Transfer Learning for Computer Vision

Topics Covered

1. Understanding The Concept of Transfer Learning
2. Transfer Learning Approach
3. Transfer Learning for Image Recognition
4. Models For Transfer Learning
5. Summary

Understanding the concept of Transfer Learning

Transfer learning generally refers to a process where a model trained on one problem is used in some way on a second related problem.

> This is typically understood in a supervised learning context, where the input is the same but the target may be of a different nature. For example, we may learn about one set of visual categories, such as cats and dogs, in the first setting, then learn about a different set of visual categories, such as ants and wasps, in the second setting.

In deep learning, transfer learning is a technique whereby a neural network model is first trained on a problem similar to the problem that is being solved. One or more layers from the trained model are then used in a new model trained on the problem of interest.

Transfer learning has the benefit of decreasing the training time for a neural network model and can result in lower generalization error.

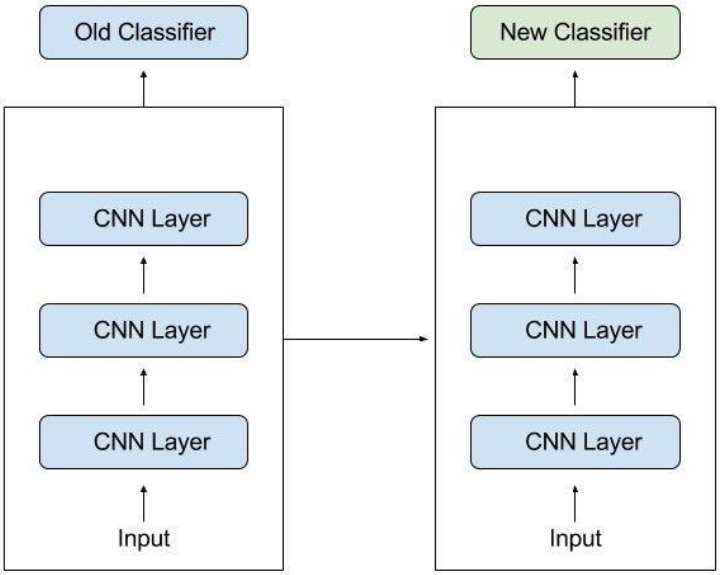
The weights in re-used layers may be used as the starting point for the training process and adapted in response to the new problem. This usage treats transfer learning as a type of weight initialization scheme. This may be useful when the first related problem has a lot more labelled data than the problem of interest and the similarity in the structure of the problem may be useful in both contexts.

> The objective is to take advantage of data from the first setting to extract information that may be useful when learning or even when directly making predictions in the second setting.

How Transfer Learning Works

In computer vision, for example, neural networks usually try to detect edges in the earlier layers, shapes in the middle layer and some task-specific features in the later layers. In transfer learning, the early and middle layers are used and we only retrain the latter layers. It helps leverage the labelled data of the task it was initially trained on.

Let's go back to the example of a model trained for recognizing a backpack on an image, which will be used to identify sunglasses. In the earlier layers, the model has learned to recognize objects, because of that we will only retrain the latter layers so it will learn what separates sunglasses from other objects.



In transfer learning, we try to transfer as much knowledge as possible from the previous task the model was trained on to the new task at hand. This knowledge can be in various forms depending on the problem and the data. For example, it could be how models are composed, which allows us to more easily identify novel objects.

Transfer Learning Approach

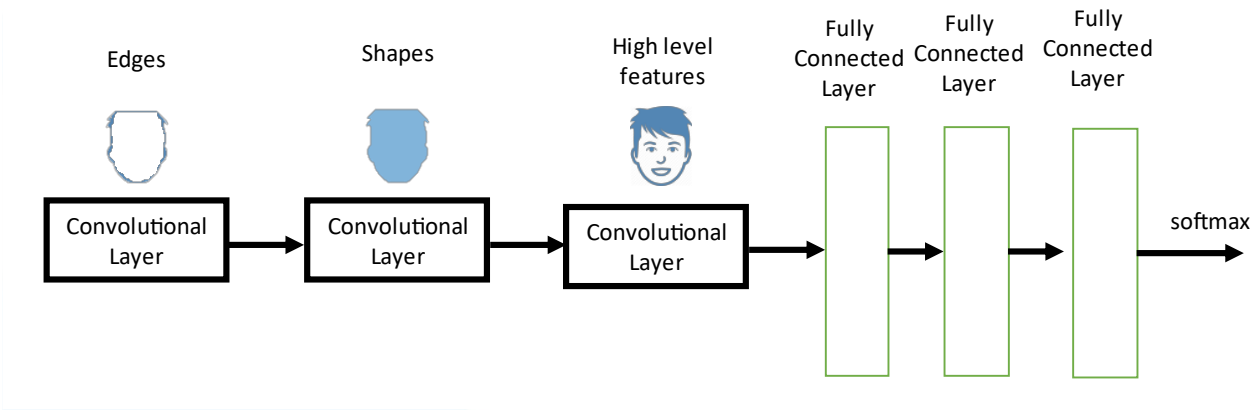
What is our objective when we train a neural network? We wish to identify the correct weights for the network by multiple forward and backward iterations. By using pre-trained models which have been previously trained on large datasets, we can directly use the weights and architecture obtained and apply the learning on our problem statement. This is known as transfer learning. We “transfer the learning” of the pre-trained model to our specific problem statement.

You should be very careful while choosing what pre-trained model you should use in your case. If the problem statement we have at hand is very different from the one on which the pre-trained model was trained – the prediction we would get would be very inaccurate. For example, a model previously trained for speech recognition would work horribly if we try to use it to identify objects using it.

We are lucky that many pre-trained architectures are directly available for us in the Keras library. **Imagenet** data set has been widely used to build various architectures since it is large enough (1.2M images) to create a generalized model. The problem statement is to train a model that can correctly classify the images into 1,000 separate object categories. These 1,000 image categories represent object classes that we come across in our day-to-day lives, such as species of dogs, cats, various household objects, vehicle types etc.

