MathWorks®

https://explore.mathworks.com/all-about-pretrained-models

Deep Learning Questions Asked and Answered

By Johanna Pingel

All About Pretrained Models

In this column, we're tackling pretrained models. There are lots of models to choose from, so there's a lot of ground to cover.

Today I want to do something slightly different. The questions I'm about to answer are all based on things asked in the MathWorks community forum. I'll summarize the answers from the forum before diving into the question I always have when browsing MATLAB Answers: Why are they asking this?

With that in mind, this column will look at the basics of choosing a pretrained model, how to know if you've made the right choice, and three questions on pretrained models:

1. Should I manipulate the data size or the model input size when training a network?
2. Why import a pretrained YOLO model into MATLAB?
3. Why freeze the weights of a pretrained model?

Choosing a pretrained model

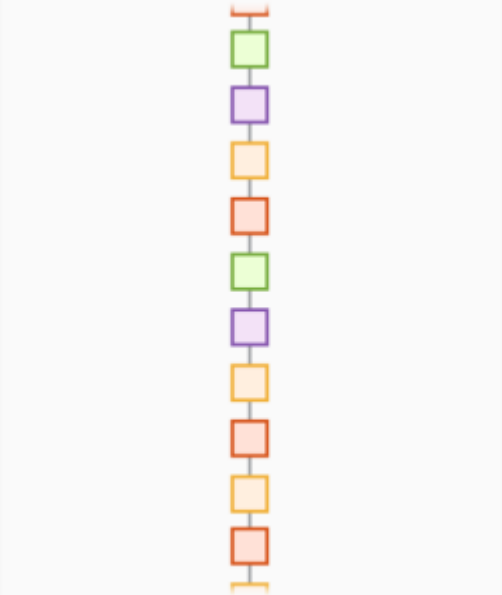
There are quite a few models to choose from, and the list will only continue to get longer as the years progress. This can be great for many reasons. It can also be somewhat daunting: How do you pick, and how do you know you made the right choice?

Rather than thinking of all pretrained models as a long list of options, I'd like to think of them in categories instead.

Basic models

These are models with simple architectures to get you up and running with confidence. They tend to have fewer layers, and allow for quick iterations on preprocessing and training options. Once you have a handle on how you want to train your model, you can move to the next section to see if you can improve your results.

Start here: GoogLeNet, VGG-16, VGG-19, and AlexNet



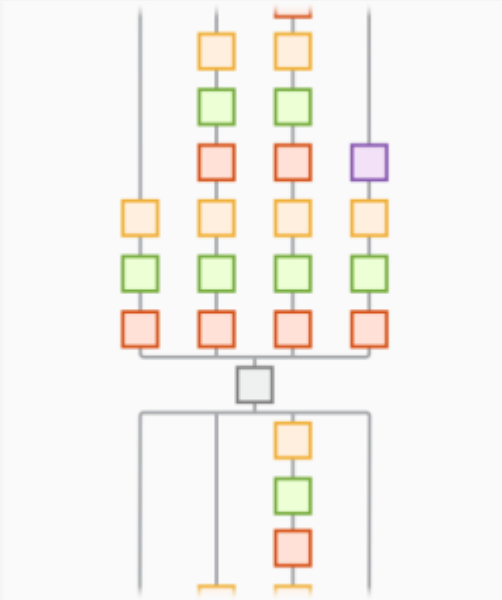
Higher accuracy models

These models cover your image-based workflows, such as image classification, object detection, and semantic segmentation. Most networks, including the basic models above, fall into this category. The difference between the Basic and the Higher Accuracy Models is these models may require more training time and have a more complicated network structure.

Start here: ResNet-50, Inception-v3, Densenet-201, Xception

Object detection workflows. DarkNet-19 and DarkNet-53 are often recommended as the foundation for detection and YOLO type workflows. I've also seen ResNet-50 used with Faster R-CNN, so there's some flexibility here too. We'll get into more details on object detection in the questions below.

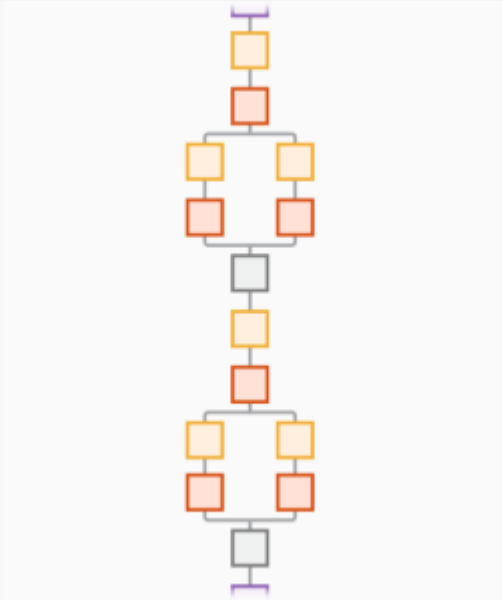
Semantic segmentation. You can take a network and turn it into a semantic segmentation network. There are also specific Segnet structures, such as `segnetLayers` and `unetLayers`.



