**Transfer Learning**

When building neural networks, starting from scratch can be time consuming.

It involves building the architecture, experimenting with it, training it and evaluating it.

It is much better to fine tune an existing network. Fine tuning one takes much less effort than creating one from scratch.

We can also take an existing network and repurpose it to a related but different task.

Repurposing a network is called **Transfer Learning**.

**Transfer Learning** takes learning from one network to another network.

CPU vs GPU

GPUs optimize for high throughput computation while CPUs are optimized for latency trying to run a single thread of instructions as quickly as possible. GPUs try and run as many simultaneous computations as possible.

That is import for pixel graphics but also deep learning. That is because the computations fundamental to deep learning have a lot of parallelism.

Networks generally train 5 times faster on a GPU than on a CPU.

Getting fast feedback can also be done using Transfer Learning.

**Transfer Learning:** Taking a pre-trained neural network and adapting the neural network to a new, different data set. This depends on the size of the new dataset, and the similarity of the new data set to the original data set.

Start with an existing network built for a similar task and then fine tune it for your problem.

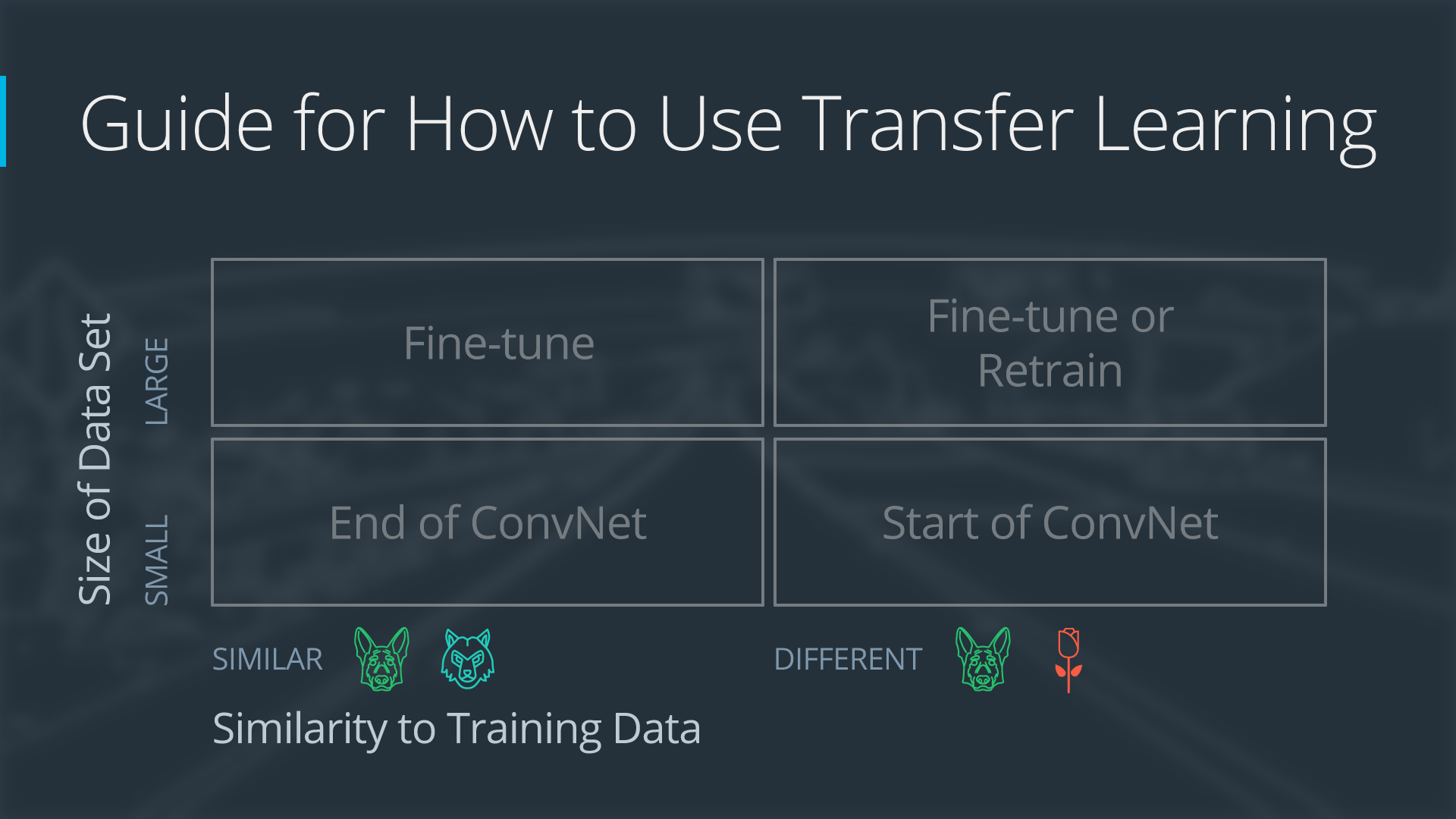
This is useful because, if someone has taken days or even weeks to train a network already, there is a lot of intelligence stored in that network which can be taken advantage of.