These pre-trained networks demonstrate a strong ability to generalize to images outside the ImageNet dataset via transfer learning. We make modifications in the pre-existing model by fine-tuning the model. Since we assume that the pre-trained network has been trained quite well, we would not want to modify the weights too soon and too much. While modifying we generally use a learning rate smaller than the one used for initially training the model.

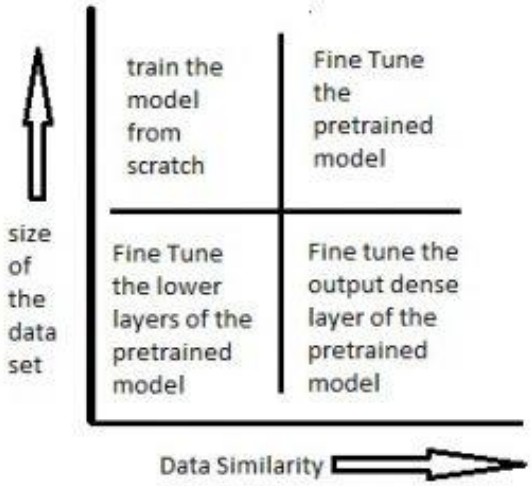
Pretrained Convolutional Neural Network



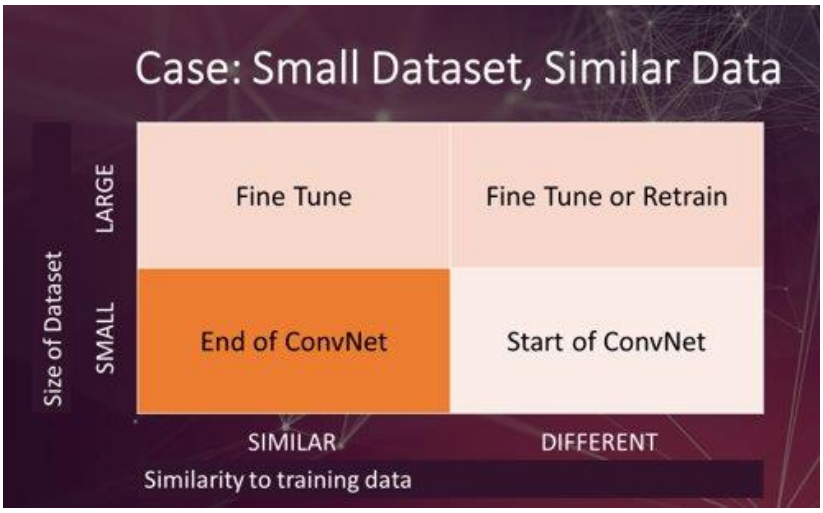
Ways to Fine tune the model

1. **Feature extraction** – We can use a pre-trained model as a feature extraction mechanism. What we can do is that we can remove the output layer(the one which gives the probabilities for being in each of the 1000 classes) and then use the entire network as a fixed feature extractor for the new data set.
2. **Use the Architecture of the pre-trained model –** What we can do is that we use architecture of the model while we initialize all the weights randomly and train the model according to our dataset again.
3. **Train some layers while freeze others** – Another way to use a pre-trained model is to train is partially. What we can do is we keep the weights of initial layers of the model frozen while we retrain only the higher layers. We can try and test as to how many layers to be frozen and how many to be trained.

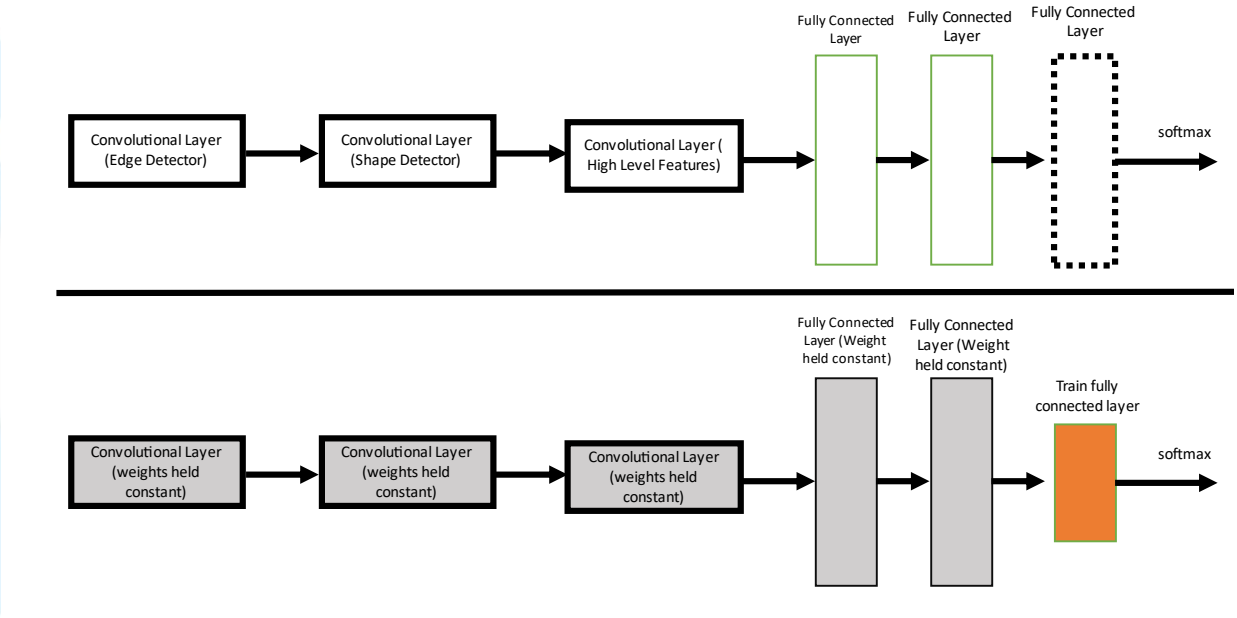
The below diagram should help you decide on how to proceed on using the pre trained model in your case


train the model from scratch
Fine Tune the pretrained model
Fine Tune the lower layers of the pretrained model
Fine tune the output dense layer of the pretrained model
size of the data set
Data Similarity

