'exported_masks');
exportMatFiles = true;  % set to false if you only want PNGs
exportPngFiles = true;  % set to false if you only want .mat

% ---------- CREATE OUTPUT FOLDER ----------
if ~exist(outputFolder, 'dir')
    mkdir(outputFolder);
end

% ---------- CONVERT GROUND TRUTH TO PIXEL LABELS ----------
pxds = pixelLabelTrainingData(gTruth);  % gTruth must be in workspace
labelDefs = gTruth.LabelDefinitions;

% Show class names
disp("Labeled Classes:");
disp(labelDefs.Name);

% ---------- EXPORT MASKS ----------
for i = 1:height(pxds)
    % Get image name and corresponding label matrix
    imageFilename = pxds{i, 1};
    [~, name, ~] = fileparts(imageFilename);
    labelMatrix = pxds{i, 2};  % uint8 matrix of class indices

    % Export PNG
    if exportPngFiles
        imwrite(uint8(labelMatrix), fullfile(outputFolder, [name
'_mask.png']));
    end

    % Export .mat
    if exportMatFiles
        mask = uint8(labelMatrix);
        save(fullfile(outputFolder, [name '_mask.mat']), 'mask');
    end
end

disp("✅ All masks exported to: " + outputFolder);
```

□ Output Details

- For **each labeled image**, the script saves:
  - `imagename_mask.png` → 8-bit indexed PNG mask.
  - `imagename_mask.mat` → Contains `mask` variable with the same data.
- **Pixel values** correspond to label indices (1, 2, 3, ...) as defined in `gTruth.LabelDefinitions`.

□ Example Use Case

- Label 3 images in the Image Labeler with labels like `car`, `door`, `building`.
- Export as `gTruth` to workspace.
- Run this script.
- You get:
- `exported_masks/`
- `├── image1_mask.png`
- `├── image1_mask.mat`
- `├── image2_mask.png`
- `├── image2_mask.mat`
- `...`

To generate **ground truth segmentation masks** from labeled images using **MATLAB's Image Labeler** and export them to a folder like `pixelLabelData/`, follow the step-by-step instructions and command-line code below.

✅ STEP-BY-STEP GUIDE (Command Line + UI)

Step 1: Launch the Image Labeler

Use MATLAB's Image Labeler to annotate your images.

```
imageLabeler
```

Inside the GUI:

1. **Add Images** you want to label.
2. Create label definitions (e.g., `Building`, `Road`, etc.).
3. Use polygon or brush tools to label objects.
4. After labeling, go to:
   Export > Export Labels to Workspace → name it as `gTruth`.

✅ Now your labeled data is in a variable called `gTruth`.

Step 2: Create a Folder to Save Masks

```
outputFolder = fullfile(pwd, 'pixelLabelData');
if ~exist(outputFolder, 'dir')
    mkdir(outputFolder);
end
```

Step 3: Convert Ground Truth to Pixel Label Data

This converts labeled polygons into pixel-wise label matrices:

```
pxds = pixelLabelTrainingData(gTruth);  % returns a table with image & mask
```

- `pxds` is a table:
  - Column 1: Image file path
  - Column 2: Pixel label matrix (2D matrix of class indices)

Step 4: Save Pixel Label Masks as Images

```
for i = 1:height(pxds)
    imgPath = pxds{i, 1};
```

```
    [~, name, ~] = fileparts(imgPath);  % get filename without extension
    mask = uint8(pxds{i, 2});  % convert to uint8 mask

    % Save the mask as PNG in the output folder
    imwrite(mask, fullfile(outputFolder, [name '_mask.png']));
end
```

✅ Output Structure (Example)

```
your_project/
│
├── your_images/
│   ├── img1.jpg
│   ├── img2.jpg
│
├── pixelLabelData/
│   ├── img1_mask.png
│   ├── img2_mask.png
```

📝 Notes:

- Each pixel in the `_mask.png` corresponds to a class index from your label definitions.
- You can visualize these masks using `imshow(mask, [])` or overlay them on images.
- These masks are directly usable for training U-Net or other segmentation models.

Great! Since you're working **only with grayscale images** and want to:

- ✅ Create **ground truth masks**
- ✅ Save **index masks** (grayscale, each pixel = class ID)
- ❌ **No RGB masks**
- ✅ Use for **training U-Net**

Here's the **clean, minimal version** specifically tailored for grayscale inputs.

✅ Step-by-Step: Export Grayscale Index Masks from Image Labeler

---

◆ Step 1: Label Your Images Using Image Labeler

```
imageLabeler
```

- Add your grayscale images
- Define class labels (e.g., `wall`, `road`, `sky`, etc.)

- Label your data
- Go to **Export > Export Labels to Workspace**, name it `gTruth`

◆ Step 2: Convert `gTruth` to Pixel Label Data

```
pxds = pixelLabelTrainingData(gTruth);  % returns table of image & mask
```

◆ Step 3: Save Ground Truth Masks as Grayscale PNGs

```
outputFolder = fullfile(pwd, 'pixelLabelData');
if ~exist(outputFolder, 'dir')
    mkdir(outputFolder);
end

for i = 1:height(pxds)
    imgPath = pxds{i, 1};
    [~, name, ~] = fileparts(imgPath);
    mask = uint8(pxds{i, 2});  % index-based label mask