Sometimes the dataset you are working with might be small, we can look for a exiting network designed for a problem similar to ours. If it is trained on a larger dataset, it can help our network generalize better.

The approach for using transfer learning is different depending on the size of the new dataset, and the similarity between the new dataset and your original dataset.

The Four cases are:

1. New data set is small, new data set is similar to original training data
2. New data set is small, new data is different from original training data
3. New data set is large, new data is similar to original training data
4. New Data set is large, new data is different from original training data

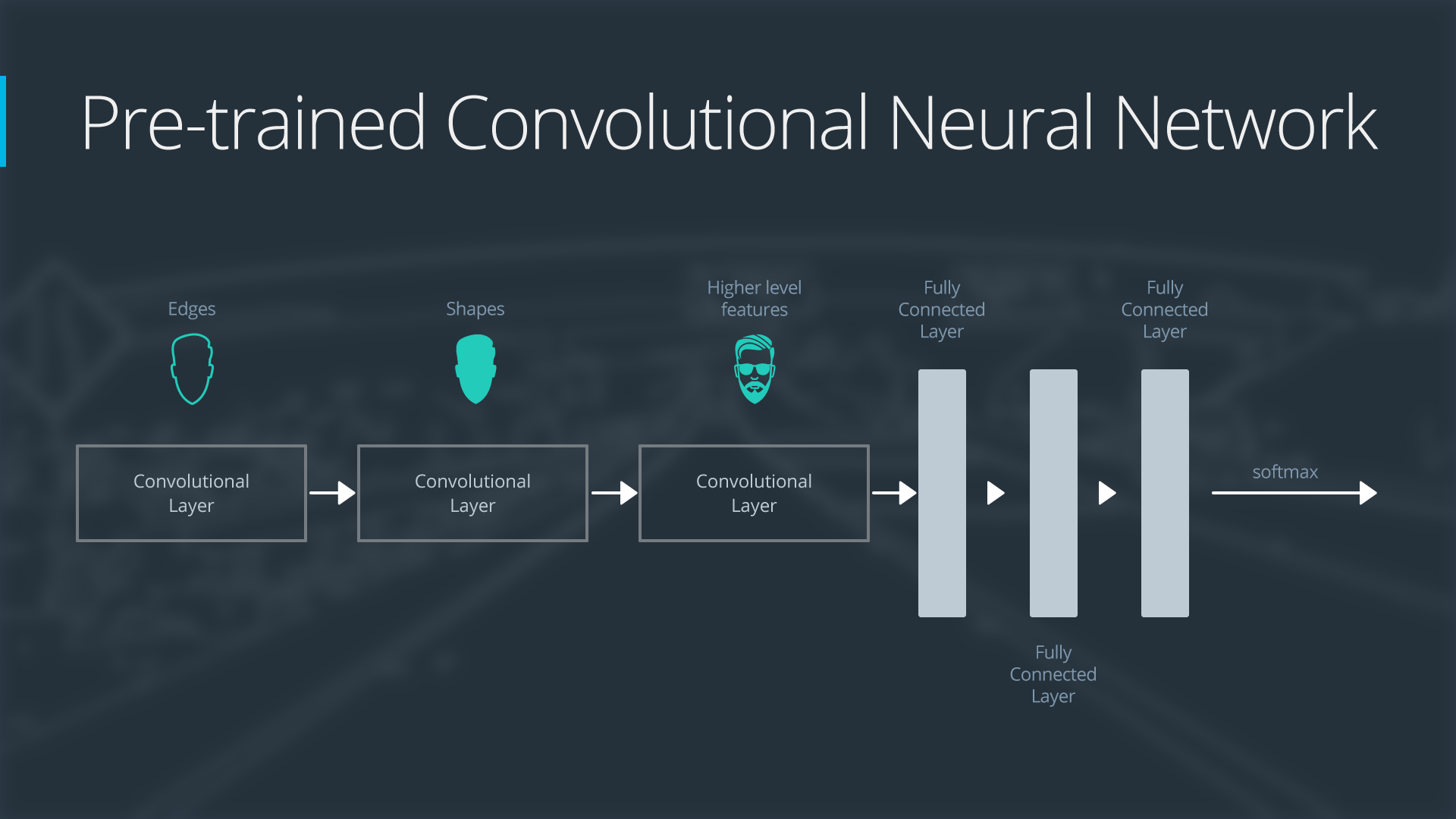


The line between a large and a small data set is somewhat subjective, but the concern is with over fitting when using transfer learning on a smaller dataset. (Ex. 1Million images is large, but 2Thousand is small)

Similar Training data (Images of Dogs and images of wolves are considered similar, but data set of flowers and dogs would be different)

Each case has its own approach.

