Transfer learning

INTRODUCTION

Transfer learning is one of the most important techniques of deep learning.

When building a vision system to solve a specific problem, you usually need to collect and label a huge amount of data to train your network. But what if we could use an existing neural network, that someone else has tuned and trained, and use it as a starting point for our new task? Transfer learning allows us to do just that.

We can download an open-source model that someone else has already trained and tuned for weeks and use their optimized parameters (weights) as a starting point to train our model just a little bit more on a smaller dataset that we have for a given task. This way we can train our network a lot faster and achieve very high results.

DEFINITION AND WHY TRANSFER LEARNING ?

- transfer learning means transferring what a neural network has learned from being trained on a specific dataset to another related problem.
- Problems transfer learning solve :

1) **Data problem:** it requires a lot of data to be able to get decent results which is not very feasible in most cases. It is relatively rare to have a dataset of sufficient size to solve your problem. It is also very expensive to acquire and label data which is mostly a manual process that has to be done by humans capturing images and labeling them one-by-one which makes it a non-trivial, very expensive task.

2) **Computation problem:** even if you are able to acquire hundreds of thousands of images for your problem, it is computationally very expensive to train a deep neural network on millions of images.

The training process of a deep neural network from scratch is very expensive because it usually requires weeks of training on multiple GPUs. Also Keep in mind that the neural network training process is an iterative process. So, even if you happen to have the computing power that is needed to train complex neural networks, having to spend a few weeks experimenting different hyperparameters in each training iteration will make the project very expensive until you finally reach satisfactory results.

Additionally, one very important benefit of using transfer learning is that it helps the model generalize its learnings and avoid overfitting.

CONT.

- to train an image classifier that will achieve near or above human level accuracy on image classification, we'll need massive amounts of data, large compute power, and lots of time on our hands.
- Knowing this would be a problem for people with little or no resources, researchers built state-of-the-art models that were trained on large image datasets like ImageNet, MS COCO, Open Images, etc. and decided to share their models to the general public for reuse.
- Even if that is the case, you might be better off using transfer learning to fine-tune the pretrained network on your large dataset.
- "In transfer learning, we first train a base network on a base dataset and task, and then we repurpose the learned features, or transfer them to a second target network to be trained on a target dataset and task. This process will tend to work if the features are general, meaning suitable to both base and target tasks, instead of specific to the base task."
-