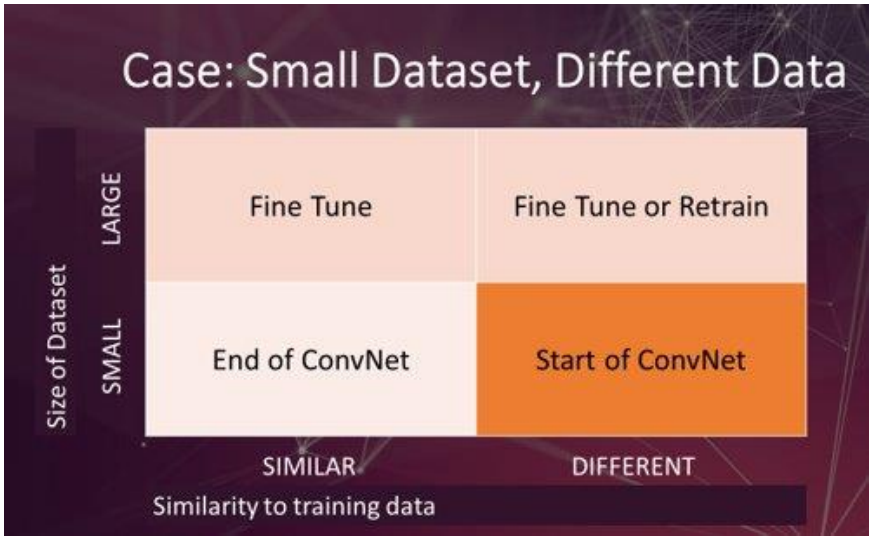
Scenario 1 – Size of the Data set is small while the Data similarity is very high



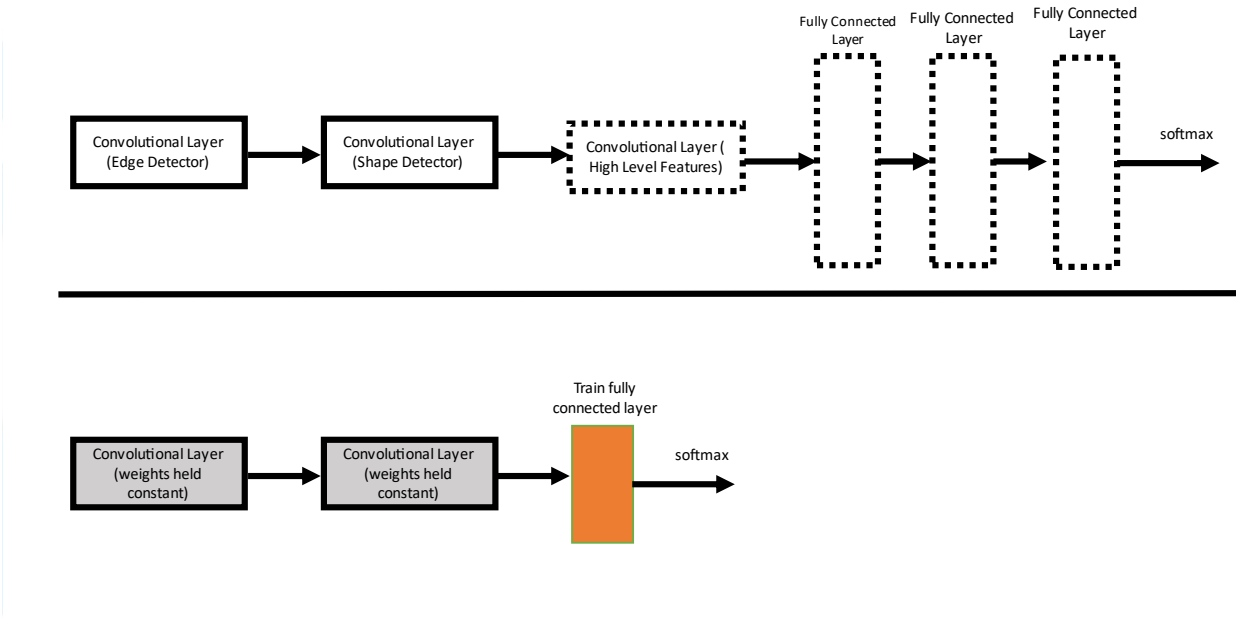
In this case, since the data similarity is very high, we do not need to retrain the model. All we need to do is to customize and modify the output layers according to our problem statement. We use the pretrained model as a feature extractor. Suppose we decide to use models trained on Imagenet to identify if the new set of images have cats or dogs. Here the images we need to identify would be similar to imagenet, however we just need two categories as my output – cats or dogs. In this case all we do is just modify the dense layers and the final softmax layer to output 2 categories instead of a 1000.