Models for Edge Deployment

When moving to hardware, model size can become increasingly important. These models are intended to have a low-memory footprint, for embedded devices like Raspberry Pi.

Start here: SqueezeNet, MobileNet-v2, ShuffLeNet, NASNetMobile



These are just general guidelines to give you some direction when attempting to choose a model. I would start from the first category and continue to use more complex models as necessary. Personally, I see nothing wrong with using AlexNet as a starting point. It's a very simple architecture to understand and *may* perform adequately depending on the problem.

How do you know you've chosen the right model?

There may not be one right choice for your task. You'll know you have an acceptable model when it performs accurately for your given task. What level of accuracy you consider acceptable can vary considerably by application. Think about the difference between an incorrect suggestion from Google when shopping and a missed blizzard warning.

Trying out a variety of pretrained networks for your application is key to ensure you get to the most accurate and robust model. And of course, network architecture is only one of many dials you need to turn for a successful outcome.

Check out the latest pretrained models available on the MATLAB Github page.

Q1

Should I manipulate the data size or the model input size when training a network?

This question is drawn from the questions How do I use grayscale images in pretrained models? and How can I change the input size of my pretrained model?

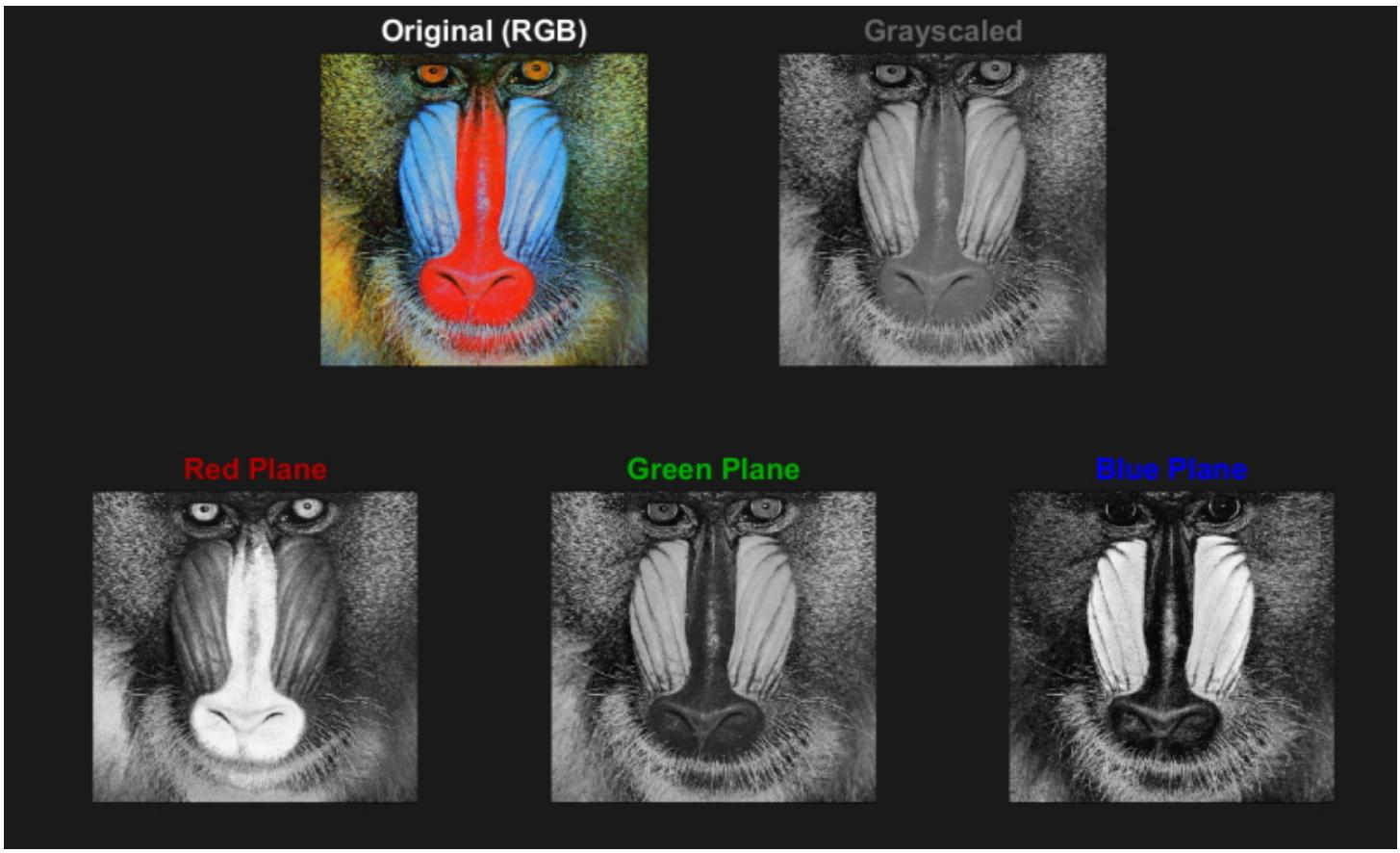
First a quick background on data input into the model.

