EECS 16ML: Introduction to Pre-trained Models

Charlie Snell csnell22@berkeley.edu

Arnav Gudibande arnavg@berkeley.edu

Divi Schmidt divi@berkeley.edu

Abstract

Recent advancements in the performance and ability of deep neural networks have made pre-trained models a very popular and practical way of bringing state-of-the-art (SOTA) performance to a wide variety of tasks. In this note, we will introduce pre-trained models by directly building off of key ideas and concepts you already know from EECS 16AB and 16ML. Specifically, we will be using image classification as the overall problem context and the concept of featurization as the primary motivation for pre-trained models. We will then give a brief overview of common pre-trained models and the considerations needed to choose among them. Finally, we will highlight some real-world applications of pre-trained models, which you will then implement in the companion programming assignment.

1 Motivating Pre-trained Models

Let us begin with a very simple problem statement: How do we determine whether some image is a dog or a cat given some labeled images of other dogs and cats? While you may immediately jump to some of the recent ideas you have seen in the previous lecture about Convolutional Networks, it is best to keep our first approach simple for now, especially since we are in the realm of binary classification.

1.1 A First Approach

In order to convert this problem statement into a format that we know how to solve, we must first setup a system of equations. Let us denote our dataset of cat and dog images as $\mathcal{D}$ and each individual image as $a_i \in \mathcal{R}^{s \times s}$ such that $0 \leq i < n$ where s is the height/width of the image and n is the size of our dataset. We also have a set of labels, which we can denote as y_i where $y_i = +1$ if a_i is a cat, and $y_i = -1$ if the image is a dog.

Now you may be thinking, I understand a_i is an image, but how do we represent it in our system of equations?

This is where the idea of featurization, or a mapping from our image to some vector, will be able to help us. You've seen this idea in 16AB before, so now let us employ a naive featurization to setup our system of equations. Our naive featurization ϕ will simply map each pixel in a_i to a vector $x_i \in \mathcal{R}^{d \times 1}$ where $d = s^2$. Now, we can stack our x_i vectors and labels y_i to produce a design matrix $X \in \mathcal{R}^{n \times d}$ and $Y \in \mathcal{R}^n$.

This looks very familiar now, and we can use the closed-form OLS solution[1] for an over-determined system of equations[2] to find $w = (X^T X)^{-1} X^T Y$ where $w \in \mathcal{R}^d$.

What is our final prediction for some image v? Simply, "cat" if $\phi(v)^T w > 0$ else "dog".

[1]Refer to EECS 16B notes for a refresher proof of this if necessary

[2]Over-determined since we are assuming $n > d$. Can you think of what happens when $n < d$?

Preprint. Under review.

However, we have just applied a regression technique to a classification problem, this may appear slightly unnatural[3] to you. We really should be using a classification technique instead. Furthermore, what happens if the dimension of our image s is really big? If s gets very large, our naive featurization will become quadratically more complex to compute. This motivates our second approach.

1.2 A Second Approach

Instead of using OLS, let us use a classification technique you recently learned, Logistic Regression. As you may recall, there is no closed-form solution for this. Instead, we can use Stochastic Gradient Descent to find the w that approximately minimizes the logistic loss[4].

Now, we just need to find a better featurization technique. As you saw in this week's lecture, convolutional filters can be used to extract features from an image. Let's do a quick example: Say we are interested in including vertical edges as a feature for an image. How would we use a filter to find all occurrences of vertical edges in our image?