Scenario 2 – Size of the data is small as well as data similarity is very low

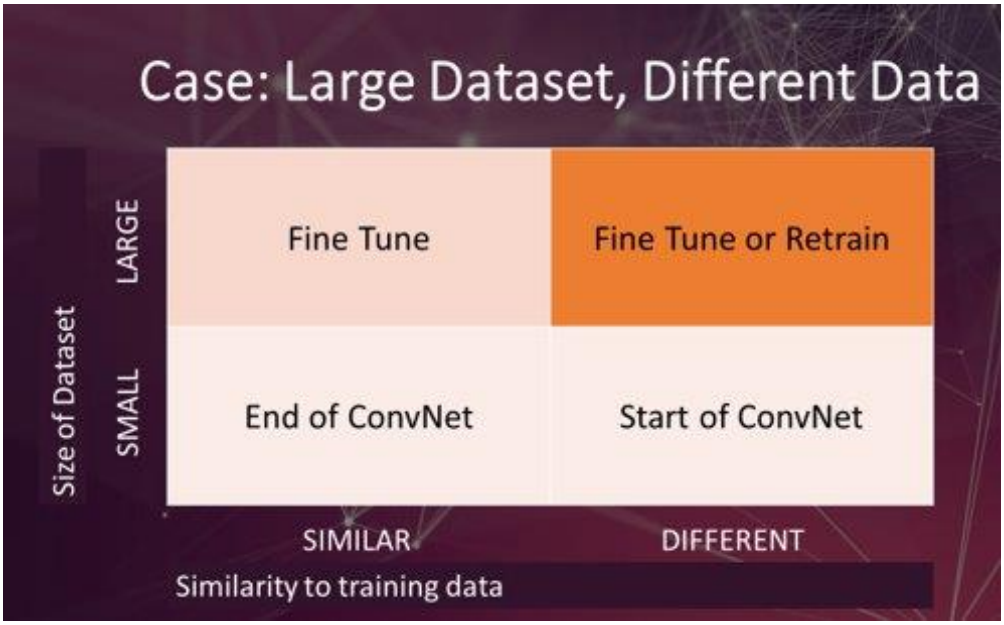


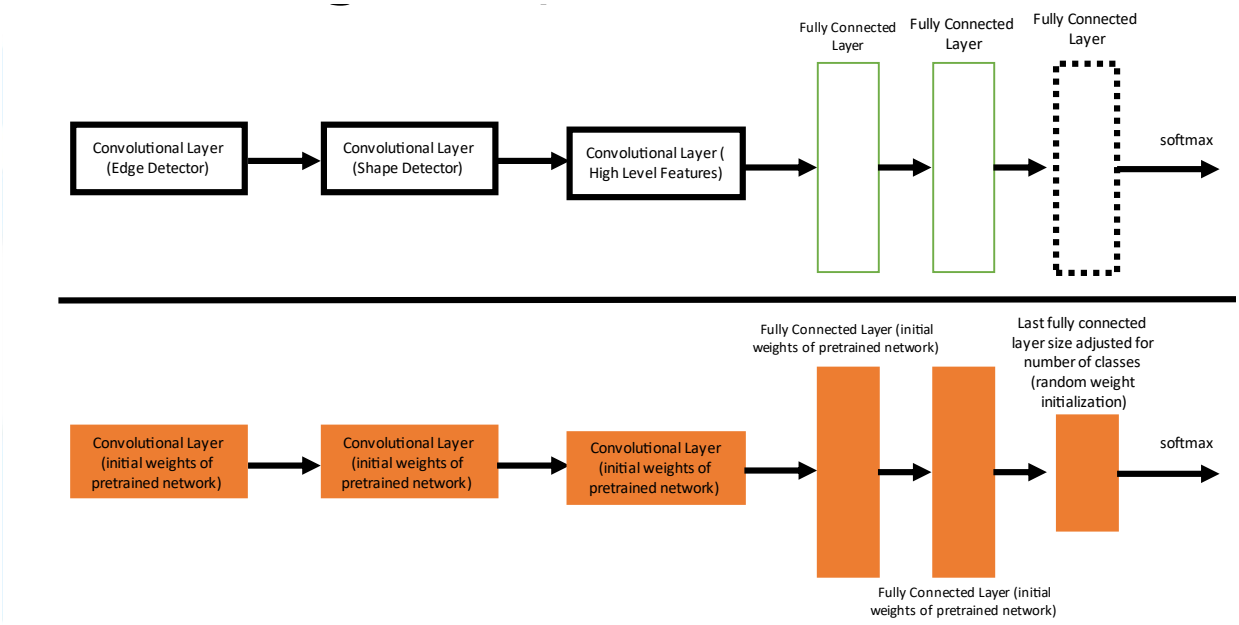
In this case we can freeze the initial (let's say k) layers of the pretrained model and train just the remaining(n-k) layers again. The top layers would then be customized to the new data set. Since the new data set has low similarity it is significant to retrain and customize the higher layers according to the new dataset. The small size of the data set is compensated by the fact that the initial layers are kept pretrained(which have been trained on a large dataset previously) and the weights for those layers are frozen.



Scenario 3 – Size of the data set is large however the Data similarity is very low

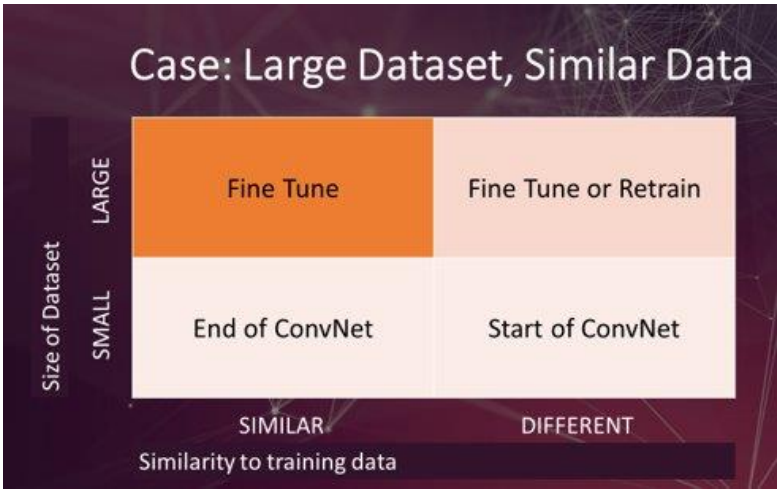
In this case, since we have a large dataset, our neural network training would be effective. However, since the data we have is very different as compared to the data used for training our pretrained models. The predictions made using pretrained models would not be effective. Hence, its best to train the neural network from scratch according to your data.