All pretrained models have an expectation of what the input data structure needs to be in order to retrain the network or predict on new data. If you have data that doesn't match the model's expectation, you'll find yourself asking these questions.

Here's where things get interesting: Do you manipulate the data, or do you manipulate the model?

The simplest way is to change the data. This can be very simple: the input size of the data can be manipulated by resizing the data. In MATLAB®, this is a simple `imresize` command. The grayscale question also becomes very simple.

Color images are typically represented in RGB format, where three layers represent the red, green, and blue colors in each plane. Grayscale images contain only one layer instead of three. The single layer of a grayscale image can be replicated to create the input structure anticipated by the network, as shown in the image below.



Looking at a very colorful image, notice that the RGB planes look like three grayscale images that combine to form a color image.

The not-so-simple way is to change the model. Why would someone go through this trouble to manipulate the model instead of the data?

The input data you have available will determine this for you.

Let's say your images are 1000x1000px and your model accepts images of size 10x10px. If you resize your image to a 10x10px you will be left with an input image of noise. This is a scenario in which you want to change the input layer of your model, rather than your input.

Image size: 1000x1000px

Image size: 10x10px

I thought it would be a complicated task to manipulate the model input before I tried it in MATLAB, and it's not that bad. I promise. All you have to do is:

1. Open the Deep Network Designer app.

2. Choose a pretrained model.

3. Delete the current input layer and replace it with a new one. This enables you to make changes to the input size.

4. Export the model, and you are ready to use it for your transfer learning application. I would recommend practicing with a basic transfer learning example.

It really is very simple and you'll be able to change the input size of the pretrained models with no manual coding.

Why import a pretrained YOLO model into MATLAB?

This question is based on Yolo v3 training on coco data set, which has a straightforward answer. The background is simple.

This example on YOLO walks through training a YOLO v2 network with ResNet-50 to use in MATLAB.

YOLO stands for "you only look once." There are multiple versions of the algorithm available, with v3 adding improvements
in locating smaller objects over v2. YOLO starts with a feature extraction network (using a pretrained model such as ResNet-50 or DarkNet-19) followed by localization.

So why import a pretrained YOLO model into MATLAB? YOLO is one of the most popular algorithms available for object detection. Object detection poses significantly more challenges than simpler object recognition problems. With object detection, you need to not just identify the object, but also decide where it is located.

There are two categories of object detectors: YOLO is a one-stage detector, and Faster R-CNN is a two-stage detector.

- One-stage detectors can achieve high speeds in detection.

  This documentation page gets into the details of the YOLO v2 algorithm.

- Two-stage detectors have high localization and object recognition accuracy.

  This documentation page looks at the basics of R-CNN algorithms.

There are many applications of object detection to explore further, but I would highly recommend starting with a simple object detection example and building from here.

Q3

Why freeze the weights of a pretrained model?

This question is based on How can I freeze specific weights of [a] neural network model? Watch the video here. To answer this question, we'll start with a little bit of code.

Freezing the weights brings two benefits, you can:

- Speed up training. Because the gradients of the frozen layers do not need to be computed, freezing the weights of many initial layers can significantly speed up network training.
- Prevent overfitting. If the new dataset is small, freezing earlier network layers can prevent those layers from overfitting to the new dataset.

You can also essentially take the weights of a pretrained model and apply them to your model to create a "trained" network without training. Check out how `assembleNetwork` works in MATLAB to see how to create a deep learning network from layers without training.

Lastly, speaking of weights, you can use a weighted classification output layer for classification problems with an imbalanced distribution of classes. Here's an example using a custom weighted classification layer.