Lets start with a CovNet that has 3 convolutional layers and three fully connected layers.

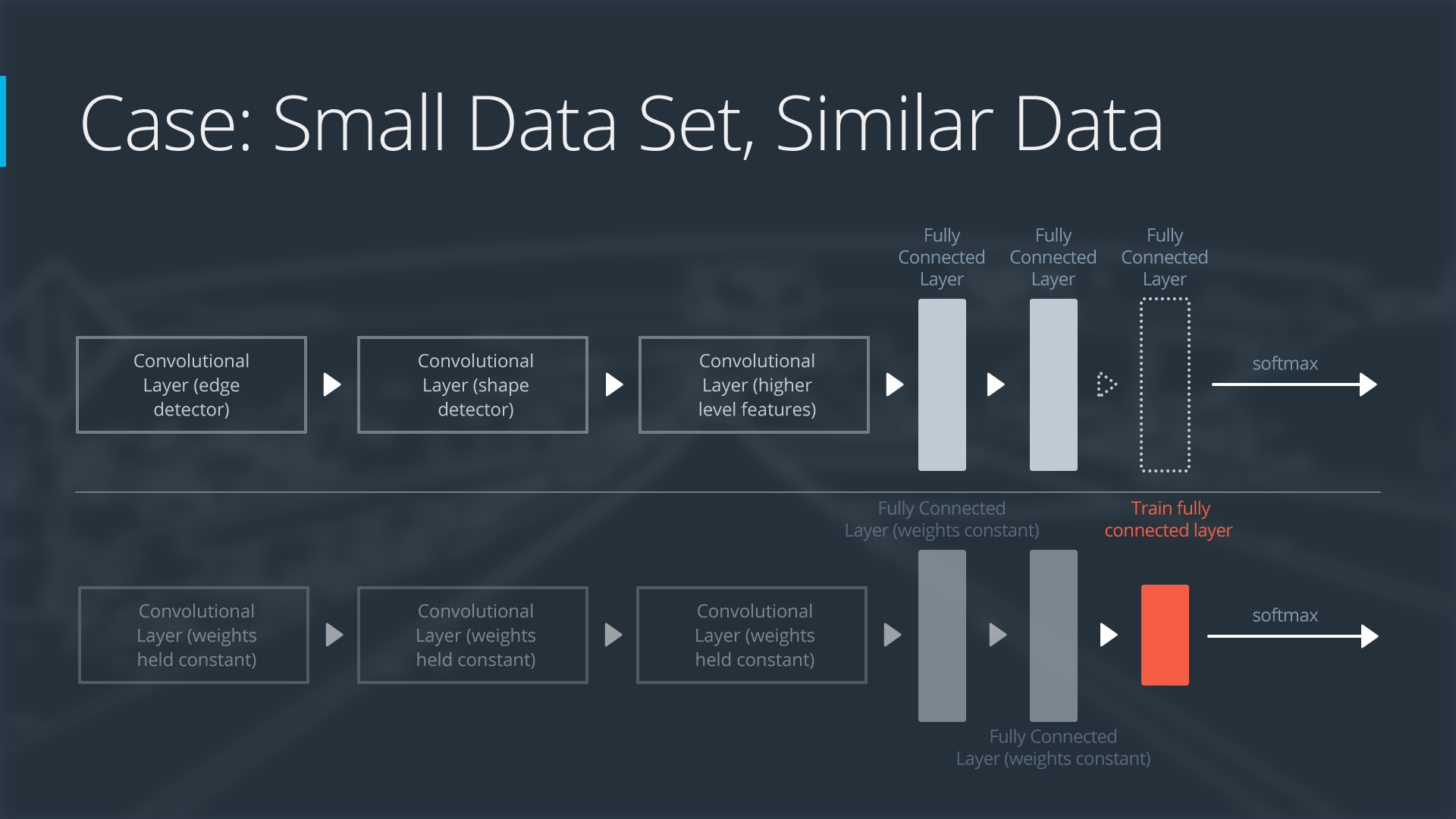


**Case 1: New Small Data Set, Similar Data with New**

Slice off the end of the neural network, add a new fully connected layer that matches the number of classes in the new data set, randomize the weights of the new fully connected layer, train the network to update the weights of the new fully connected layer.

To avoid overfitting on the small data set, the weights of the original network will be held constant.

Since datasets are similar, images from each data set will have similar higher level features, so the pre-trained neural network layers already contain relevant information about the new data set and should be kept.