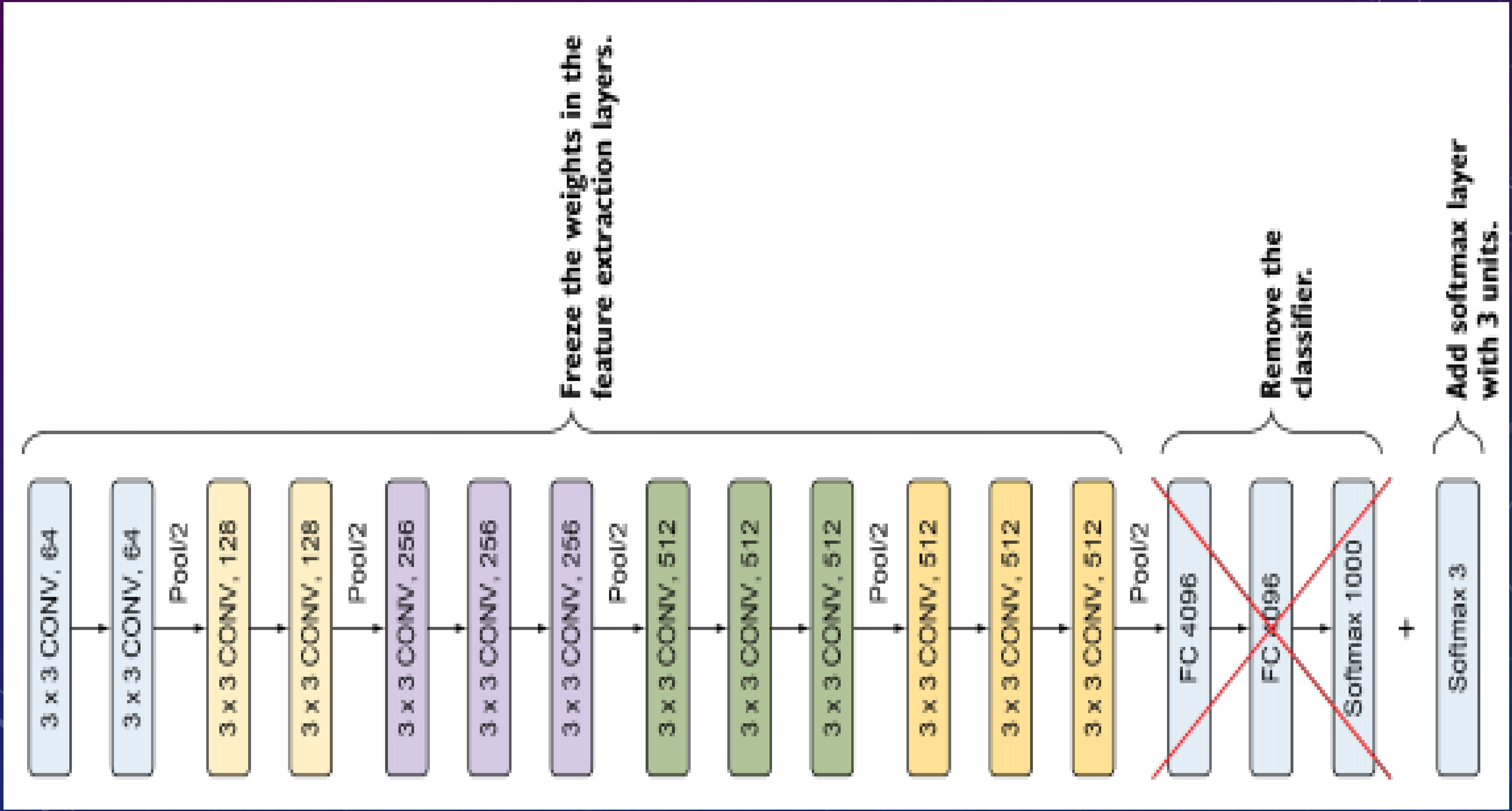
EXAMPLE

- First, we need to find a dataset that has similar features to our problem at hand. This involves spending sometime exploring different open-source datasets to find the closest one to our problem.
- Next, we need to choose a network that has been trained on ImageNet (Example of datasets) and achieved good results.

For example VGG16

- To adapt the VGG16 network to our problem, we are going to download the VGG16 network with the pretrained weights and remove the classifier part then add our own classifier.
- Then retrain the new network. This is called **using a pretrained network as a feature extractor**.
- A **pretrained model** is a network that has been previously trained on a large dataset, typically on a large-scale image classification task. We can either 1) directly use the pretrained model as it to run our predictions, or 2) use the pretrained feature extraction part of the network then add our own classifier. The classifier here could be one or more dense layers or even traditional machine learning algorithms like Support Vector Machines (SVM).
-

EXAMPLE CONT.



