Transfer Learning

Traditional ML

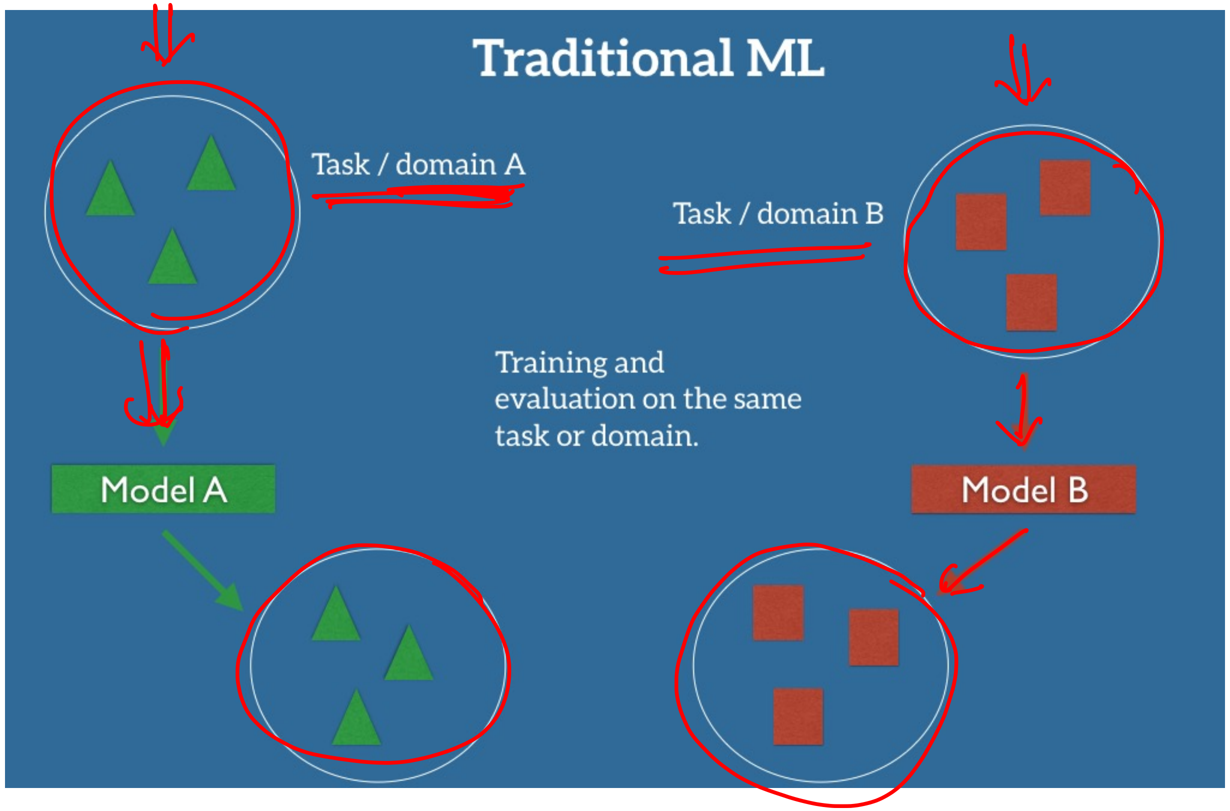
problem 1 (1) → labelled data (2) → cleansed data (3) → training (4) → model (5)

new data (6) → model → result (7)

problem 2 → → cleansed data → training → model

Traditional ML

- In the classic supervised learning scenario of machine learning, if we intend to train a model for some task and domain AA, we assume that we are provided with labelled data for the same task and domain
- We can see this clearly in Figure, where the task and domain of the training and test data of our model AA is the same
- Let us assume that
  - a task is the objective our model aims to perform, e.g. recognize objects in images
  - a domain is where our data is coming from, e.g. images taken in San Francisco coffee shops.