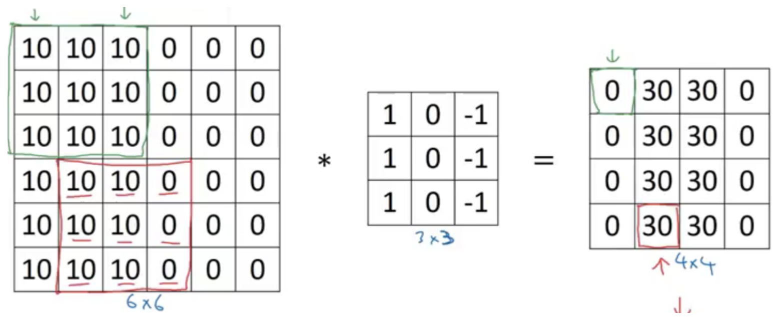

Figure 1: A vertical edge detector convolving an image. Source: Andrew Ng Coursera

After convolving the 6×6 image a_i with a 3×3 filter f_1, we were left with a 4×4 result where all non-zero values are instances of when the filter passed over a vertical edge in our original image.

This is great! Not only does our output tell us where these vertical edge features appear in the original image, but also the resulting image is smaller, which means our naive[5] featurization ϕ will be more efficient. Explicitly, we can represent the result of this single vertical-edge featurization as $\phi(a_i * f_1)$ where $*$ is the convolution operation.

Unfortunately, manually designing good feature maps is fairly challenging, as you will see in the programming assignment. Is there a way that we can use pre-trained models to do this?

1.3 A Third Approach

Pre-trained models are simply models that are trained on a large dataset, usually with a lot of computational resources, that solve a task similar to the one that you are trying to solve. We can view pre-trained models as the ultimate feature maps, because the features that they learn are incredibly comprehensive and expressive.

In order to use a pre-trained model to solve our cat vs dog classifier, we can simply denote the pre-trained model as a function $\Phi : a_i \mapsto \mathcal{R}^d$, similar to the naive map ϕ from the previous section. Then, we create our design matrix $X \in \mathcal{R}^d$ and use Logistic Regression to find the most optimal w. This is known as fine-tuning, and you will be able to implement this in the programming assignment.

[3]As you may have seen in previous parts of the course, MSE (and therefore OLS) is not suitable for binary classification for a wide variety of reasons.

[4]Refer to the Week 4 EECS 16ML notes for a refresher on this.

[5]This "naive" featurization on the result is actually analogous to the fully connected layer in the CNN that you saw in the previous lecture.

2 Overview of Common Pre-trained Models

2.1 History

Before Convolutional Neural Networks became so powerful for computer vision, a lot of techniques for tasks like image classification involved hand crafted filters. These filters would convolve over an image and extract information like the location of edges in an image (see Figure 2 for example filters). Given enough enough hand crafted features one could train a decent image classifier, or at least decent by pre 2012 standards. Since then deep learning has taken off and Convolutional Neural Networks are able to learn features more powerful than any human could craft by hand. This is why pertained models are so powerful, they learn really general image features from training on tasks like image classification on a huge dataset of images. These learned image features can than be used to fine-tune a model on some other task, like object detection.

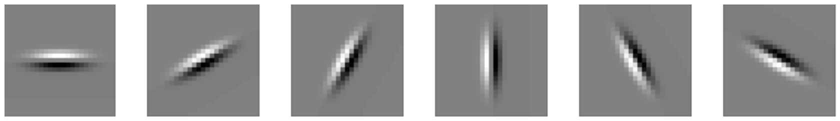

Figure 2: some examples of handcrafted edge detector filters. source: CS189 conv nets lecture

2.2 The Imagenet Dataset

Most state of the art pre-trained models are trained on the Imagenet dataset. This dataset consists of over 14 million color images, and each image is labeled into one of 21,841 categories. These categories are organized into a hierarchical tree structure, starting with high level categories before getting more specific (e.g. natural object -> rock, stone -> chondrite). The categories range from objects like rocks to cars and radiators. The dataset is traditionally linked to the now famous Imagenet challenge. This challenge revolved around three different computer vision tasks: image classification, object localization, and object detection. The classification task is the most well known. All the tasks used a smaller subset of the whole Image-net dataset. The classification dataset specifically used 1000 classes and 1.2 million training examples. Generally state of the art pertained models are trained on some version of this classification Imagenet challenge dataset.

