Suggestions of implementing algorithm on larger dataset like with [Dogs vs Cats Kernel Edition](https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition) on Kaggle:

I have been working to create something that will enable students who have completed this course to experiment with larger datasets. This is work in progress, but I am at a stable point so can share the following recommendations:

1. Since you are already on **Kaggle**, I would suggest you work with [Dogs vs Cats Kernel Edition](https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition) which has more recent data.
2. Kaggle is a great environment to work in. When you create a Kaggle **kernel**, all common packages are pre-installed for you and you can work with the familiar Jupyter notebooks just like Coursera. As of this writing (Aug 2018), in addition to CPU and RAM, you get access to a NVIDIA Tesla K80 GPU and connection to the Internet. All for free! Other than busy times during a competition starting or ending, there is practically nobody there so you will have the environment all to yourself.
3. Having said the above, I would encourage you to work on your local machine to see the power of what I am going to recommend to you. To start, download the [data](https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data) for the competition to your local machine.
4. In later courses (especially Course 3), you will learn about Convolutional Neural Networks (CNN), transfer learning, programming in Keras, etc. If you have some familiarity with these concepts, great, but we will just build on what you have learnt in Course 1. After you have done that, you can venture into the CNN stuff. I definitely encourage you to take Course 3 if you are interested in computer vision.
5. If you recall, in Course 1, you took cat **pictures**of size **64x64x3** and flattened them into **vectors** of size **12288**. In machine learning parlance, you took **mx64x64x3 examples** and converted them into **mx12288 features**. As you know, 64x64x3 is a tiny size for pictures, but the course had to use that size to get the models to run in a reasonable time for all students and in any case, it is better to start small. In any case, a more typical input image size is **224x224x3** or **299x299x3**, which means feature size of **150528** and **268203**. And we have **25000** training examples and **12500** test examples in this competition. The lights should dim when you try to run a fully connected network of that size on your computer :) In addition, you have to learn to do things in **batches**. When I was not using batches, I was crashing even the Kaggle environment! Finally, a network that large is very hard to train.
6. To make things work on local machines (though you can stay on Kaggle), I have processed the input images through common transfer learning models like **inception\_v3**, **resnet50**, etc. (*as Prof Ng says, "if you don't know what that means, don't worry about it :)*), and made the resulting [Cats vs Dogs Redux Transfer Features](https://www.kaggle.com/kanwalinder/cats-vs-dogs-redux-transfer-features) publicly available on Kaggle. You can download datasets for all transfer models that I used. Now what you have are features that are **mx2048** (for features created using **inception\_v3**) which are **6**times smaller than the features used in class. Think of this step as a gaint **flatten**operation that gave you very compact features to input into a fully connected classifier. Please review the [Overview](https://www.kaggle.com/kanwalinder/cats-vs-dogs-redux-transfer-features/home) tab of the dataset for more details.
7. If you click on the Kernels tab for the dataset above, you will see companion [README: Dogs vs Cats Redux Classifier](https://www.kaggle.com/kanwalinder/readme-dogs-vs-cats-redux-classifier) kernel. You can download the kernel to your local machine (but you will have to setup the packages locally) or fork the kernel on Kaggle. In any case, this kernel classifies the dataset (you can use datasets for all models to see variation in prediction accuracy) and creates a Kaggle submission file and a review file to look at the results. The whole process is fast! You can even "compete" (the competition is closed) in the competition to go all the way. The best result you will see is a **log loss** of about **0.21** which will fetch you a leaderboard position between 700 and 800 which is crazy given how little effort we have put into it. These simple classifiers are giving us an prediction accuracy over **99%**! **NOTE**: I wrote the classifier in Keras but used the familiar **LAYERS\_DIMS** parameter to specify the size (**(1024, 512, 256, 128)** which is tiny) of the network. I also used **DROPOUT** (of **0.5**) which you will learn about in Course 2 as a technique to prevent overfitting, In any case, the code should be a gentle introduction to Keras and hopefully encourage you to go further in the specialization.]
8. I am writing some code to review the results and to do some data processing tricks to improve the score on the competition which I will share later.
9. (**OPTIONAL**) If you would like to see how the data was processed through the transfer models, please review the kernel for [Dogs vs Cats Features from Transfer Models](https://www.kaggle.com/kanwalinder/dogs-vs-cats-features-from-transfer-models/notebook). I wrote the code for readability and did not use all the tricks like data generators which obscure what is going on. If you fork this kernel, grab some coffee :) And make sure GPU and Internet is turned on (off by default) for your Kaggle kernel. And just for kicks, compare running things (perhaps with smaller samples) using just the CPU vs CPU+GPU. It is an impressive change the first time you see it. And when using both, you get you see the CPU pegged as the images are loaded and then the GPU go at it.

If you like the work, please upvote it on Kaggle :) Please let me know if you have any questions or comments. Thanks!