**Case 2: New Small Data Set, Different Data than New**

Slice off most of the pre-trained layers at the beginning of the network,

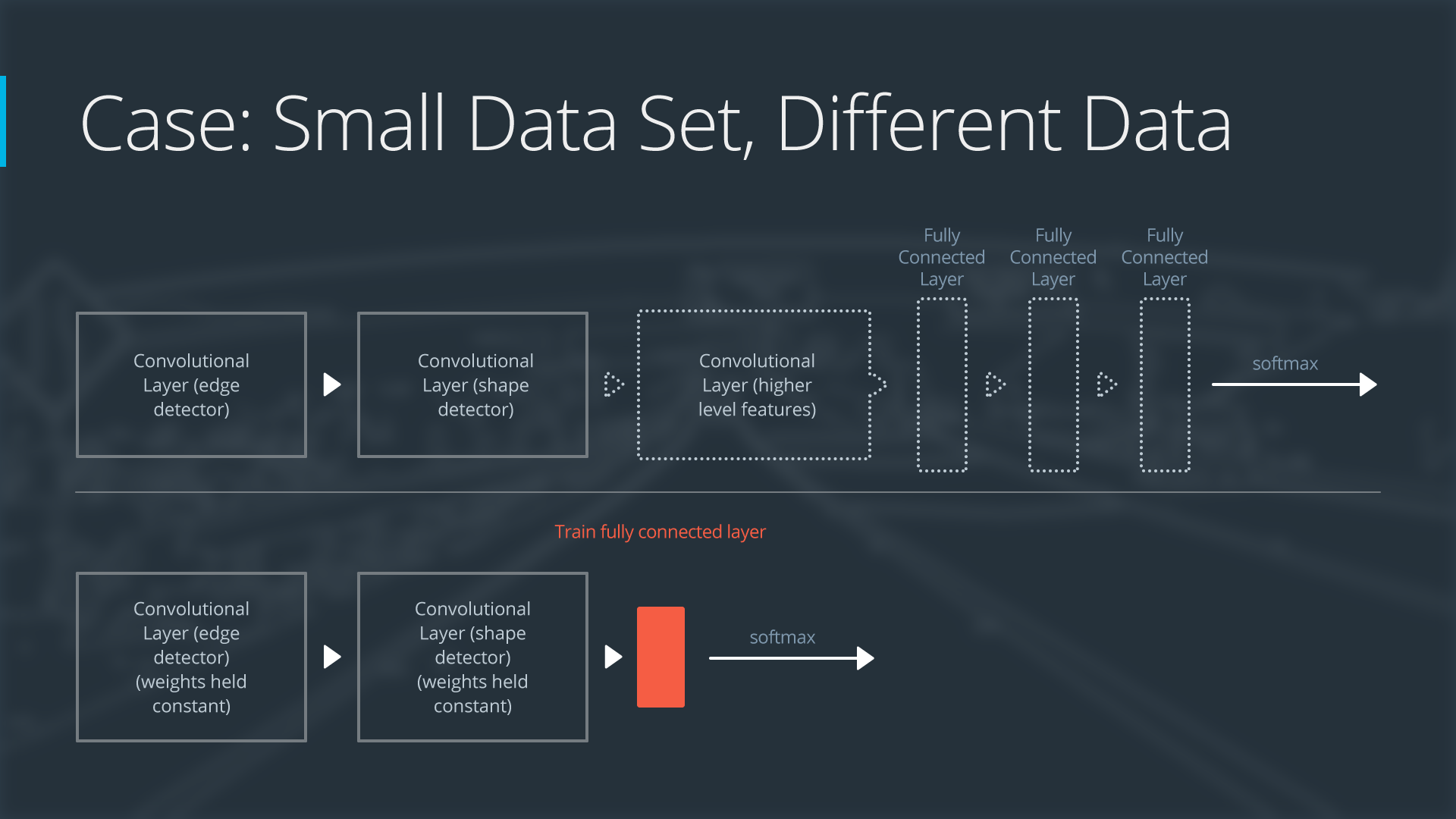
add to the remaining pre-trained layers a new fully connected layer that matches the number of classes in the new dataset,

randomize the weights of the new fully connected layer while freezing the weights from the pre-trained network,

Train the network to update the weights of the new fully connected.

We keep the weights of the original neural network constant like the first case because of the small data set.

Because the original training set and the new data don’t share higher level features, so the new network will only use the layers containing lower level features.



Case 3: New Large Data Set, Similar Data New

Remove the last fully connected layer and replace with a layer matching the number of classes in the new dataset.