2.3 The learned features

Since the Imagenet dataset has such a diverse set of images, these models learn a highly robust set of features. Just visualizing the filters from the first layer (figure 3), shows that the model has learned things like edge detectors and texture detectors. Some of these look similar to the hand crafted filters that were used before deep learning (see Figure 2), but others totally unique. This is just the first layer of filters, the later layers can only build on these features and extract more nuanced, high-level information. Unfortunately we can't visualize the filters from these later layers quite as easily because they typically have more than three input channels, so they don't really correspond to rgb. [6]

2.4 SOTA models

In 2020 there are so many different pre-trained models. Some are more accurate and others use fewer parameters or run faster. There are generally different pros or cons when considering what pre-trained models to use. Figure 5 plots a bunch of different pre-trained models along 3 different axes, top 1 accuracy, floating point operations when classifying an image, and the number of parameters in the model (the size of the circle). In general larger resents tend to be the best performing models, meanwhile models like MobileNet are a bit less accurate but use fewer parameters and fewer floating point operations. For example, you might want to use MobileNet and take the slight hit in accuracy, if you need your model to run on a phone so it can't take a lot of time or space. There can be a few factors to weigh in deciding what model to use in real world applications, so we'd recommend

[6] If this idea of understanding the inner workings on these networks interested you, then you should check out this blog post. It uses some different techniques to visualize the inner-workings of these pre-trained image classifiers. OpenAI Microscope is also a great resource for exploring this kind of thing.

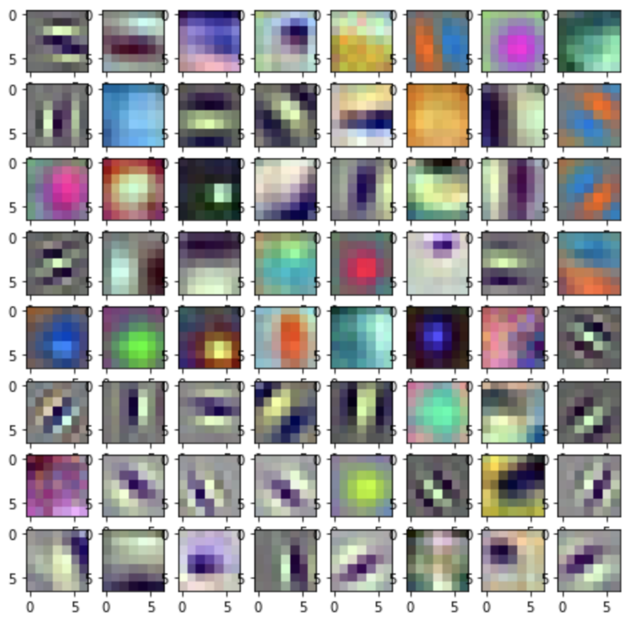

Figure 3: Visualizations of the first layer learned filters from a resnet pretrained on Imagenet.

referencing a chart like Figure 3 and doing some experimentation when deciding. Going with some version of resent is usually a good starting point though.

3 Using pre-trained models

As we have seen previously, many people have put in countless hours to create high quality CNNs. In addition, we have seen the motivation and benefit of using such high quality models. The next natural question then becomes: How do I actually use these models? As we will see in this next section, the way in which you use pre-trained models can vary greatly across tasks. It is important to understand this so that you can apply the correct techniques when solving a task of your own.

3.1 Transfer Learning

In order to understand how to use pre-trained models, first we introduce the idea of transfer learning, since most techniques fall under this category. Transfer learning is a technique where a model trained on one task is "repurposed" for a second task. Pre-trained models are used often to perform transfer learning, as these are both easy to obtain and have high quality features. As outlined in (https://cs231n.github.io/transfer-learning/), there are two major scenarios in which one can use a pre-trained model for transfer learning.

3.2 Pre-trained model as a fixed feature extractor

In this approach, we use a subset of the pre-trained model (Ex: all but the last layer) as a fixed feature extractor. We do not perform any additional training on our pre-trained model (thus our weights are fixed), and instead train a linear classifier on top of these outputs. Some examples of linear classifiers that you have seen are a one layer neural network or a linear SVM.