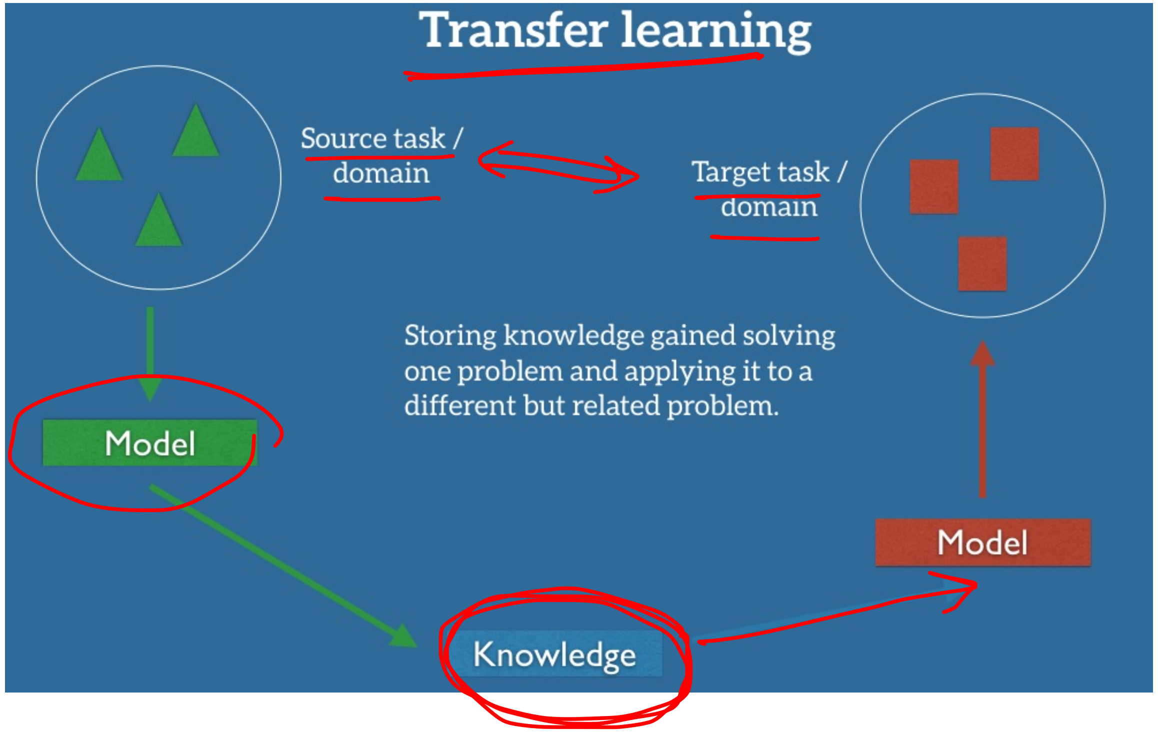


Traditional ML

- We can now train a model AA on this dataset and expect it to perform well on unseen data of the same task and domain
- On another occasion, when given data for some other task or domain BB, we require again labelled data of the same task or domain that we can use to train a new model BB so that we can expect it to perform well on this data
- The traditional supervised learning paradigm breaks down when we do not have sufficient labelled data for the task or domain we care about to train a reliable model

Transfer Learning

- Transfer learning allows us to deal with these scenarios by leveraging the already existing labelled data of some related task or domain
- We try to store this knowledge gained in solving the source task in the source domain and apply it to our problem of interest
- In practice, we seek to transfer as much knowledge as we can from the source setting to our target task or domain
- This knowledge can take on various forms depending on the data
  - it can pertain to how objects are composed to allow us to more easily identify novel objects
  - it can be with regard to the general words people use to express their opinions, etc