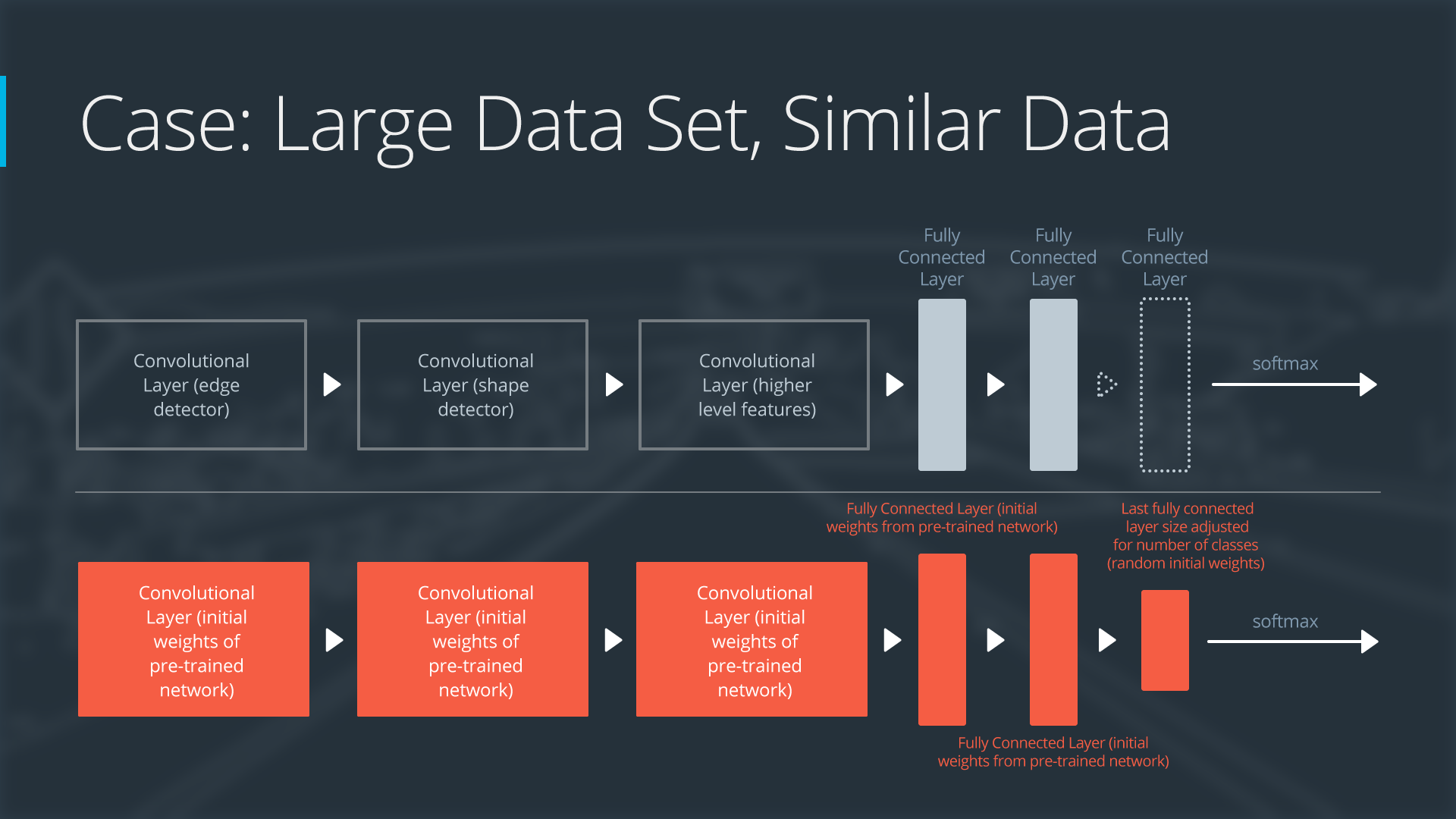
Randomly Initialize the weights in the fully connected layer,

Initialize the weights using pre-trained weights

Re-train the entire neural network.

Overfitting isn’t much of a concern with larger data sets, you can re-train all of the weights,

Because the original training set and the new one share high level features, the entire neural network is used as well.