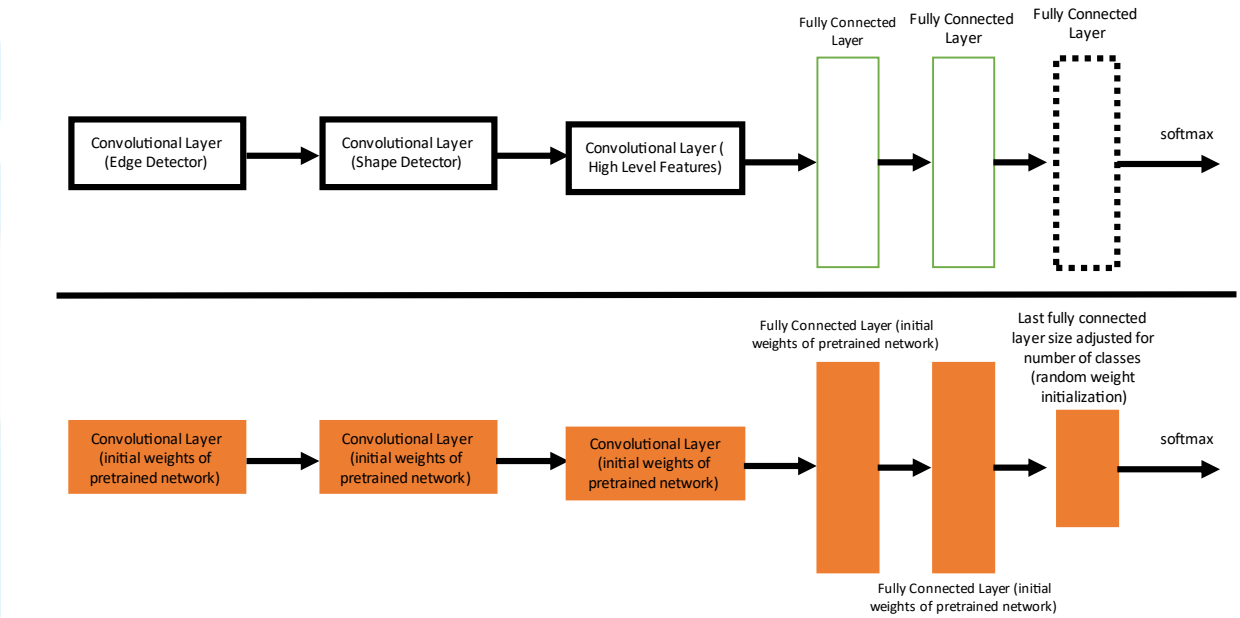




Scenario 4 – Size of the data is large as well as there is high data similarity

This is the ideal situation. In this case the pretrained model should be most effective. The best way to use the model is to retain the architecture of the model and the initial weights of the model. Then we can retrain this model using the weights as initialized in the pre-trained model.





Transfer Learning for Image Recognition

A range of high-performing models have been developed for image classification and demonstrated on the annual ImageNet Large Scale Visual Recognition Challenge, or ILSVRC.

This challenge, often referred to simply as ImageNet, given the source of the image used in the competition, has resulted in a number of innovations in the architecture and training of convolutional neural networks. In addition, many of the models used in the competitions have been released under a permissive license.

These models can be used as the basis for transfer learning in computer vision applications.

This is desirable for a number of reasons, not least:

- **Useful Learned Features:** The models have learned how to detect generic features from photographs, given that they were trained on more than 1,000,000 images for 1,000 categories.
- **State-of-the-Art Performance:** The models achieved state of the art performance and remain effective on the specific image recognition task for which they were developed.
- **Easily Accessible:** The model weights are provided as free downloadable files and many libraries provide convenient APIs to download and use the models directly.

The model weights can be downloaded and used in the same model architecture using a range of different deep learning libraries, including Keras.

How to Use Pre-Trained Models

The use of a pre-trained model is limited only by your creativity.

For example, a model may be downloaded and used as-is, such as embedded into an application and used to classify new photographs.

Alternately, models may be downloaded and use as feature extraction models. Here, the output of the model from a layer prior to the output layer of the model is used as input to a new classifier model.

Recall that convolutional layers closer to the input layer of the model learn low-level features such as lines, that layers in the middle of the layer learn complex abstract features that combine the lower level features extracted from the input, and layers closer to the output interpret the extracted features in the context of a classification task.

Armed with this understanding, a level of detail for feature extraction from an existing pre-trained model can be chosen. For example, if a new task is quite different from classifying objects in photographs (e.g. different to ImageNet), then perhaps the output of the pre-trained model after the few layers would be appropriate. If a new task is quite similar to the task of classifying objects in photographs, then perhaps the output from layers much deeper in the model can be used, or even the output of the fully connected layer prior to the output layer can be used.

The pre-trained model can be used as a separate feature extraction program, in which case input can be pre-processed by the model or portion of the model to a given an output (e.g. vector of numbers) for each input image, that can then use as input when training a new model.

Alternately, the pre-trained model or desired portion of the model can be integrated directly into a new neural network model. In this usage, the weights of the pre-trained can be frozen so that they are not updated as the new model is trained. Alternately, the weights may be updated during the training of the new model, perhaps with a lower learning rate, allowing the pre-trained model to act like a weight initialization scheme when training the new model.

We can summarize some of these usage patterns as follows:

- **Classifier:** The pre-trained model is used directly to classify new images.
- **Standalone Feature Extractor:** The pre-trained model, or some portion of the model, is used to pre-process images and extract relevant features.
- **Integrated Feature Extractor:** The pre-trained model, or some portion of the model, is integrated into a new model, but layers of the pre-trained model are frozen during training.
- **Weight Initialization:** The pre-trained model, or some portion of the model, is integrated into a new model, and the layers of the pre-trained model are trained in concert with the new model.

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

It may not be clear as to which usage of the pre-trained model may yield the best results on your new computer vision task, therefore some experimentation may be required.

Models for Transfer Learning

There are perhaps a dozen or more top-performing models for image recognition that can be downloaded and used as the basis for image recognition and related computer vision tasks.

Perhaps three of the more popular models are as follows:

- VGG (e.g. VGG16 or VGG19).
- GoogLeNet (e.g. InceptionV3).
- Residual Network (e.g. ResNet50).
- Alexnet

These models are both widely used for transfer learning both because of their performance, but also because they were examples that introduced specific architectural innovations, namely consistent and repeating structures (VGG), inception modules (GoogLeNet), and residual modules (ResNet).

