1. CPU更加注重单线程更快执行，低延迟；

GPU更加关注吞吐量，并行执行；

同样配置，GPU比CPU快5-10倍；

1. 已有的网络，可能是他人花了数周，数月训练的网络，已经相当智能，用已有的网络，可以更快的加速自己的工作进程。

可能自己手上训练集很小，这时候最好找一下是否已经有一些网络可以解决类似问题；如果存在类似，并且在大数据集上训练的网络，以此为基础，让你的网络取得更好的训练效果。

The Four Main Cases When Using Transfer Learning

Transfer learning involves taking a pre-trained neural network and adapting the neural network to a new, different data set.

Depending on both:

the size of the new data set, and

the similarity of the new data set to the original data set

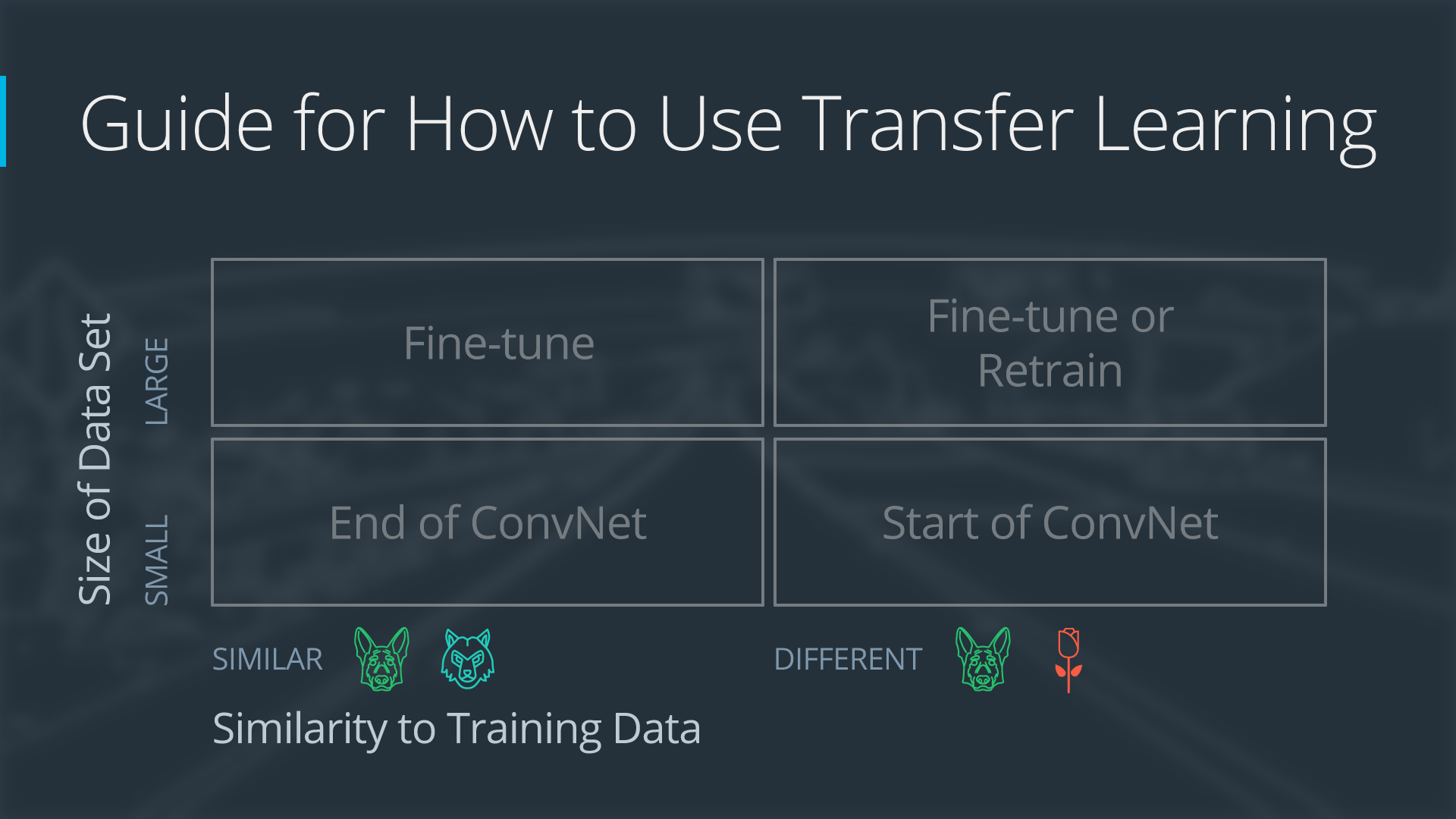
the approach for using transfer learning will be different. There are four main cases:

new data set is small, new data is similar to original training data

new data set is small, new data is different from original training data

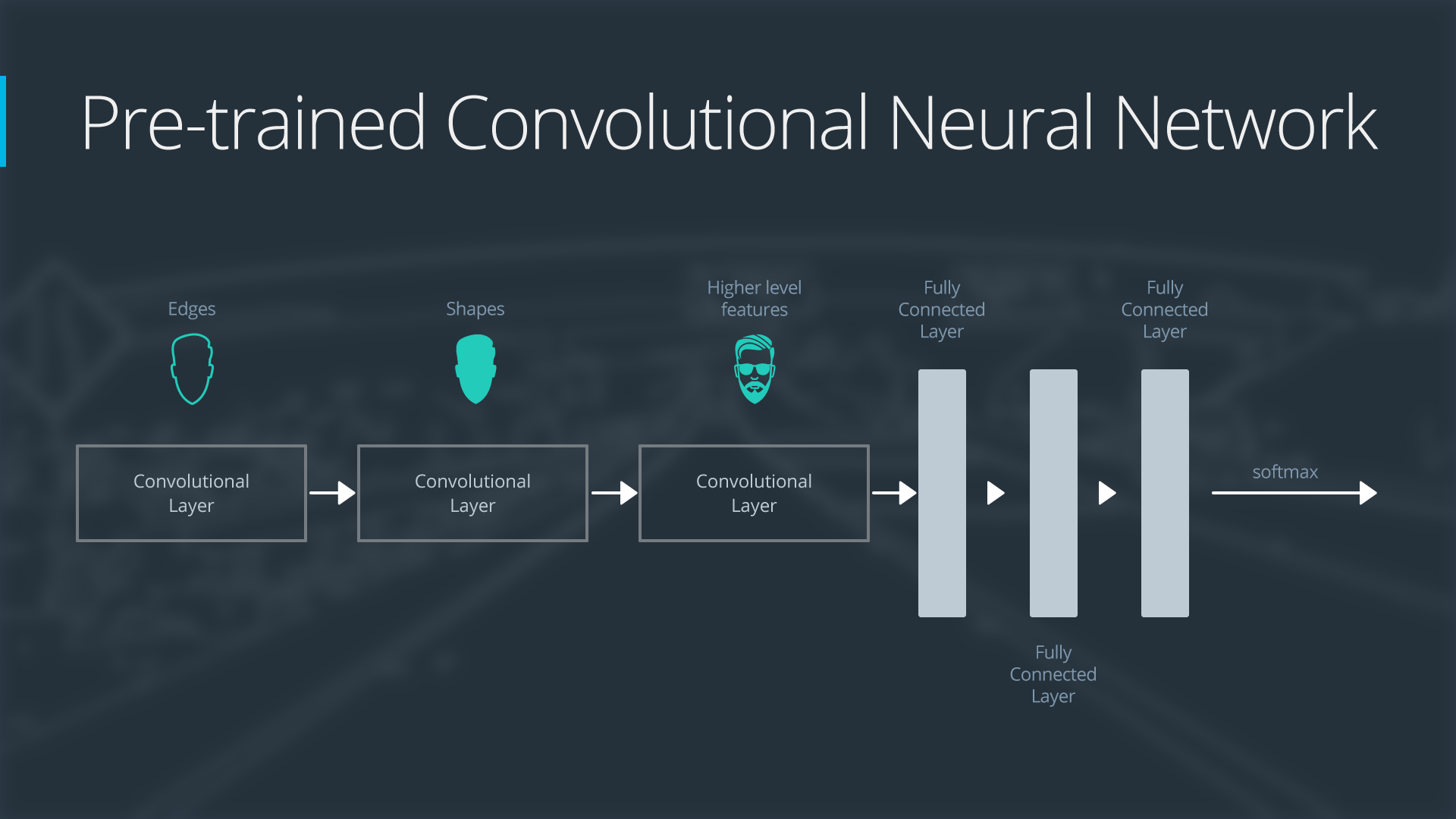
new data set is large, new data is similar to original training data

new data set is large, new data is different from original training data



迁移学习案例：

原始网络

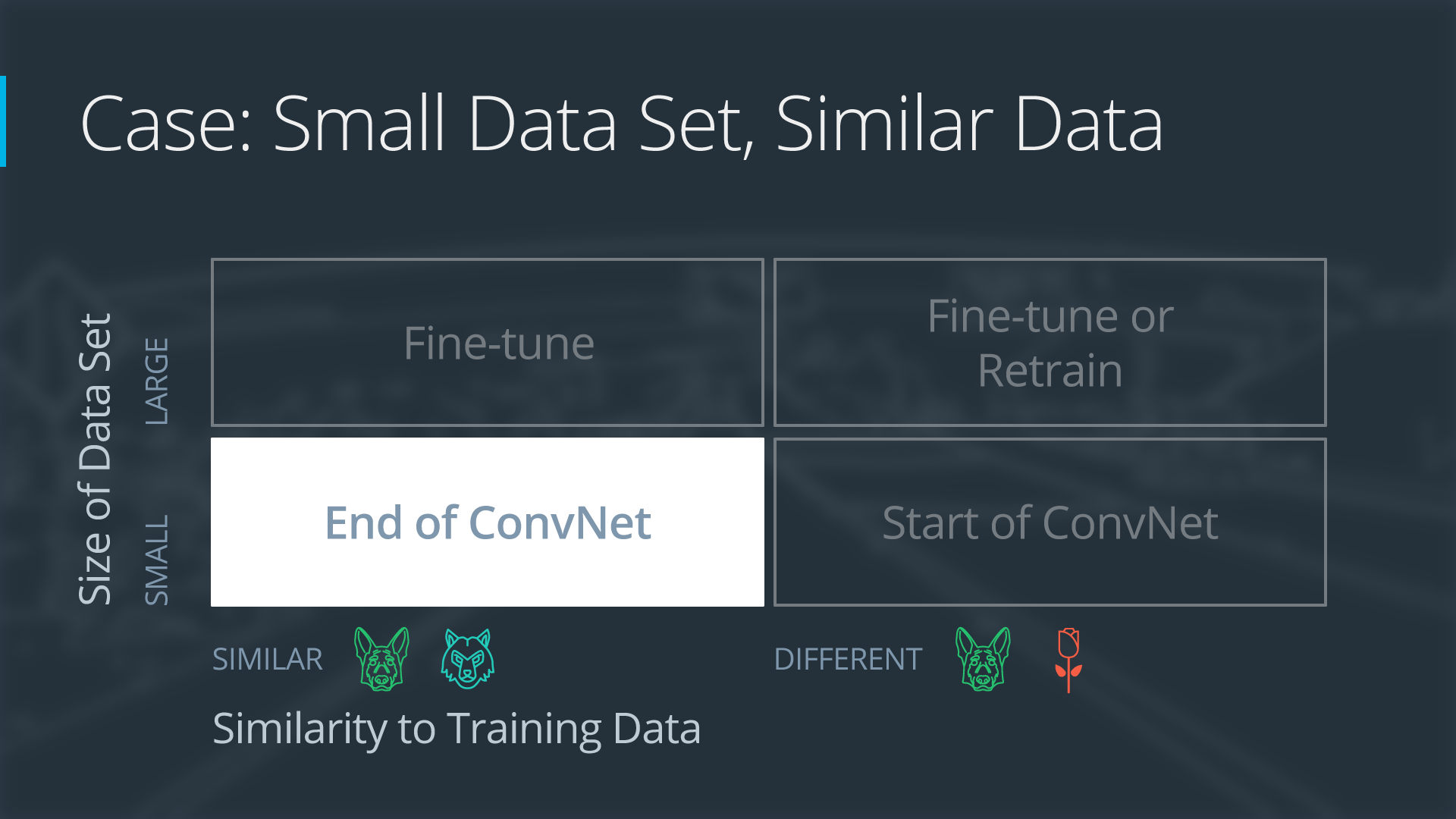


Here is an generalized overview of what the convolutional neural network does:

* the first layer will detect edges in the image
* the second layer will detect shapes
* the third convolutional layer detects higher level features

Each transfer learning case will use the pre-trained convolutional neural network in a different way.