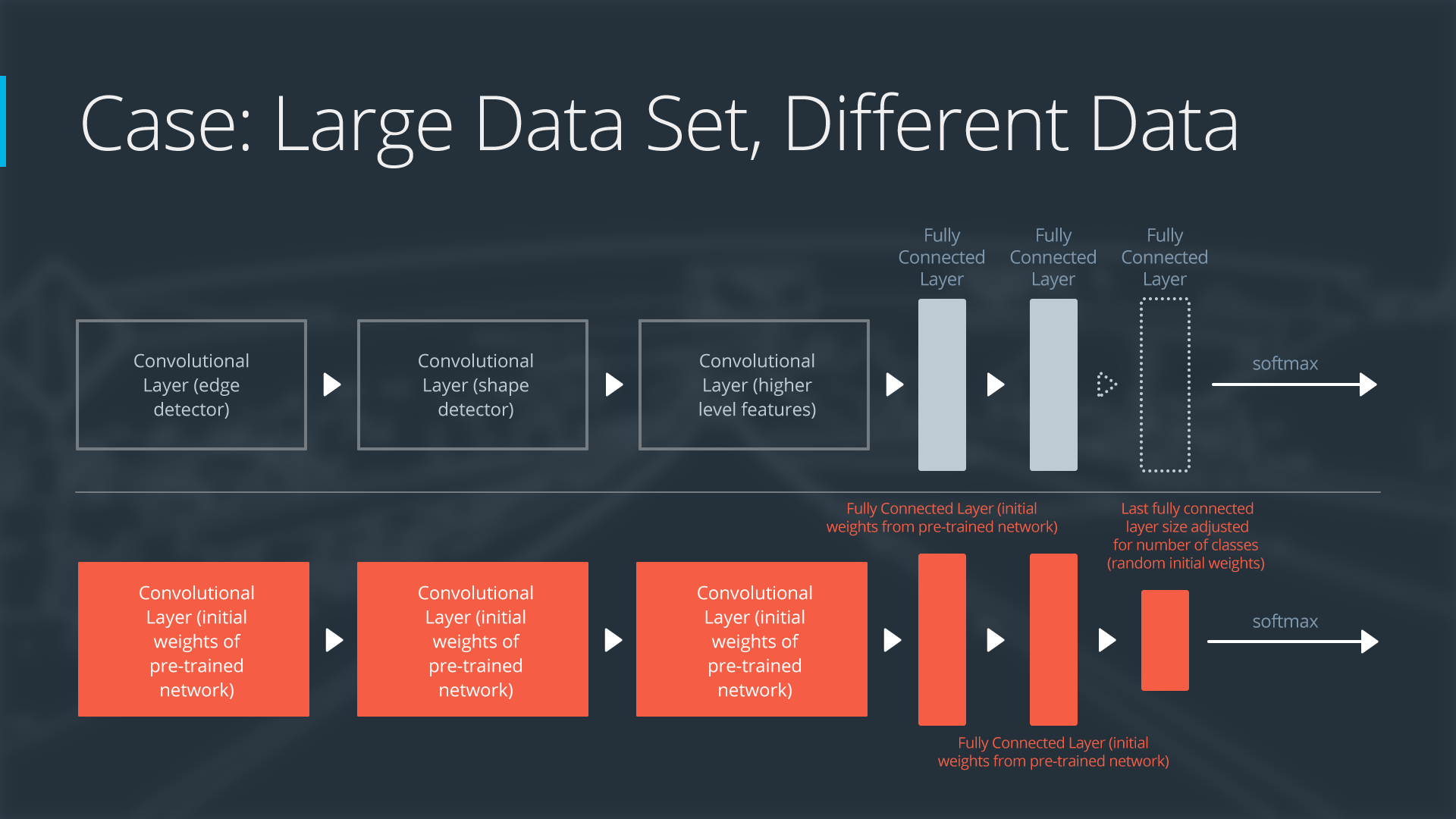


**Case 4: New Large Data Set, Different Data than New**

Remove the last fully connected layer and replace with a layer matching the number of classes in the new data set, retrain the network from scratch with randomly initialized weights, alternatively, you could just use the same strategy as the “large and similar” data case.

Even though the data set is different from the training data, initializing the weights from the pre-trained network might make training faster. So it is a similar case to the large, similar data set.

If using the pre-trained network as a starting point does not produce a successful model, another option is to randomly initialize the convolutional network weights and train them from scratch.



**So here are some popular already existing networks.**

**ImageNet, AlexNet, VGG, Empirics, GoogLeNet, ResNet**

Over time, Images became more present and services like amazon’s mechanical turk made it more cost effective to hand label images. These databases led to the creation of ImageNet.