    % Save grayscale mask: pixel value = class index (1, 2, ...)
    imwrite(mask, fullfile(outputFolder, [name '_mask.png']));
end
```

✅ Each `_mask.png` file now contains a **grayscale image** where:

- `0` = background or unlabeled
- `1, 2, 3, ...` = class indices as defined in `gTruth.LabelDefinitions`

✅ Step 4: Prepare `pixelLabelDatastore` for Training

```
classNames = gTruth.LabelDefinitions.Name;
labelIDs = 1:numel(classNames);  % assuming 1-based class indices

pxds = pixelLabelDatastore(outputFolder, classNames, labelIDs);
```

✅ You can now use this in U-Net training.

🔍 Example to Visualize a Mask

```
[label, info] = read(pxds);
imshow(label, []);
title("Grayscale Mask - Each Pixel = Class Index");
```

Exactly! Your workflow to generate grayscale index masks from the Image Labeler and then use them directly in U-Net training is spot on.

Since you asked, here's a minimal **MATLAB U-Net training snippet** using those grayscale masks and grayscale images:

Minimal U-Net Training for Grayscale Images + Grayscale Masks

```
% Assume you have:
% - Grayscale input images in 'imageFolder'
% - Grayscale index masks in 'pixelLabelData' (from your previous step)
% - classNames and labelIDs defined as before

imageFolder = fullfile(pwd, 'images');  % your grayscale images
maskFolder = fullfile(pwd, 'pixelLabelData');  % masks as grayscale index
PNGs

classNames = gTruth.LabelDefinitions.Name;
labelIDs = 1:numel(classNames);

imds = imageDatastore(imageFolder);

pxds = pixelLabelDatastore(maskFolder, classNames, labelIDs);

% Combine into training datastore
ds = pixelLabelImageDatastore(imds, pxds);

% Create U-Net for grayscale images (1 channel)
inputSize = [size(readimage(imds,1),1) size(readimage(imds,1),2) 1];
numClasses = numel(classNames);

lgraph = unetLayers(inputSize, numClasses);

% Set training options
options = trainingOptions('adam', ...
    'MaxEpochs', 25, ...
    'MiniBatchSize', 8, ...
    'Shuffle', 'every-epoch', ...
    'Plots','training-progress', ...
    'Verbose', false);

% Train the network
net = trainNetwork(ds, lgraph, options);
```

Key points:

- **Input images**: grayscale (single channel)
- **Masks**: grayscale indexed masks, each pixel = class ID
- `unetLayers` automatically adapts to your input size and channels
- Use `pixelLabelImageDatastore` to pair images & masks easily for training

Great! Here's a **full example** with everything included for training a U-Net on grayscale images and grayscale index masks — including validation split and simple augmentation.

Full MATLAB U-Net Training Script for Grayscale Images & Masks

```
% === Setup folders ===
imageFolder = fullfile(pwd, 'images');          % Folder with grayscale
input images
maskFolder = fullfile(pwd, 'pixelLabelData');  % Folder with grayscale
index masks

% === Load label info ===
% Assuming you already have gTruth from Image Labeler
classNames = gTruth.LabelDefinitions.Name;
labelIDs = 1:numel(classNames);

% === Create datastores ===
imds = imageDatastore(imageFolder);
pxds = pixelLabelDatastore(maskFolder, classNames, labelIDs);

% === Split data into training and validation sets (80/20 split) ===
numFiles = numel(imds.Files);
idx = randperm(numFiles);
numTrain = round(0.8 * numFiles);

trainIdx = idx(1:numTrain);
valIdx = idx(numTrain+1:end);

imdsTrain = subset(imds, trainIdx);
pxdsTrain = subset(pxds, trainIdx);

imdsVal = subset(imds, valIdx);
pxdsVal = subset(pxds, valIdx);

% === Create pixelLabelImageDatastore for training and validation ===
dsTrain = pixelLabelImageDatastore(imdsTrain, pxdsTrain, ...
    'DataAugmentation', imageDataAugmenter('RandXReflection', true));

dsVal = pixelLabelImageDatastore(imdsVal, pxdsVal);

% === Define U-Net ===
inputSize = [size(readimage(imds,1),1), size(readimage(imds,1),2), 1]; %
grayscale
numClasses = numel(classNames);

lgraph = unetLayers(inputSize, numClasses);

% === Training options ===
options = trainingOptions('adam', ...
    'MaxEpochs', 30, ...
    'MiniBatchSize', 8, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', dsVal, ...
    'ValidationFrequency', 50, ...
    'Plots', 'training-progress', ...
    'Verbose', false);

% === Train the network ===
```

```
net = trainNetwork(dsTrain, lgraph, options);

% === Save the trained network ===
save('trainedUNet_grayscale.mat', 'net');
```

Explanation:

- **Data split:** Random 80/20 split for training and validation
- **Augmentation:** Simple horizontal flips during training only
- **Input size:** Automatically set from your first grayscale image (single channel)
- **Training options:** Adam optimizer, batch size 8, 30 epochs, with validation monitoring
- **Saving:** Network saved for later inference or deployment

---