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



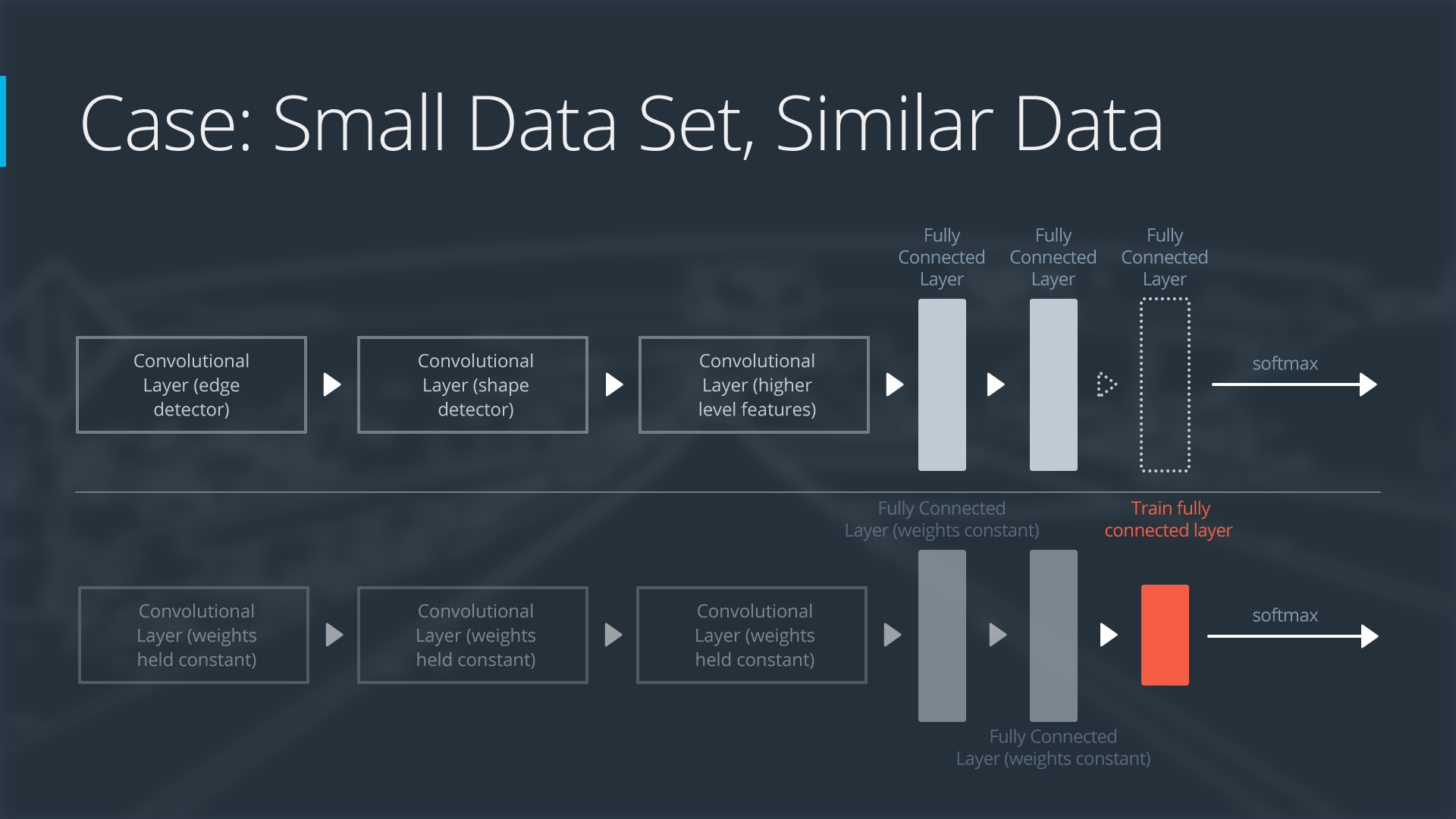
If the new data set is small and similar to the original training data:

* 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; freeze all the weights from the pre-trained network
* train the network to update the weights of the new fully connected layer