Keras provides access to a number of top-performing pre-trained models that were developed for image recognition tasks.

They are available via the Applications API, and include functions to load a model with or without the pre-trained weights, and prepare data in a way that a given model may expect (e.g. scaling of size and pixel values).

The first time a pre-trained model is loaded, Keras will download the required model weights, which may take some time given the speed of your internet connection. Weights are stored in the. keras*/models/* directory under your home directory and will be loaded from this location the next time that they are used.

When loading a given model, the "*include_top*" argument can be set to *False*, in which case the fully-connected output layers of the model used to make predictions is not loaded, allowing a new output layer to be added and trained.

Overview of ImageNet data set

ImageNet is an ongoing research effort to provide researchers around the world with image data for training large-scale object recognition models.

ImageNet is an image dataset organized according to the WordNet hierarchy. Each meaningful concept in WordNet, possibly described by multiple words or word phrases, is called a "synonym set" or "synset". There are more than 100,000 synsets in WordNet; the majority of them are nouns (80,000+). In ImageNet,they aim to provide on average 1000 images to illustrate each synset. Images of each concept are quality-controlled and human-annotated. In its completion, we hope ImageNet will offer tens of millions of cleanly labelled and sorted images for most of the concepts in the WordNet hierarchy.

Based on statistics about the dataset recorded on the ImageNet homepage, there are a little more than 14 million images in the dataset, a little more than 21 thousand groups or classes (synsets), and a little more than 1 million images that have bounding box annotations (e.g. boxes around identified objects in the images). The photographs were annotated by humans using crowdsourcing platforms such as Amazon's Mechanical Turk.

The project to develop and maintain the dataset was organized and executed by a collocation between academics at Princeton, Stanford, and other American universities.

The project does not own the photographs that make up the images; instead, they are owned by the copyright holders. As such, the dataset is not distributed directly; URLs are provided to the images included in the dataset.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

The ImageNet Large Scale Visual Recognition Challenge or ILSVRC for short is an annual competition helped between 2010 and 2017 in which challenge tasks use subsets of the ImageNet dataset.
The goal of the challenge was to both promote the development of better computer vision techniques and to benchmark the state of the art.

The annual challenge focuses on multiple tasks for "*image classification*" that includes both assigning a class label to an image based on the main object in the photograph and "*object detection*" that involves localizing objects within the photograph.

> LSVRC annotations fall into one of two categories: (1) image-level annotation of a binary label for the presence or absence of an object class in the image, […] and (2) object-level annotation of a tight bounding box and class label around an object instance in the image

The general challenge tasks for most years are as follows:

- **Image classification**: Predict the classes of objects present in an image.
- **Single-object localization**: Image classification + draw a bounding box around one example of each object present.
- **Object detection**: Image classification + draw a bounding box around each object present.

More recently, and given the great success in the development of techniques for still photographs, the challenge tasks are changing to more difficult tasks such as labeling videos.

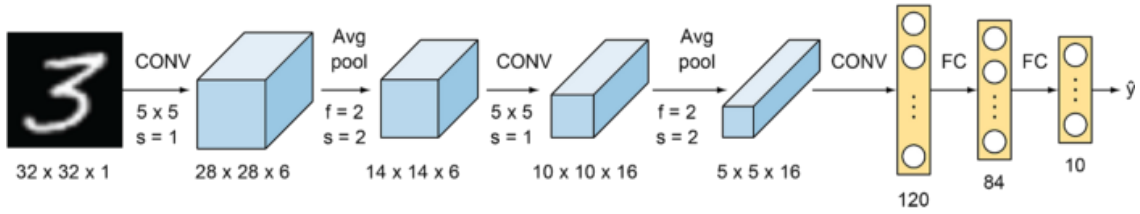
The datasets comprised approximately 1 million images and 1,000 object classes. The datasets used in challenge tasks are sometimes varied (depending on the task) and were released publicly to promote widespread participation from academia and industry.

For each annual challenge, an annotated training dataset was released, along with an unannotated test dataset for which annotations had to be made and submitted to a server for evaluation. Typically, the training dataset was comprised of 1 million images, with 50,000 for a validation dataset and 150,000 for a test set.

The state-of-the-art pre-trained networks included in the Keras core library represent some of the highest performing Convolutional Neural Networks on the ImageNet challenge over the past few years. These networks also demonstrate a strong ability to *generalize* to images outside the ImageNet dataset via *transfer learning*, such as feature extraction and fine-tuning.

LeNet-5

This is also known as the Classic Neural Network that was designed by Yann LeCun, Leon Bottou, Yosuha Bengio and Patrick Haffner for handwritten and machine-printed character recognition in 1990's which they called LeNet-5. The architecture was designed to identify handwritten digits in the MNIST data-set. The architecture is pretty straightforward and simple to understand. The input images were gray scale with dimension of 32*32*1 followed by two pairs of Convolution layer with stride 2 and Average pooling layer with stride 1. Finally, fully connected layers with Softmax activation in the output layer. Traditionally, this network had 60,000 parameters in total.



Lenet-5 Architecture

VGG16 and VGG19

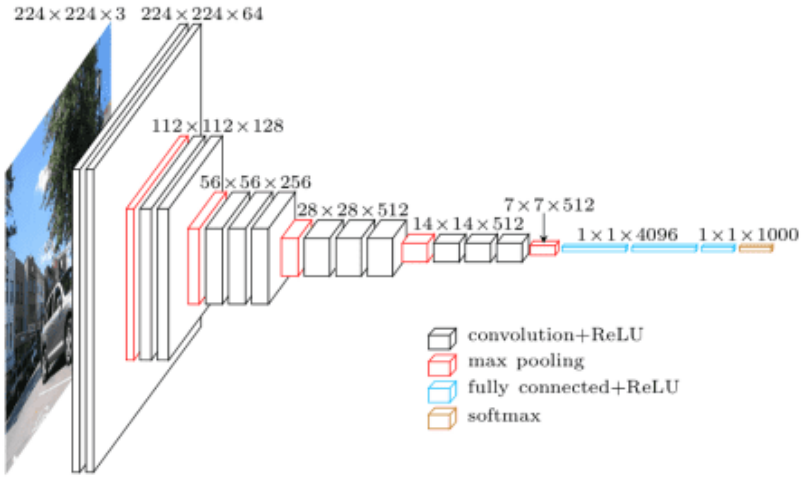


Figure 1: A visualization of the VGG architecture

The VGG network architecture was introduced by Simonyan and Zisserman in their 2014 paper, ***Very Deep Convolutional Networks for Large Scale Image Recognition***.