HOW TRANSFER LEARNING WORKS

1. What is really being learned by the network during training? The short answer is “Feature maps”.

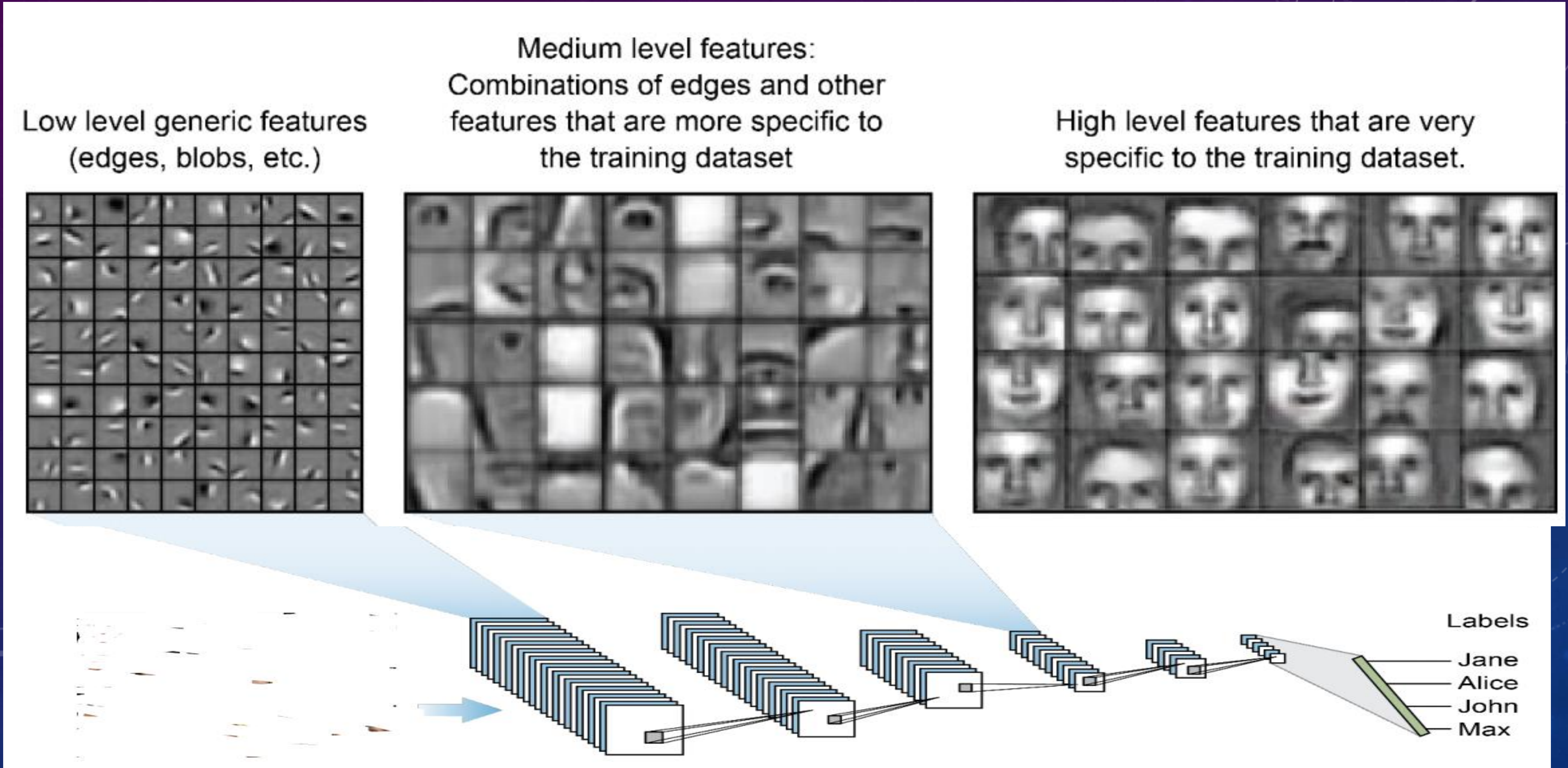
2. How are these features learned? During the backpropagation process, the weights are updated until we get to the “optimized weights” that minimize the error function.

3. What is the relationship between features and weights? A feature map is the result of passing the weights filter on the input image during the convolution process

4. So, what is really being transferred from one network to another? To transfer features, we download the **optimized weights** of the pretrained network. These weights are then re-used as the starting point for the training process and retrained to adapt to the new problem.

When training is complete, we output two main items: 1) the network architecture, and 2) the trained weights.

HOW DO NEURAL NETWORKS LEARN FEATURES?