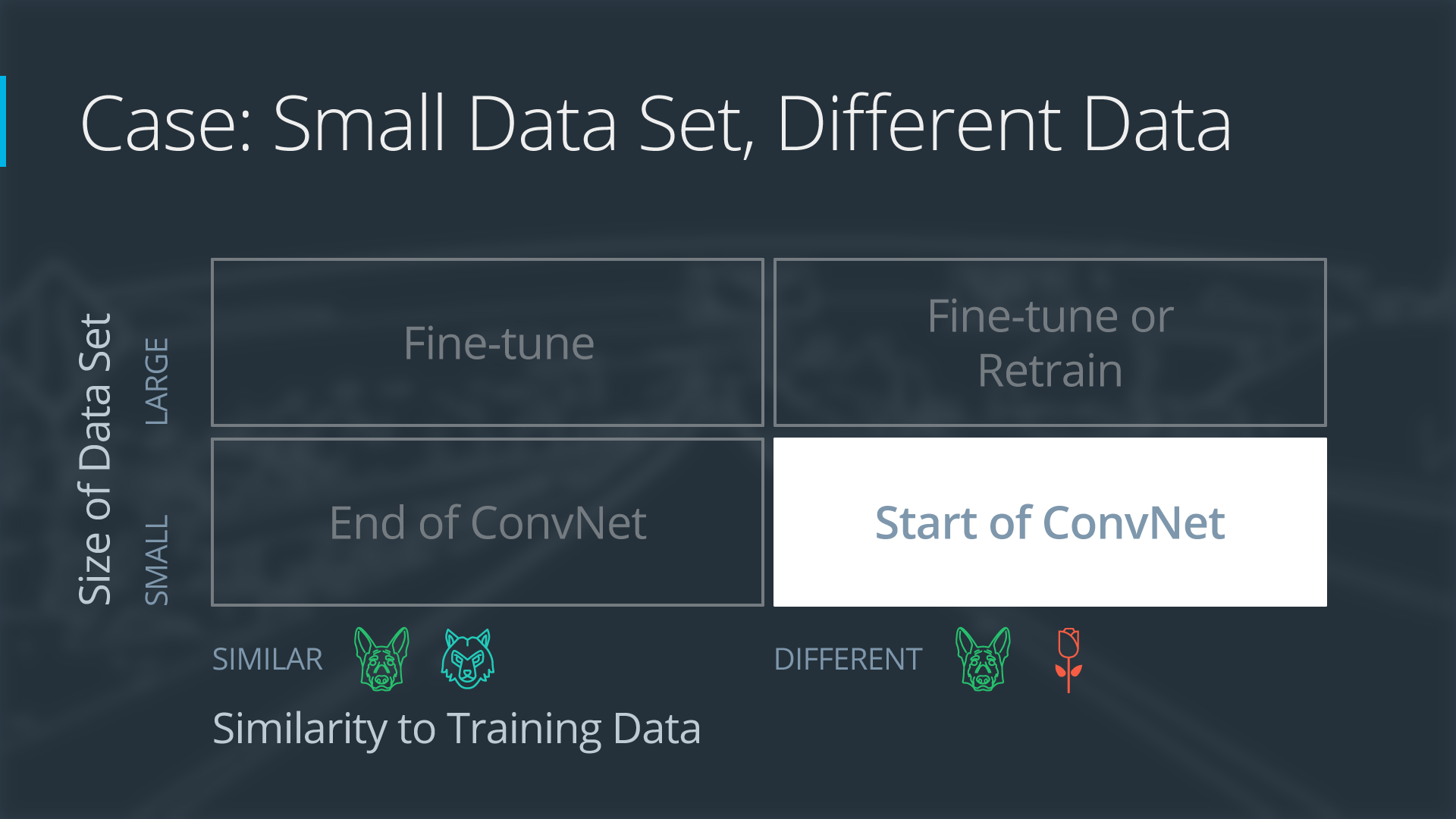
To avoid overfitting on the small data set, the weights of the original network will be held constant rather than re-training the weights.

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

Here's how to visualize this approach:



### Case 2: Small Data Set, Different Data



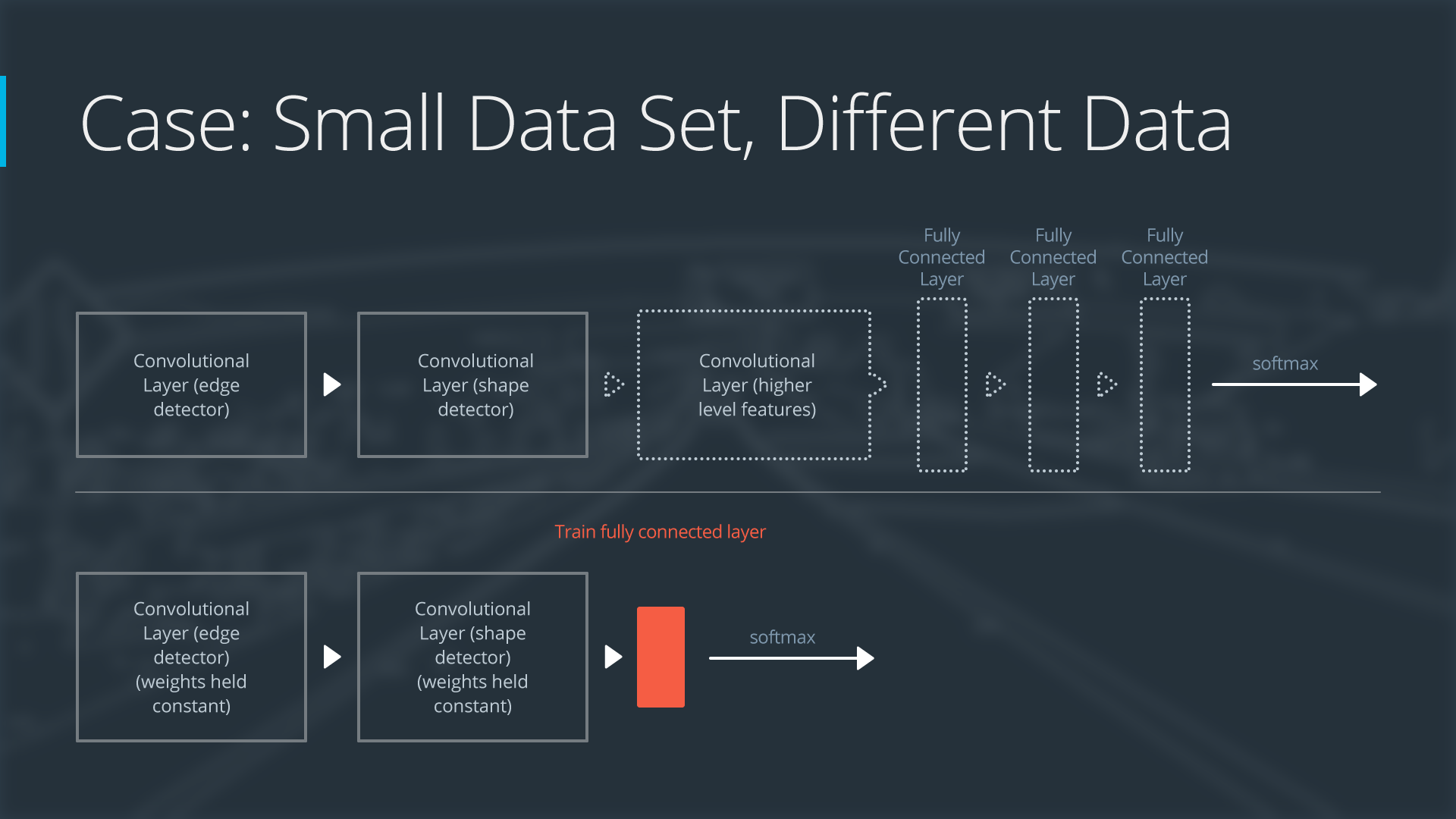
If the new data set is small and different from the original training data:

* slice off most of the pre-trained layers near 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 data set
* randomize the weights of the new fully connected layer; freeze all the weights from the pre-trained network
* train the network to update the weights of the new fully connected layer

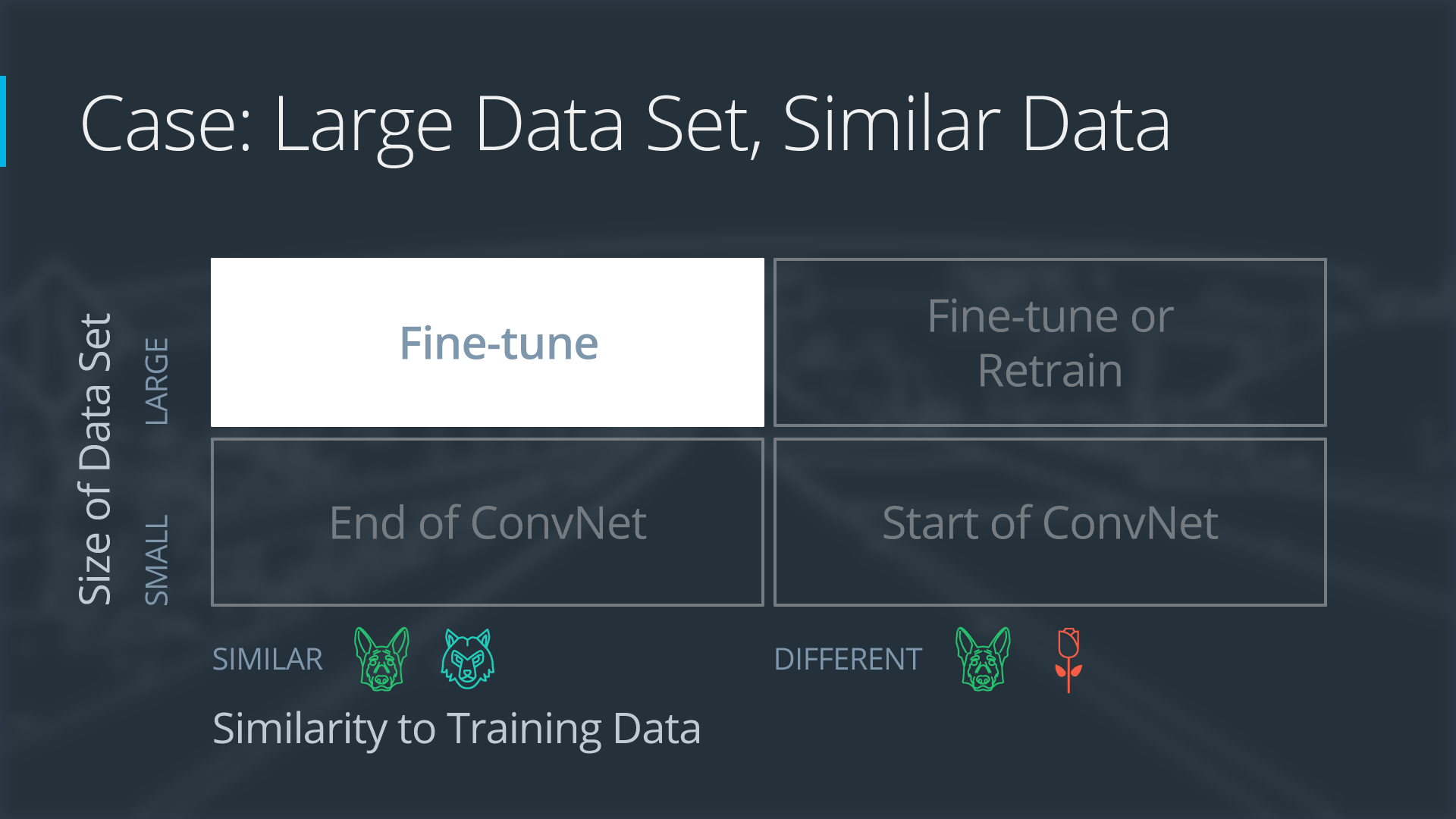
Because the data set is small, overfitting is still a concern. To combat overfitting, the weights of the original neural network will be held constant, like in the first case.

But the original training set and the new data set do not share higher level features. In this case, the new network will only use the layers containing lower level features.

Here is how to visualize this approach:



### Case 3: Large Data Set, Similar Data



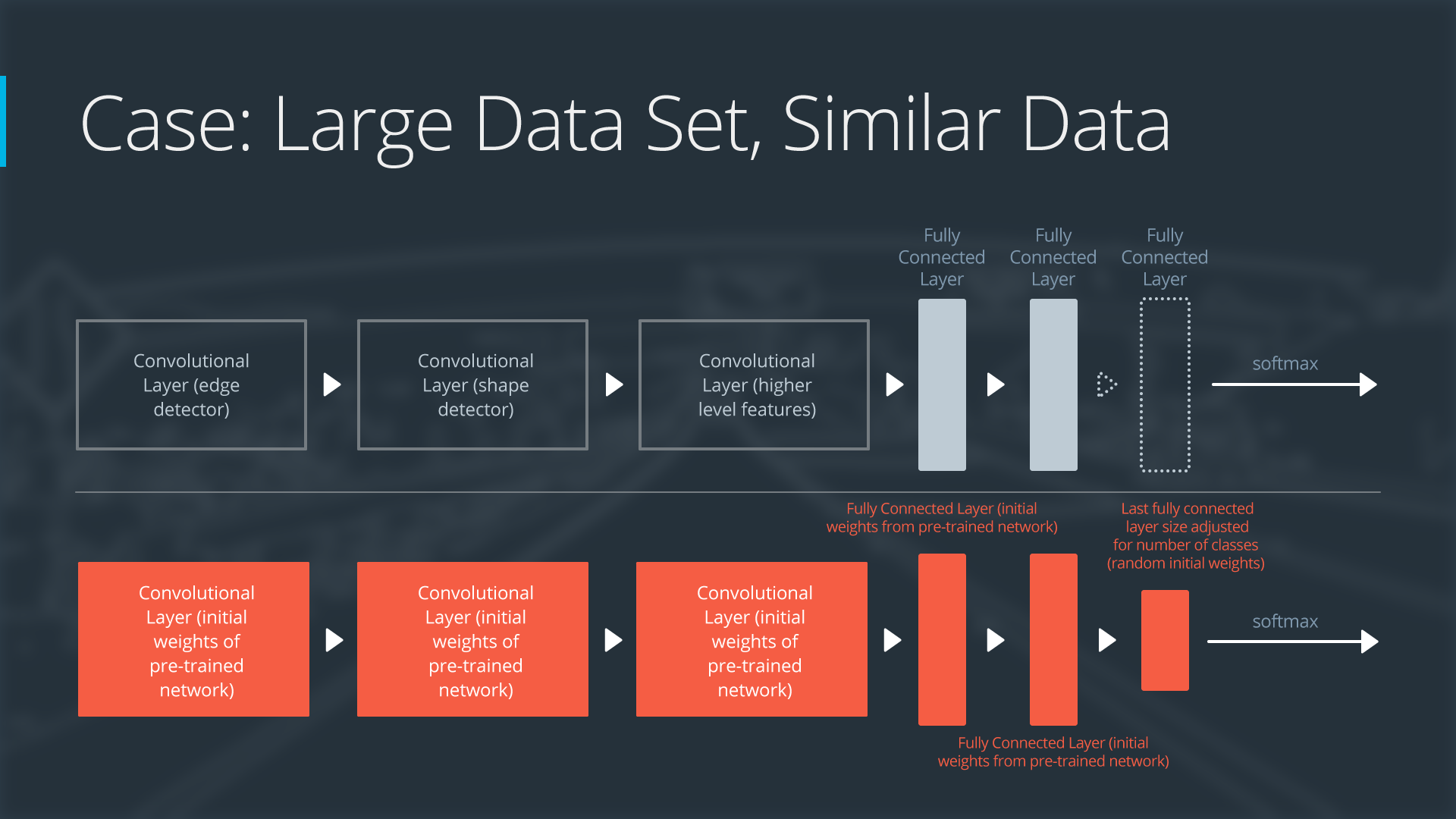
If the new data set is large and similar to the original training data:

* remove the last fully connected layer and replace with a layer matching the number of classes in the new data set
* randomly initialize the weights in the new fully connected layer
* initialize the rest of the weights using the pre-trained weights
* re-train the entire neural network

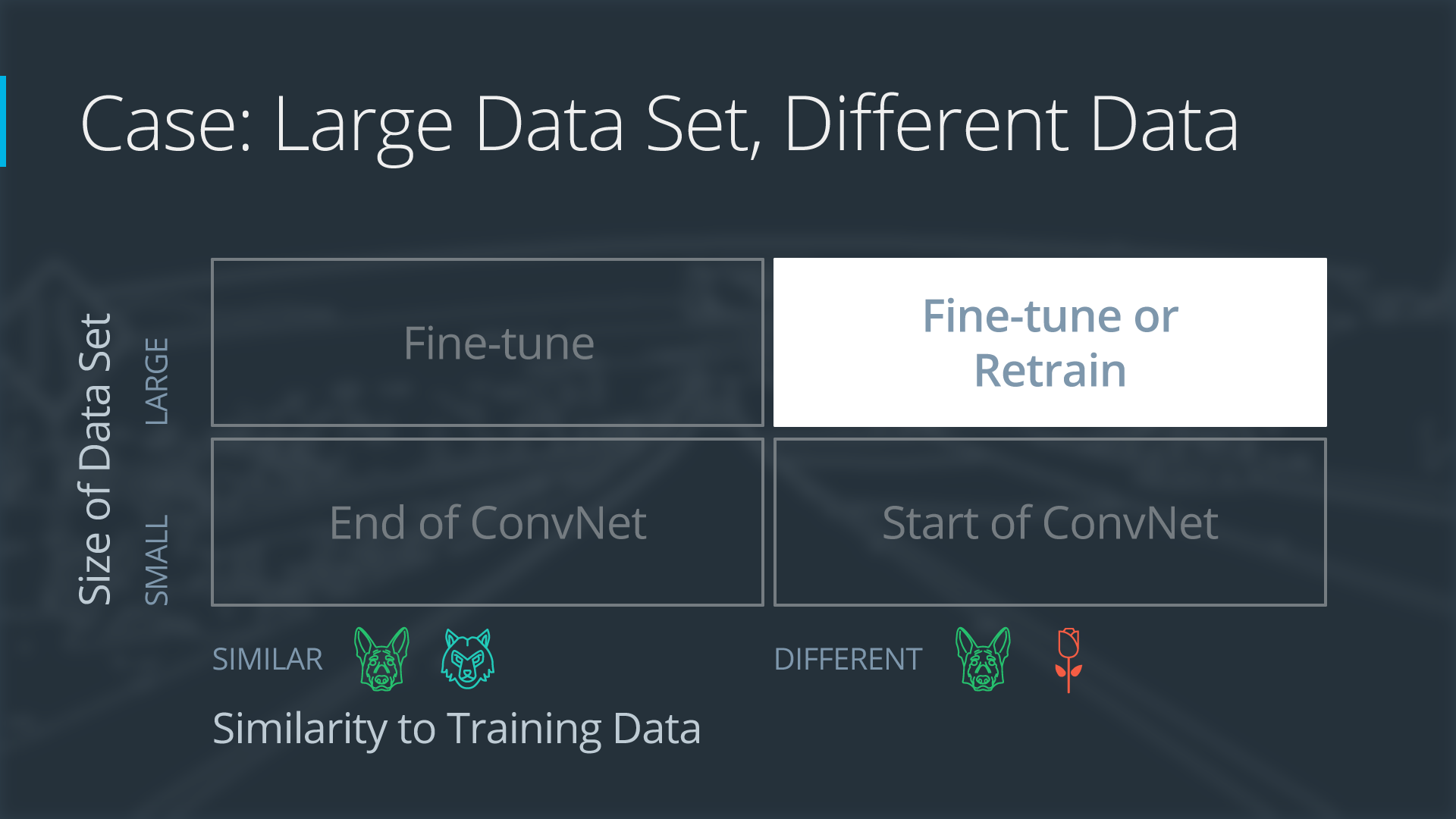
Overfitting is not as much of a concern when training on a large data set; therefore, you can re-train all of the weights.