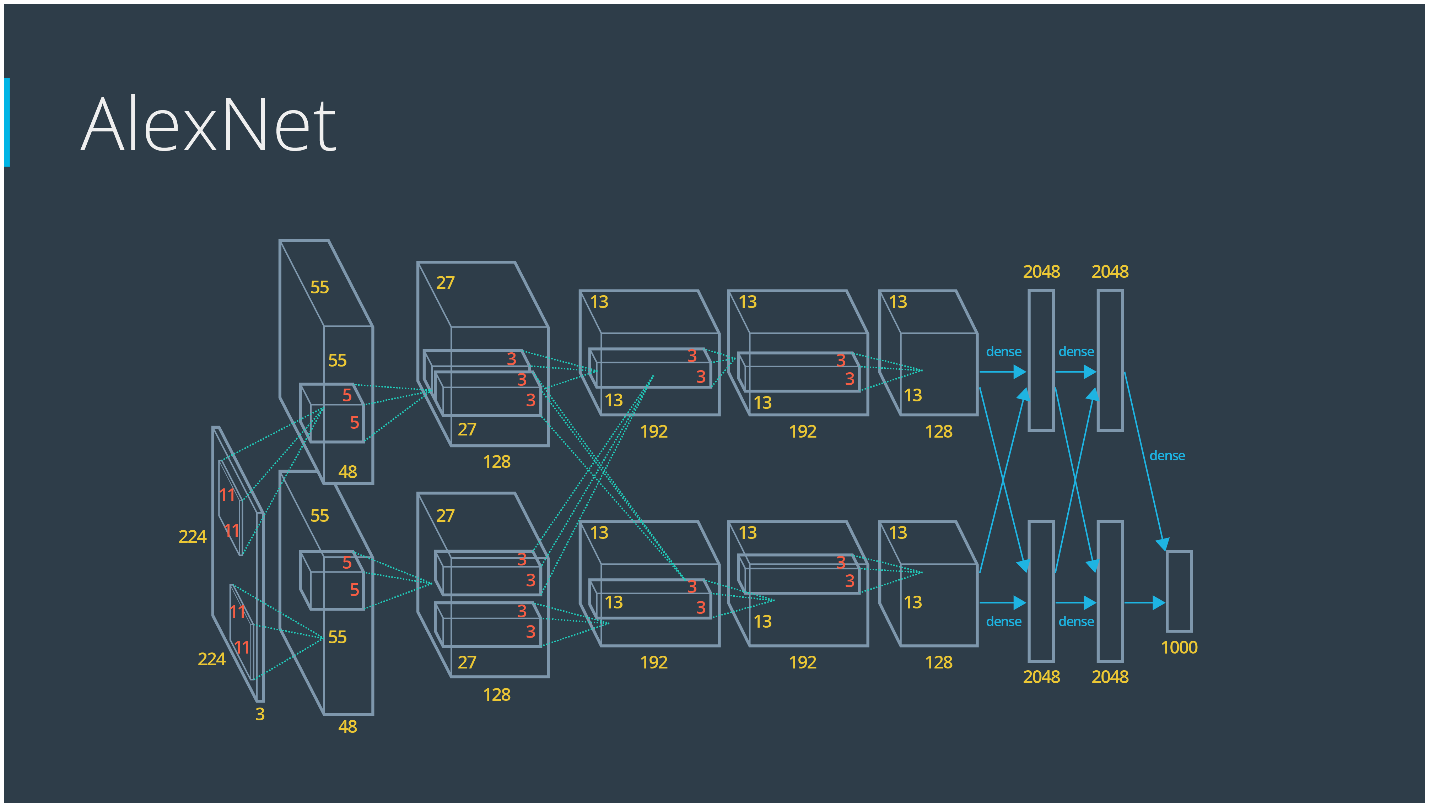
**ImageNet:** A large database of labeled Images

Now there is an **ImageNet Large Scale Visual Recognition competition** is an annual competition to build the best networks for object detection and localization.

AlexNet came out of this competition in 2012 as a breakthrough

**AlexNet:**

* Used the best GPUs available in 2012,
* It pioneered Dropout and using RELU as an activation function
* Put the network on two GPs, which allows for building a larger network
* Original Research Paper stated that parallelizing the network decreased the classification error rate by 1.7% when compared to a neural network that used half as many neurons on one GPU.
* Has 1000 classes as the output due to being trained to classify the ImageNet Database.



Overview:

Two popular methods of applying transfer learning are feature extraction and fine-tuning.

1. **Feature Extraction:** Take a pre-trained neural network and replace the final (classification) layer with a new classification layer, or perhaps even a small feed-forward network that ends with a new classification layer. During training the weights in all the pre-trained layers are frozen, so only the weights for the new layer(s) are trained. That means the gradient doesn’t flow backwards past the first new layer.
2. **Fine-Tuning:** This is similar extraction expect the pre-trained weights aren’t frozen. The network is trained end to end.