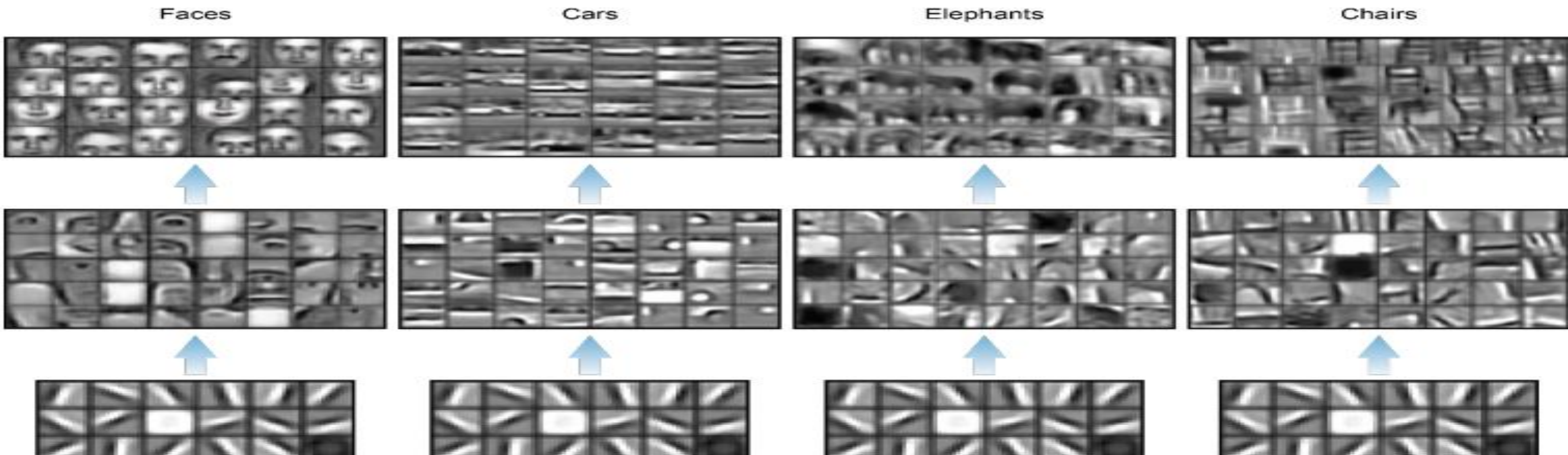

Low level generic features
(edges, blobs, etc.)
Medium level features:
Combinations of edges and other features that are more specific to the training dataset
High level features that are very specific to the training dataset.
Labels
Jane
Alice
John
Max

CONT.

- he neural network learns the features in your dataset step-by-step in an increasing level of complexity layer after the other. These are called feature maps.
- The deeper you go through the network layers, the more image specific features are learned.
- The first layer detects low level features such as edges and curves. The output of the first layer becomes input to the second layer which produces higher level features, like semi-circles and squares.
- The next layer assembles the output of the previous layer to parts of familiar objects, and a subsequent layer detects the objects.
- As we go through more layers, the network yields an activation map that represents more and more complex features.
- he deeper you go into the network, the filters begin to be more responsive to a larger region of the pixel space.
- Higher level layers amplify aspects of the received inputs that are important for discrimination and suppress irrelevant variations.

CONT.

- the earlier layer’s features are very similar for all models.
- The lower level features are almost always transferable from one task to another because they contain generic information like the structure and the nature of how images look.
- Transferring information like lines, dots, curves, and small parts of the objects is very valuable for the network to learn faster and with less data on the new task.
- he deeper we go in to the network, we notice that the features start to be more specific until the network overfits its training data and it becomes harder to generalize to different tasks



WHAT ABOUT THE TRANSFERABILITY OF FEATURES EXTRACTED AT LATER LAYERS IN THE NETWORK?

- The transferability of features that are extracted at later layers depends on the similarity of the original and new datasets.
- The idea here is that all images must have shapes and edges so the early layers are usually transferable between different domains.

Based on the similarity of the source and target domains, we can decide whether to transfer only the low level features from the source domain or all the high level features or somewhere in between.

- **Source Domain**: the original dataset that the pretrained network is trained on.
  Target Domain: the new dataset that we want to train the network on.

TRANSFER LEARNING APPROACHES

- **There are three major transfer learning approaches as follows:**
  1. Pretrained network as a classifier
  2. Pretrained network as feature extractor
  3. Fine tuning

- Each approach can be effective and save significant time in developing and training a deep convolutional neural network model.

- We should the appropriate approach for each application.

1)PRETRAINED NETWORK AS A CLASSIFIER

- The pre-trained model is used directly to classify new images with no changes applied to it and no extra training.
- All we do here is download the network architecture and its pretrained weights. Then run the predictions directly on our new data.
- In this case, we are saying that the domain of our new problem is very similar to the one that the pretrained network was
  trained on and it is ready to just be “deployed”. So no training is done here.
- Datasets contain the object which will be detected .
- Using a pretrained network as a classifier doesn’t really involve any layers freezing or extra model training. Instead, it is just taking a network that was trained on your similar problem and deploying it directly to your task.