3.3 Fine-tuning

In contrast to the previous strategy, we do not fix all of the weights of a pre-trained model when fine-tuning. Instead, we fix only the earlier layers of the network (or none at all), and update the later layers of the network (or all layers) while training the network on our new task. The main motivation for this is that earlier layers in a ConvNet are more generalizable (think of the edge detectors and

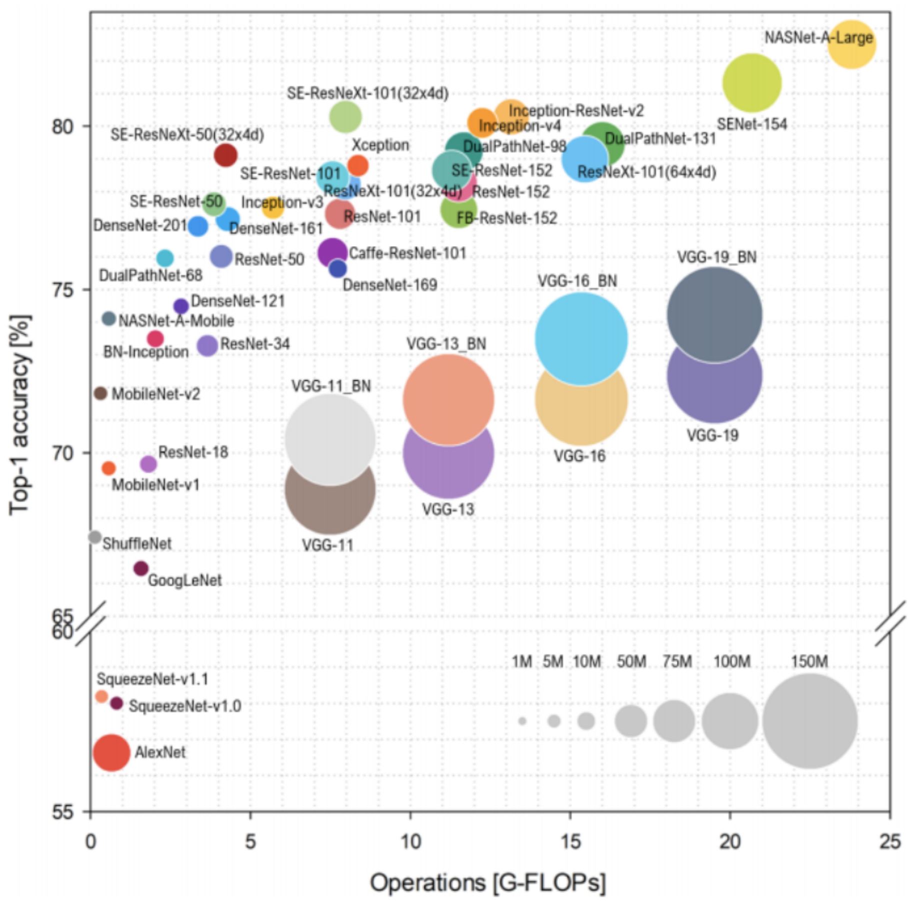

Figure 4: a plot of a bunch of different pre-trained models. The y axis represents top 1 model accuracy, the x axis represents the floating point operations required to classify an image. The size of the circles depicts the number of parameters in each model. source.

other filters we saw earlier in the notes). The layers later in the network are more specialized for the task they were trained for. For example, a model pre-trained on ImageNet would contain many features later in the network that are helpful for differentiating between dog breeds. This may or may not be helpful for your task so it is important to pick the right number of layers to train and layers to freeze when fine-tuning a network.

3.4 When and Where to Use these Methods

We have just explained a couple ways to use pre-trained models. Now it's time to teach you when and when not to use each of these methods. The answer may be very different based on which task you are trying to accomplish, so we have boiled the typical cases down into 4 scenarios: small dataset and similar to pre-trained dataset, small dataset and dissimilar to the pre-trained dataset, large dataset and similar, and large dataset and dissimilar.