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

Here is how to visualize this approach:

[](https://classroom.udacity.com/nanodegrees/nd013/parts/edf28735-efc1-4b99-8fbb-ba9c432239c8/modules/6b6c37bc-13a5-47c7-88ed-eb1fce9789a0/lessons/818a5b8e-44b3-42f9-9921-e0e0e49f104e/concepts/10489223-72fa-4393-848b-f882ba3cf7f9)

### [Case 4: Large Data Set, Different Data](https://classroom.udacity.com/nanodegrees/nd013/parts/edf28735-efc1-4b99-8fbb-ba9c432239c8/modules/6b6c37bc-13a5-47c7-88ed-eb1fce9789a0/lessons/818a5b8e-44b3-42f9-9921-e0e0e49f104e/concepts/10489223-72fa-4393-848b-f882ba3cf7f9)



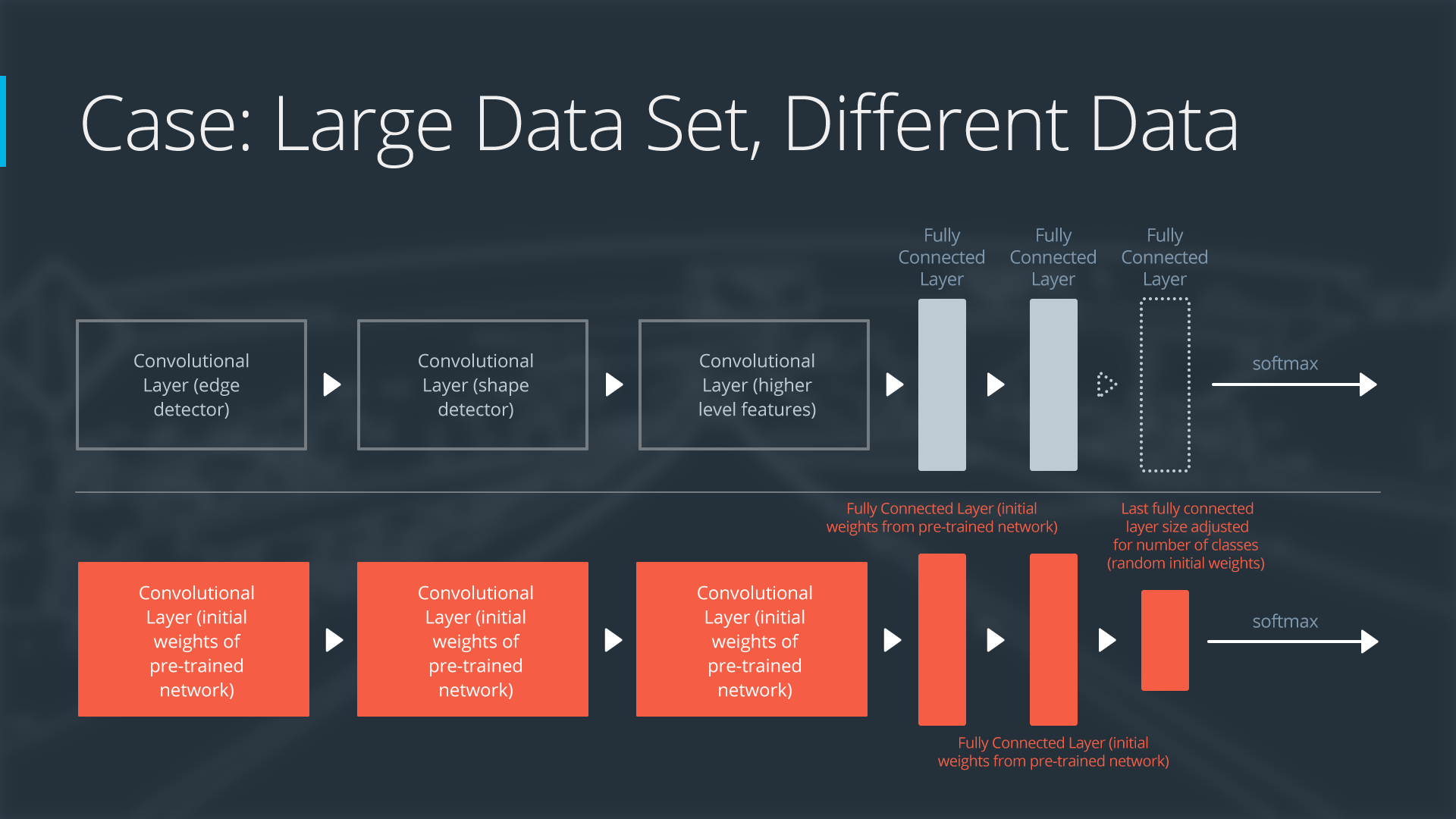
If the new data set is large and different from the original training data:

* 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 this case is exactly the same as the case with a 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 neural network weights and train the network from scratch.