2)PRETRAINED NETWORK AS A FEATURE EXTRACTOR

- We take a pretrained CNN on ImageNet(ex of datasets), freeze its feature extraction part, remove the classifier part, and add our own new dense classifier layers.
- We usually go with this scenario when our new task is similar to the original dataset that the pretrained network was trained on.
- This means that we can utilize the high level features that were extracted from the ImageNet dataset in to this new task.
- To do that, we will freeze all the layers from the pretrained network and only train the classifier part that we just added on the new dataset.
- This approach is called “using a pretrained network as a feature extractor” because we froze the feature extractor part to transfer all the learned feature maps to our new problem.
- We only added a new classifier, which will be trained from scratch, on top of the pretrained model so that we can repurpose the feature maps learned previously for our dataset.
- We only added a new classifier, which will be trained from scratch, on top of the pretrained model so that we can repurpose
the feature maps learned previously for our dataset.
- The reason we remove the classification part of the pretrained network is that it is often very specific to the original classification task, and subsequently specific to the set of classes on which the model was trained.

3)FINE-TUNING

- So far, we've seen two basic approaches of using a pretrained network in transfer learning: 1)pretrained network as a classifier, and 2) pretrained network as a feature extractor. We usually use these two approaches when the target domain is somewhat similar to the source domain.
- Transfer learning works great even when the domains are very different. We just need to extract the **correct** feature maps from the source domain and "fine tune" it to fit the target domain.
- Fine tuning is when you decide to freeze part of the feature extraction part of the network not all of it.
- we can decide to freeze the network at the appropriate level of feature maps:

1. If the domains are similar, we might want to freeze all the network up to the last feature map level
2. If the domains are very different, we might decide to freeze the pretrained network after feature maps 1 and retrain all the remaining layers

Between these two options a range of fine tuning options that we can apply.
We typically decide the appropriate level of fine tuning by trial and error. But there are guidelines that we can follow to intuitively decide on the fine tuning level of the pretrained network. The decision is a function of two factors:

1) the amount of data that we have

2) the level of similarity between the source and target domains.

WHAT IS FINE TUNING?

- The formal definition of fine-tuning is freezing a few of the network layers that are used for feature extraction, and jointly training both the non-frozen layers and the newly added classifier layers of the pretrained model. It is called fine tuning because when we retrain the feature extraction layers, we "fine tune" the higher order feature representations to make them more relevant for the new task dataset.

WHY FINE TUNING IS BETTER THAN TRAINING FROM SCRATCH?

- When we train a network from scratch, we usually randomly initialize the weights and apply gradient descent optimizer to find the best set of weights that optimizes our error function
- Since these weights start with random values, there is no guarantee that they will start with values that are close to the desired optimal values. And if the initialized value is far away from the optimal value, the optimizer will take a long time to converge.
- This is when fine tuning can be very useful. The pretrained network weights have been already optimized to learn from its dataset. This means that when we use this network in our problem, we start with the weights values that it ended with. This makes the network much faster to converge than having to randomly initialize the weights. This is what the term “fine-tuning” refers to.
- We are basically fine-tuning the already-optimized weights to fit our new problem instead of training the entire network from scratch with random weights.
- starting with the trained weights will converge faster than having to train the network from scratch with randomly initialized weights.
- *USE A SMALLER LEARNING RATE WHEN FINE TUNING*

CHOOSE THE APPROPRIATE LEVEL OF TRANSFER LEARNING

- **Choosing the appropriate level of using transfer learning is a function of two important factors:**
  1) **The size of the target dataset (small or large) :** When we have a small dataset, there is probably not much information that the network would learn from training more layers, so it will tend to overfit the new data. In this case we probably want to do less fine tuning and rely more on the source dataset.

  2) **Domain similarity of the source and target datasets:** how similar is your new problem to the domain of the original dataset . For example, if your problem is to classify cars and boats, ImageNet could be a good option because it contains a

  lot of images of similar features. On the other hand, if your problem is to classify chest cancer on x-ray images, this is a completely different domain that will likely require a
lot of fine-tuning.