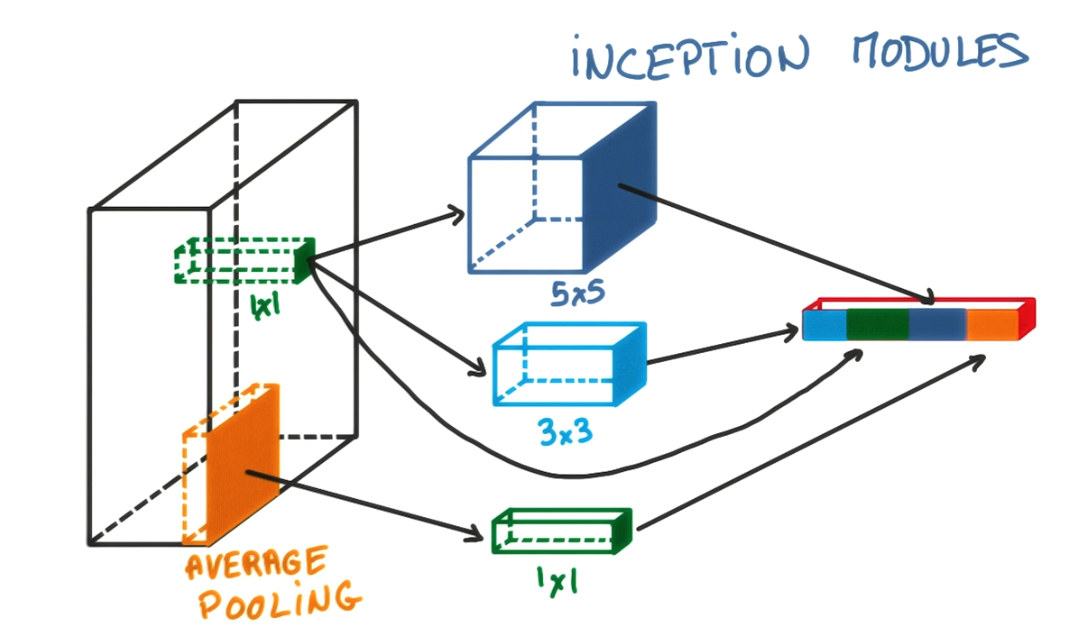
AlexNet is used as a good starting point still today even though there are new architectures that have better results. A simplified version of AlexNet that removes some features that were found to be unnecessary is used. Most implementations of AlexNet found online is the simplified version.

**VGG:**

* Introduced in the in 2014 as a part of the ImageNet competition.
* Introduced by the Visual Geometry group in Oxford University
* Has a simple yet elegent architecture (Great for Transfer learning)
* A series of 3x3 Convolutional layers broken up with 2x2 pooling layers, and ending with a trio of fully connected layers at the end
* Great as a starting point for image Classification

**GoogLeNet:**

* Also in the 2014 ImageNet, Google introduced this Architecture that was slightly better than VGG
* Runs very fast is the main Idea
* Uses the concept of an **Inception Module**
* Its speed makes it useful for real time.



Uses all these methods instead of choosing one, and then concatenates the answers. You can choose the parameters for each so that the total number of parameters is small.

**ResNet:**

* 2015 ImageNet winner from Microsoft Research
* Has 152 Layers in its Network. (ALOT)
* Same structure is repeated over and over again, but it has connections in the neural network that skips layers, so that very deep neural networks can practically be trained.
* Achieves a loss of 3%, which is actually better than human accuracy

**Tips:**

Even though some networks may have more classes, some datasets may be harder to classify. For example, the traffic sign classifier had 43 classes, while if we look at the example of the Cifar10 dataset, which has 10 classes, it is more complex to classify. This is due to the complexity of those classes.

A ship is drastically different from a frog or a deer, etc. That drastic difference between classes is what makes harder to classify. These are the kinds of datasets where the advantage of using a pre-trained model are much more apparent.

**Transfer Learning Lab Notes:**

To end this lab, let's summarize when we should consider:

1. Feature extraction (train only the top-level of the network, the rest of the network remains fixed)
2. Finetuning (train the entire network end-to-end, start with pre-trained weights)
3. Training from scratch (train the entire network end-to-end, start from random weights)

Consider feature extraction when ...

... the new dataset is small and similar to the original dataset. The higher-level features learned from the original dataset should transfer well to the new dataset.

Consider finetuning when ...

... the new dataset is large and similar to the original dataset. Altering the original weights should be safe because the network is unlikely to overfit the new, large dataset.

... the new dataset is small and very different from the original dataset. You could also make the case for training from scratch. If you choose to finetune, it might be a good idea to only use features from the first few layers of the pre-trained network; features from the final layers of the pre-trained network might be too specific to the original dataset.

Consider training from scratch when ...

... the dataset is large and very different from the original dataset. In this case we have enough data to confidently train from scratch. However, even in this case it might be beneficial to initialize the entire network with pretrained weights and finetune it on the new dataset.

Finally, keep in mind that for a lot of problems you won't need an architecture as complicated and powerful as VGG, Inception, or ResNet. These architectures were made for the task of classifying thousands of complex classes. A smaller network might be a better fit for a smaller problem, especially if you can comfortably train it on moderate hardware.