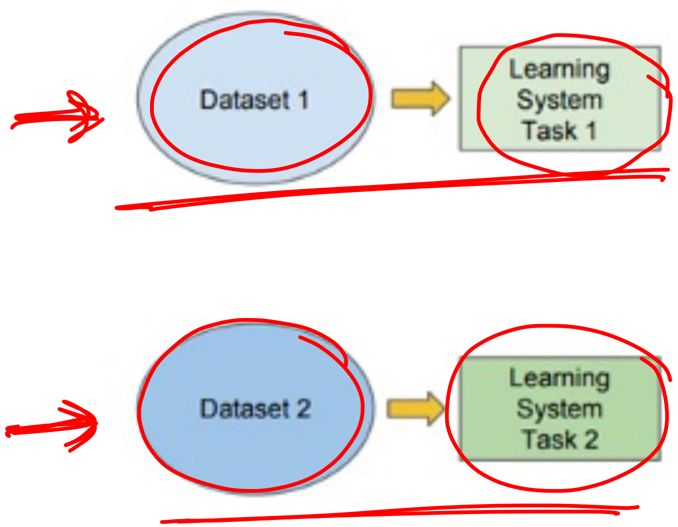


Traditional ML vs Transfer Learning

Traditional ML vs Transfer Learning

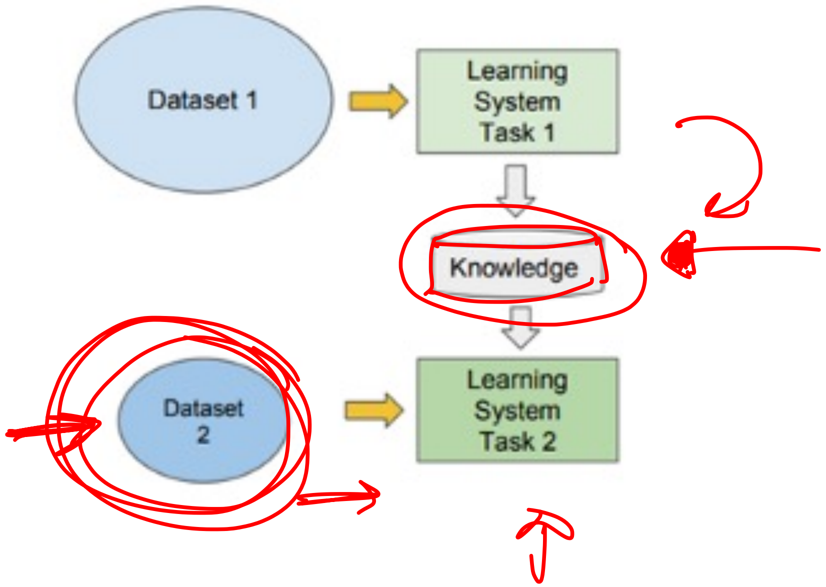
Traditional ML

- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



Transfer Learning

- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data



Important takeaways

- Transfer learning, as we have seen so far, is having the ability to utilize existing knowledge from the source learner in the target task
- During the process of transfer learning, the following three important questions must be answered
  - **What to transfer**
    - This is the first and the most important step in the whole process.
    - We try to seek answers about which part of the knowledge can be transferred from the source to the target in order to improve the performance of the target task
    - When trying to answer this question, we try to identify which portion of knowledge is source-specific and what is common between the source and the target