3.4.1 Small and Similar

If the dataset is similar to the original dataset, then the features from the later/higher layers of the network should be relevant to our new task. Therefore, we should first try using all or almost all layers of our pre-trained model as a fixed feature extractor and train a linear classifier on these features. We should not fine-tune a large number of layers in this case because we have a small dataset and this could lead to overfitting.

3.4.2 Large and Similar

If the dataset is similar to the original dataset, then we would also expect the features from the higher layers in the network to be helpful. However, since we have a large dataset, overfitting may not be a concern and we can use fine-tuning some or all of the model. If a linear classifier doesn't reach the accuracy you hope for, then fine-tuning may be your best bet.

3.4.3 Small and Dissimilar

In the case where your data is dissimilar to the data your model was pre-trained on (ImageNet is all natural images, so think cartoons, medical imaging, etc.), then we want to extract more general features from our pre-trained network. Extracting features from lower layers in the network may be your best bet. Again, since the dataset is small, we want to stay away from fine-tuning parts of our network for fear of overfitting. Thus, training a linear classifier using features from lower in the network may work best.

3.4.4 Large and Dissimilar

Based on the previous three cases, what would you recommend for a dataset that is large and dissimilar? Make a hypothesis before reading the next section in order to test your intuition.

If your dataset is large, similar to the previous case, we no longer have concerns of overfitting. If your dataset is dissimilar to the original one, then the higher level features may not be very useful. Thus, we will want to fine-tune large portions of our model so that we can learn more meaningful representations for our task. Depending on how dissimilar your data is, you may just want to fine-tune the entire model and treat the pre-trained weights as more of an "intelligent weight initialization." This may also bring up the question of whether you need the pre-trained model at all. This question is closely related to negative transfer, which is out of scope for this unit, but may be worth looking into if you encounter this case. As a general note, if you suspect your model is not benefitting from the pre-trained weights and pre-decided model architecture, it may be better to just train a ConvNet from scratch.

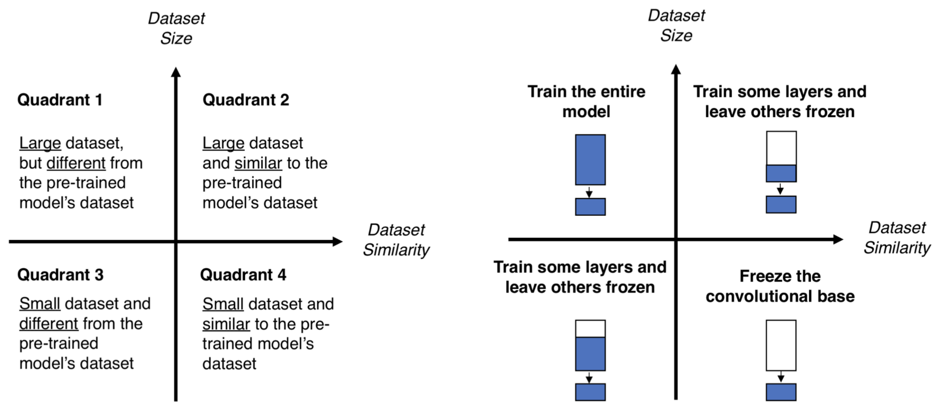
Dataset Size
Quadrant 1
Large dataset, but different from the pre-trained model's dataset
Quadrant 2
Large dataset and similar to the pre-trained model's dataset
Dataset Similarity
Quadrant 3
Small dataset and different from the pre-trained model's dataset
Quadrant 4
Small dataset and similar to the pre-trained model's dataset
Dataset Size
Train the entire model
Train some layers and leave others frozen
Dataset Similarity
Train some layers and leave others frozen
Freeze the convolutional base

Figure 5: an outline of the different strategies for transfer learning source.