These two factors develop the four major scenarios below:
1. Target dataset is small and similar to the source dataset
2. Target dataset is large and similar to the source dataset
3. Target dataset is small and very different from the source dataset
4. Target dataset is large and very different from the source dataset

SCENARIO #1: TARGET DATASET IS SMALL AND SIMILAR TO SOURCE DATASET

- Since the original dataset is similar to our new dataset, we can expect that the higher-level features in the pretrained ConvNet to be relevant to our dataset as well. Then it might be best to freeze the feature extraction part of the network and only retrain the classifier.
- If you have a small amount of data, be careful of overfitting when you fine tune your pretrained network.

SCENARIO #2: TARGET DATASET IS LARGE AND SIMILAR TO THE SOURCE DATASET

- Since both domains are similar, we can freeze the feature extraction part and retrain the classifier similar to what we did in scenario #1. But, since we have more data in the new domain, we can get a performance boost from fine tuning through all or part of the pretrained network with more confidence that we won’t overfit
- Fine tuning through the entire network is
  not really needed because the higher-level features are related (since the datasets are similar). So a good start is to freeze approximately 60% - 80% of the pretrained network and retrain the rest on the new data.

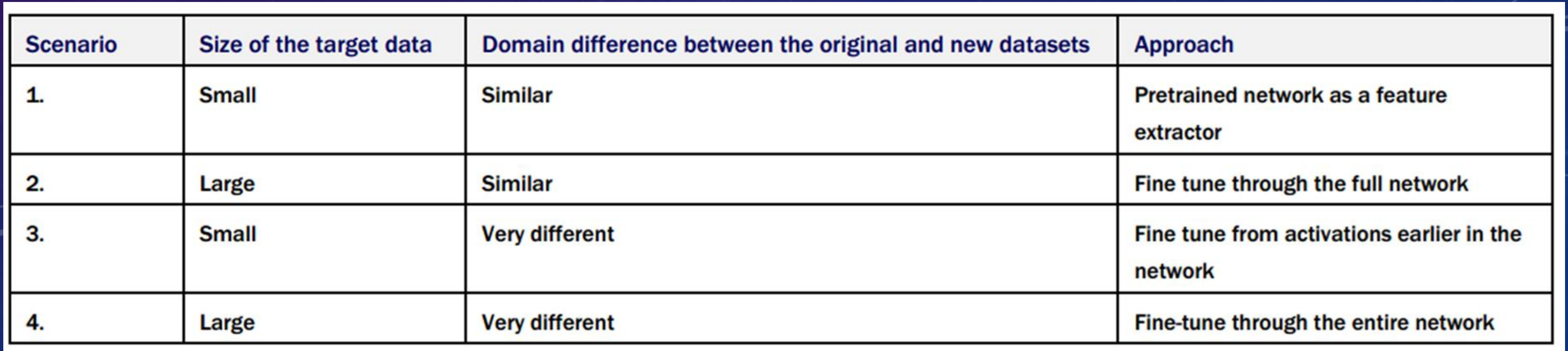
SCENARIO #3: TARGET DATASET IS SMALL AND DIFFERENT FROM THE SOURCE DATASET

- Since the dataset is different, it might not be best to freeze the higher-level features of the pretrained network because they contain more dataset-specific features. Instead, it would work better to retrain layers from somewhere earlier in the network.
- Or even don’t freeze any layers fine tune the entire network. However, since you have a small dataset, fine tuning the entire network on your small dataset might not be a good idea because it makes it prone to overfitting.
- A mid-way solution would work better in this case. So a good start is to freeze approximately the first third or half of the pretrained network. Afterall, the early layers contain very generic feature maps that would be useful for your dataset even if it is very different.