This network is characterized by its simplicity, using only *3×3* convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. Two fully-connected layers, each with 4,096 nodes are then followed by a softmax classifier (above).

The “16” and “19” stand for the number of weight layers in the network (columns D and E in **Figure 2** below):

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2: Table 1 of ***Very Deep Convolutional Networks for Large Scale Image Recognition***, Simonyan and Zisserman (2014).

In 2014-, 16- and 19-layer networks were considered *very* deep (although we now have the ResNet architecture which can be successfully trained at depths of 50-200 for ImageNet and over 1,000 for CIFAR-10).

Simonyan and Zisserman found training VGG16 and VGG19 challenging (specifically regarding convergence on the deeper networks), so in order to make training easier, they first trained *smaller* versions of VGG with less weight layers (columns A and C) first.

The smaller networks converged and were then used as *initializations* for the larger, deeper networks — this process is called **pre-training**.

While making logical sense, pre-training is a very time consuming, tedious task, requiring an *entire network* to be trained **before** it can serve as an initialization for a deeper network.

We no longer use pre-training (in most cases) and instead prefer Xaiver/Glorot initialization or MSRA initialization (sometimes called He et al. initialization from the paper, ***Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification***). You can read more about the importance of weight initialization and the convergence of deep neural networks inside ***All you need is a good init***, Mishkin and Matas (2015).

There are Two major Drawbacks with VGGNet:

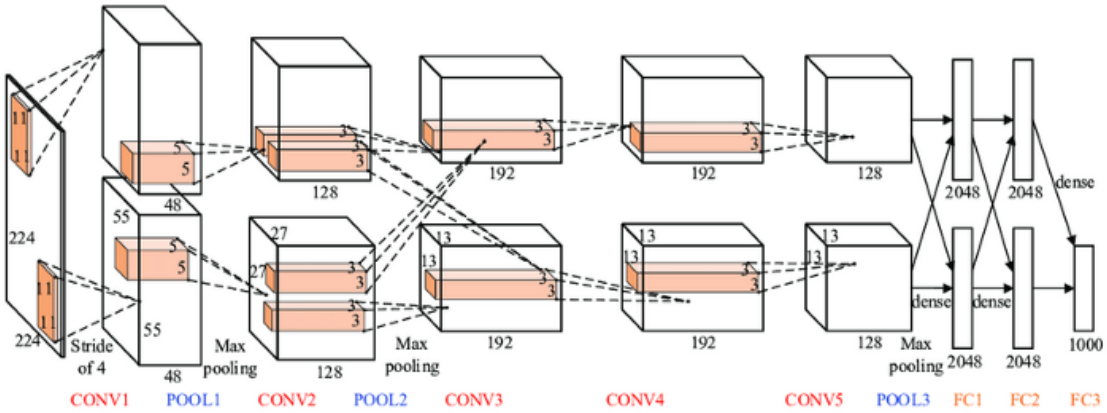
1. It is Painfully slow to train.
2. The network architecture weights themselves are quite large (in terms of disk/bandwidth)

Due to its depth and number of fully-connected nodes, VGG is over 533MB for VGG16 and 574MB for VGG19. This makes deploying VGG a tiresome task.

We still use VGG in many deep learning image classification problems; however, smaller network architectures are often more desirable (such as SqueezeNet, GoogLeNet, etc.).

AlexNet

This network was very similar to LeNet-5 but was deeper with 8 layers, with more filters, stacked convolutional layers, max pooling, dropout, data augmentation, ReLU and SGD. AlexNet was the winner of the ImageNet ILSVRC-2012 competition, designed by Alex Krizhevsky, Ilya Sutskever and Geoffery E. Hinton. It was trained on two Nvidia Geforce GTX 580 GPUs, therefore, the network was split into two pipelines. AlexNet has 5 Convolution layers and 3 fully connected layers. AlexNet consists of approximately 60 M parameters. A major drawback of this network was that it comprises of too many hyper-parameters. A new concept of Local Response Normalization was also introduced in the paper.



AlexNet Architecture

ResNet50

Unlike traditional sequential network architectures such as AlexNet, OverFeat, and VGG, ResNet is instead a form of "exotic architecture" that relies on micro-architecture modules (also called "network-in-network architectures").

The term micro-architecture refers to the set of "building blocks" used to construct the network. A collection of micro-architecture building blocks (along with your standard CONV, POOL, etc. layers) leads to the macro-architecture (i.e,. the end network itself).

First introduced by He et al. in their 2015 paper, **Deep Residual Learning for Image Recognition**, the ResNet architecture has become a seminal work, demonstrating that extremely deep networks can be trained using standard SGD (and a reasonable initialization function) through the use of residual modules:

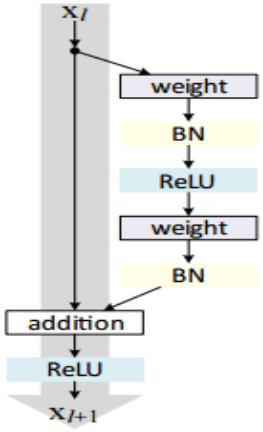


Figure 3: The residual module in ResNet as originally proposed by He et al. in 2015.