Here is how to visualize this approach:



1. Deep LearningHistory

深度学习最近五年兴起主要原因，海量可使用的标记数据，计算能力；

1. ImageNet

<https://www.kaggle.com/c/imagenet-object-localization-challenge>

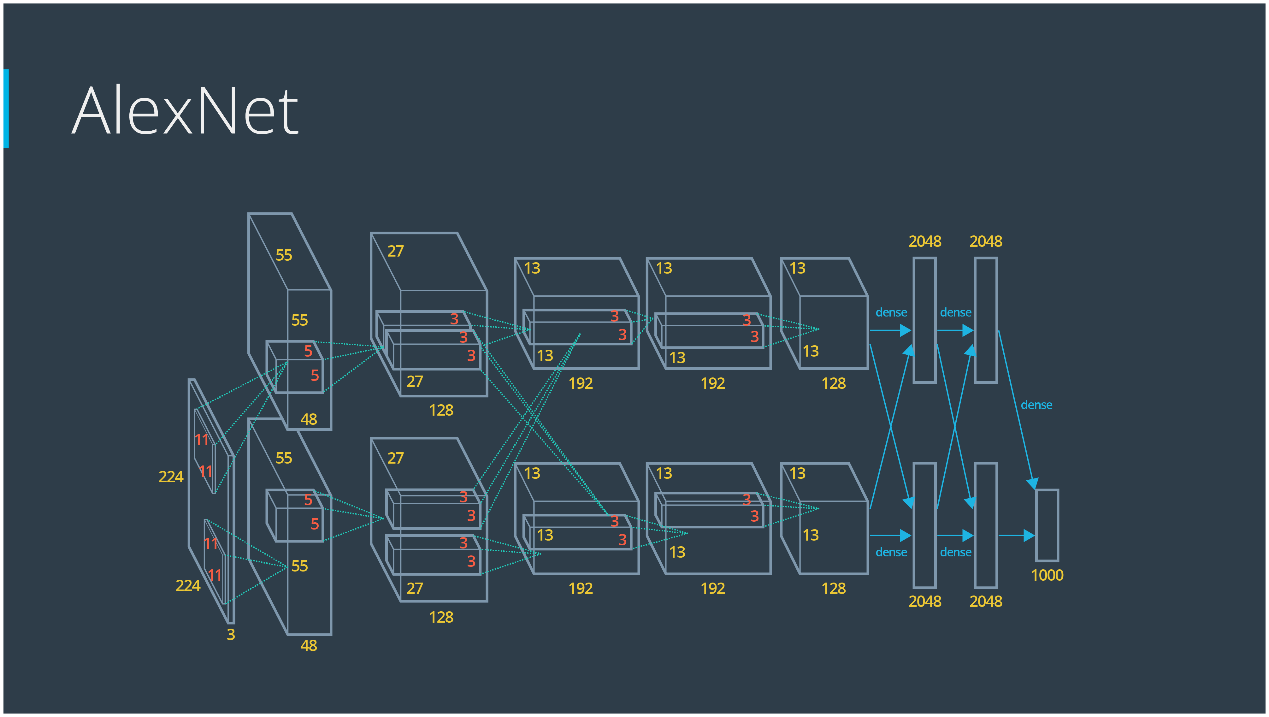
ImageNet was its own competition from 2012-2017, but now it's hosted on Kaggle! There are 1,000 different image categories between over 14 million images, so it's a great way to get involved with large datasets.

Pre-training a network with the ImageNet dataset is a very common way to get a strong neural network that can be used for transfer learning. With recent versions of Keras, you can easily import a pre-trained network by using the Keras Applications models. We'll come back to this soon.

5. AlexNet Architecture

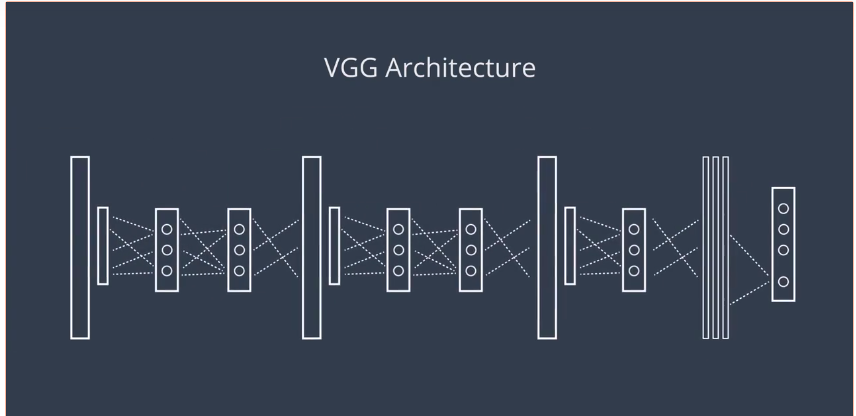
<https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>

AlexNet puts the network on two GPUs, which allows for building a larger network. Although most of the calculations are done in parallel, the GPUs communicate with each other in certain layers. The original research paper on AlexNet said 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.



6.vgg

<https://arxiv.org/pdf/1409.1556.pdf>



As we mentioned earlier, you can fairly quickly utilize a pre-trained model with [**Keras Applications**](https://keras.io/applications/). VGG16 is one of the built-in models supported. There are actually two versions of VGG, VGG16 and VGG19 (where the numbers denote the number of layers included in each respective model), and you can utilize either with Keras, but we'll work with VGG16 here.

Pre-processing for ImageNet weights

There is another item to consider before jumping into using an ImageNet pre-trained model.

These networks are typically pre-trained with a specific type of pre-processing, so you need to make sure to use the same pre-processing steps, or your network's outputs will likely be erratic.

VGG uses 224x224 images as input, so that's another thing to consider.

**from** keras.preprocessing **import** image

**from** keras.applications.vgg16 **import** preprocess\_input

**import** numpy **as** np

img\_path = 'your\_image.jpg'

img = image.load\_img(img\_path, target\_size=(224, 224))

x = image.img\_to\_array(img)

x = np.expand\_dims(x, axis=0)

x = preprocess\_input(x)