SCENARIO #4: TARGET DATASET IS LARGE AND DIFFERENT FROM THE SOURCE DATASET

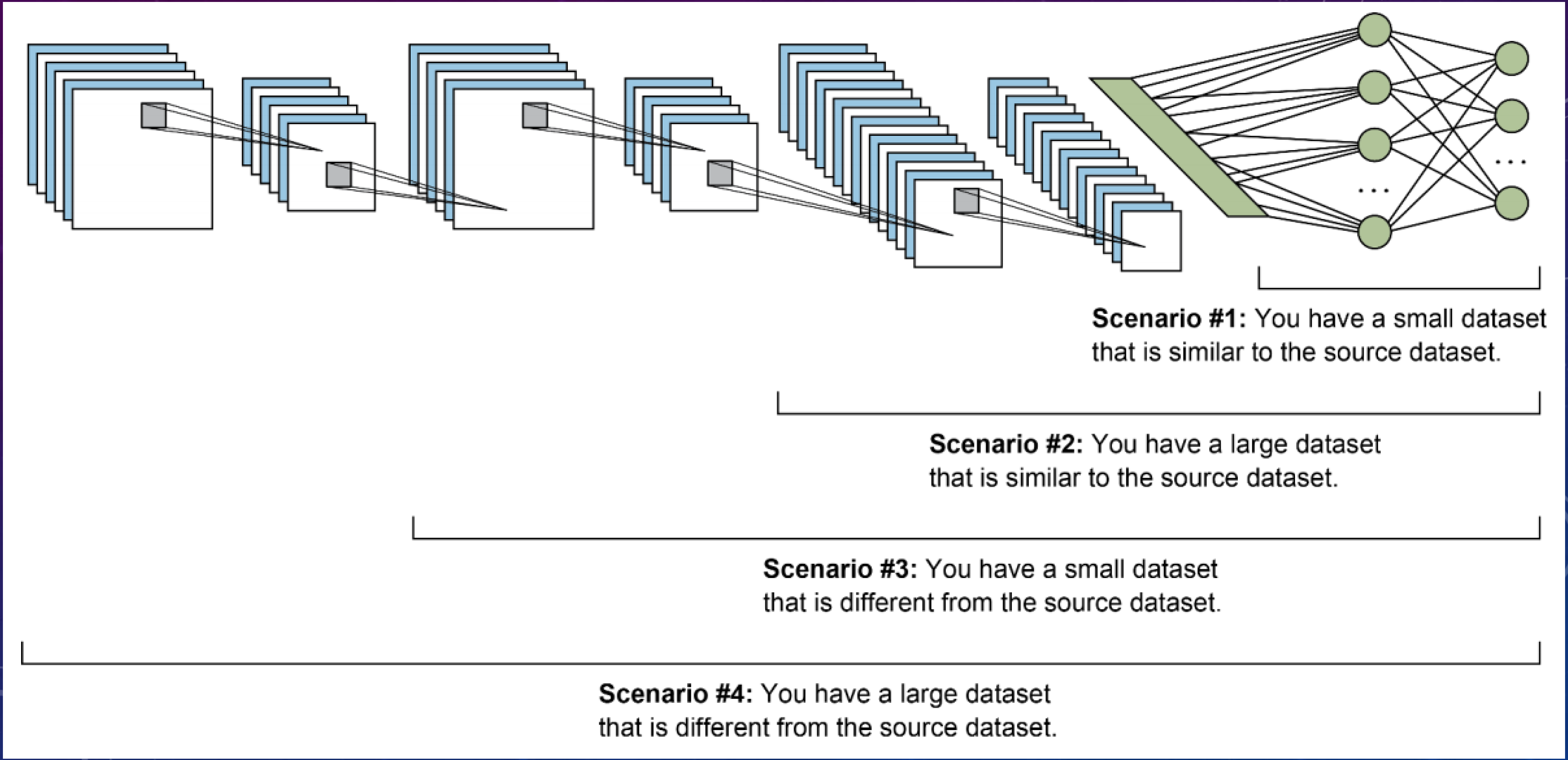
- Since the new dataset is large, you might be tempted just train the entire network from scratch and not use transfer learning at all.
- However, in practice it is often still very beneficial to initialize with weights from a pretrained model
- It makes the model converges faster.
- In this case, we have a large dataset that provides us confidence to fine tune

through the entire network without having to worry about overfitting.



Scenario	Size of the target data	Domain difference between the original and new datasets	Approach
1.	Small	Similar	Pretrained network as a feature extractor
2.	Large	Similar	Fine tune through the full network
3.	Small	Very different	Fine tune from activations earlier in the network
4.	Large	Very different	Fine-tune through the entire network

SUMMARY



Scenario #1: You have a small dataset that is similar to the source dataset.

Scenario #2: You have a large dataset that is similar to the source dataset.

Scenario #3: You have a small dataset that is different from the source dataset.

Scenario #4: You have a large dataset that is different from the source dataset.