Further accuracy can be obtained by updating the residual module to use identity mappings, as demonstrated in their 2016 followup publication, **Identity Mappings in Deep Residual Networks**:

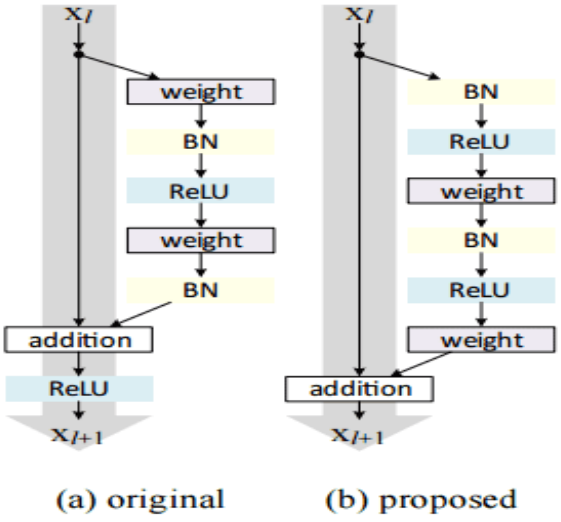


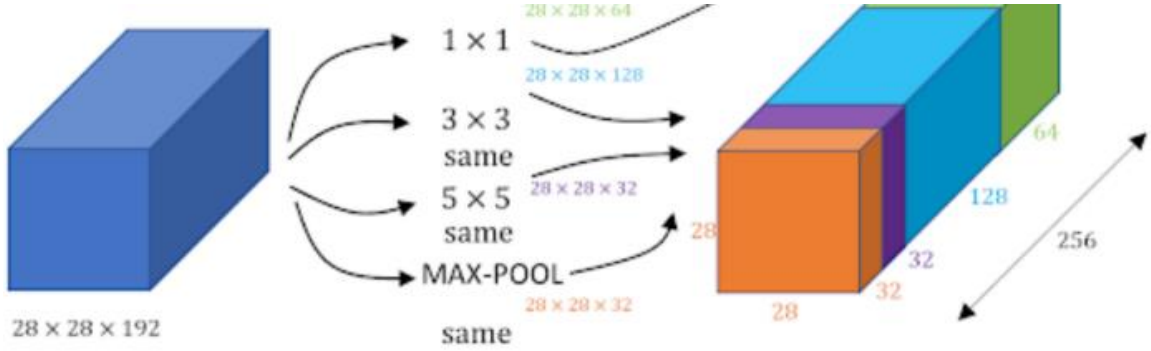
Figure 4: (Left) The original residual module. (Right) The updated residual module using pre-activation.

That said, keep in mind that the ResNet50 (as in 50 weight layers) implementation in the Keras core is based on the former 2015 paper.

Even though ResNet is much deeper than VGG16 and VGG19, the model size is actually substantially smaller due to the usage of global average pooling rather than fully-connected layers — this reduces the model size down to 102MB for ResNet50.

Inception Net(GoogleLeNet)

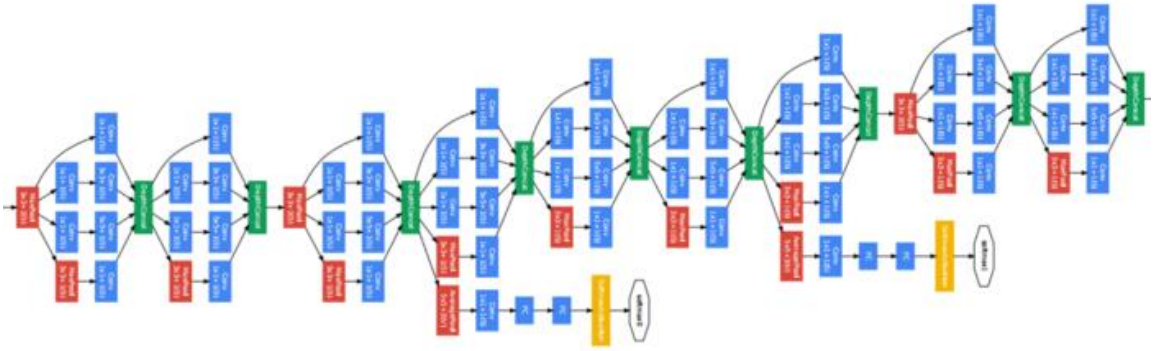
Inception network also known as GoogleLeNet was proposed by developers at google in “Going Deeper with Convolutions” in 2014. The motivation of InceptionNet comes from the presence of sparse features Salient parts in the image that can have a large variation in size. Due to this, the selection of right kernel size becomes extremely difficult as big kernels are selected for global features and small kernels when the features are locally located. The InceptionNets resolves this by stacking multiple kernels at the same level. Typically it uses 5*5, 3*3 and 1*1 filters in one go. For better understanding refer to the image below:



Inception Module of GoogleLe Net

Note: Same padding is used to preserve the dimension of the image.

As we can see in the image, three different filters are applied in the same level and the output is combined and fed to the next layer. The combination increases the overall number of channels in the output. The problem with this structure was the number of parameter (120M approx.) that increases the computational cost. Therefore, 1*1 filters were used before feeding the image directly to these filters that act as a bottleneck and reduces the number of channels. Using 1*1 filters, the parameter were reduced to 1/10 of the actual. GoogLeNet has 9 such inception modules stacked linearly. It is 22 layers deep (27, including the pooling layers). It uses global average pooling at the end of the last inception module. Inception v2 and v3 were also mentioned in the same paper that further increased the accuracy and decreasing computational cost.



Several Inception modules are linked to form a dense network

Side branches can be seen in the network which predicts output in order to check the shallow network performance at lower levels.

Summary

- Transfer learning generally refers to a process where a model trained on one problem is used in some way on a second related problem.
- Transfer learning has the benefit of decreasing the training time for a neural network model and can result in lower generalization error.
- The pre-trained model can be used as a separate feature extraction program, in which case input can be pre-processed by the model or portion of the model to a given an output/
- ImageNet is an ongoing research effort to provide researchers around the world with image data for training large-scale object recognition models.
- Even though ResNet is much deeper than VGG16 and VGG19, the model size is actually substantially smaller due to the usage of global average pooling.