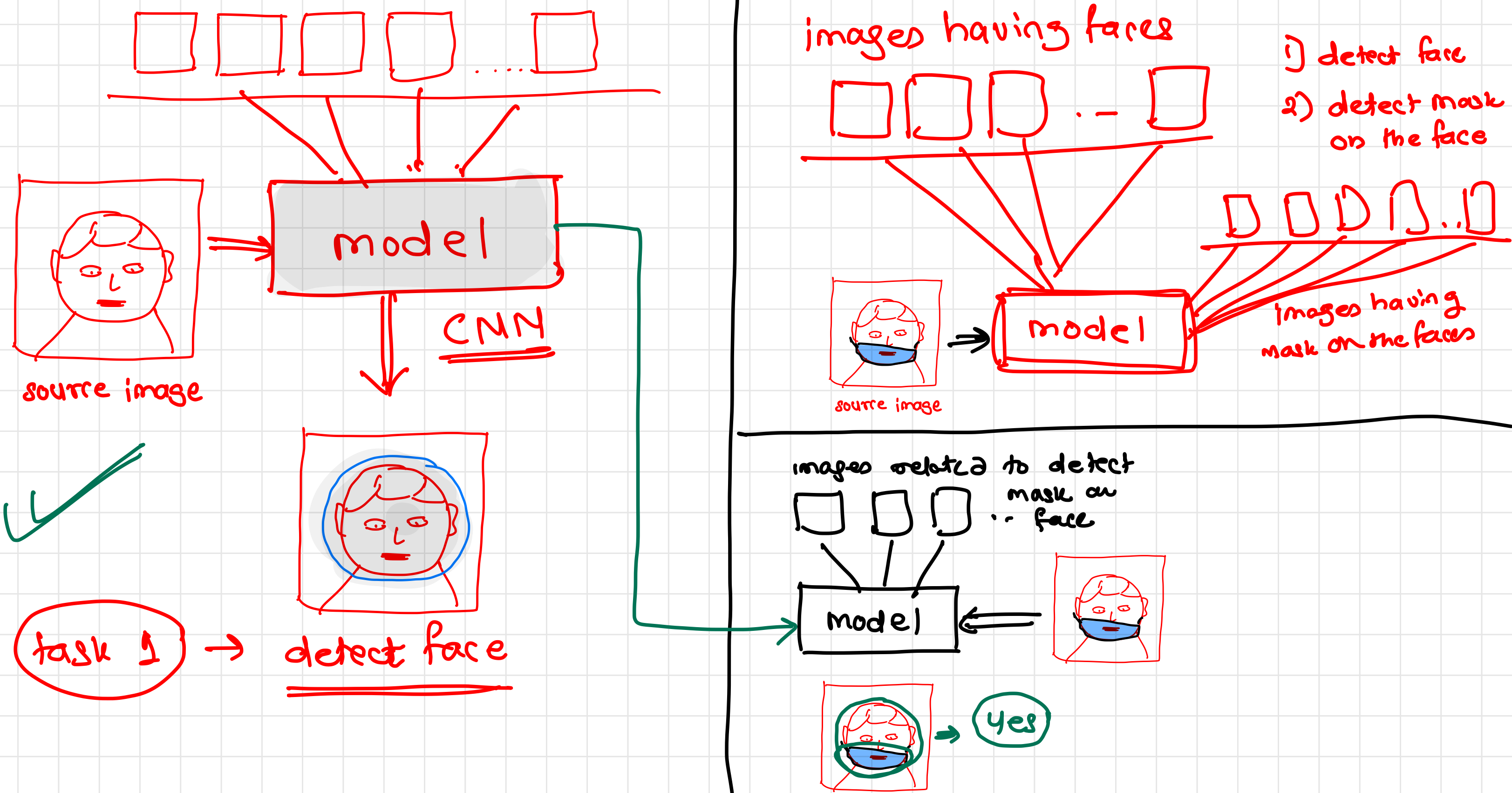
model → weights

  - **When to transfer**
    - There can be scenarios where transferring knowledge for the sake of it may make matters worse than improving anything (also known as negative transfer)
    - We should aim at utilizing transfer learning to improve target task performance/results and not degrade them
    - We need to be careful about when to transfer and when not to

[transfer only if tasks are Related]

task 1   detect face

task 2   detect mask on face

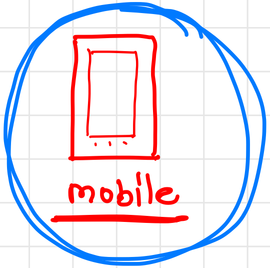


Important takeaways

- **How to transfer**
  - Once the *what* and *when* have been answered, we can proceed towards identifying ways of actually transferring the knowledge across domains/tasks
  - This involves changes to existing algorithms and different techniques

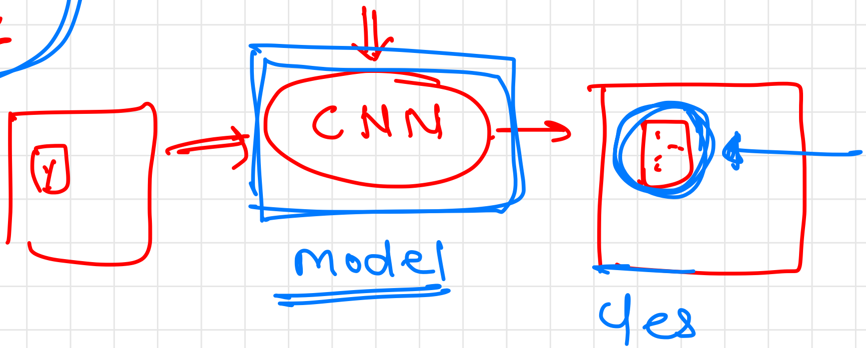
    - framework/library dependent

Object detection



① load images containing objects

② create the model



CNN

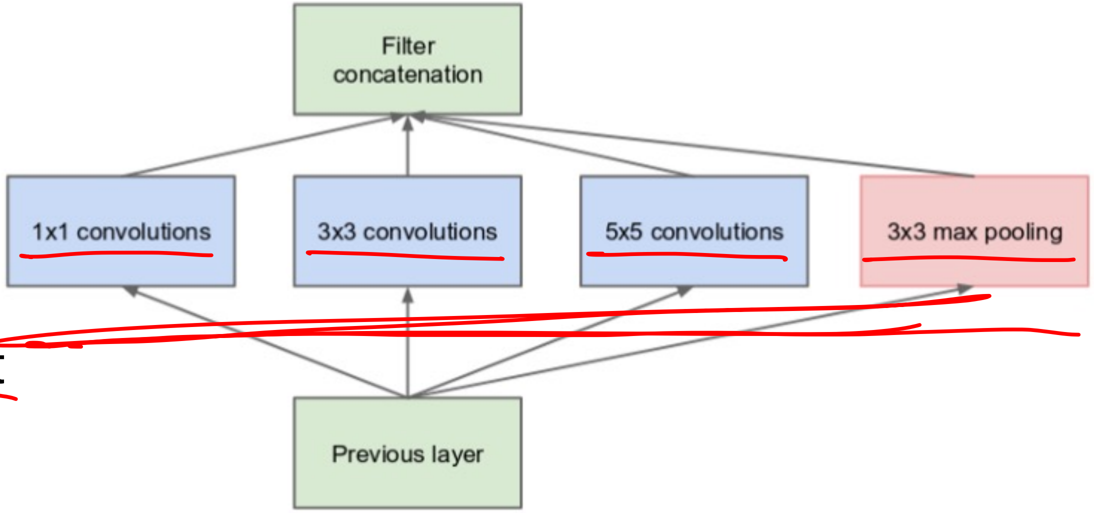
Inception Network

Introduction

- Inception Modules are used in CNN to allow for more efficient computation and deeper Networks through a dimensionality reduction with stacked 1×1 convolutions
- The modules were designed to solve the problem of computational expense, as well as overfitting, among other issues
- The solution, in short, is to take multiple kernel filter sizes within the CNN, and rather than stacking them sequentially, ordering them to operate on the same level
-

How does it work ?

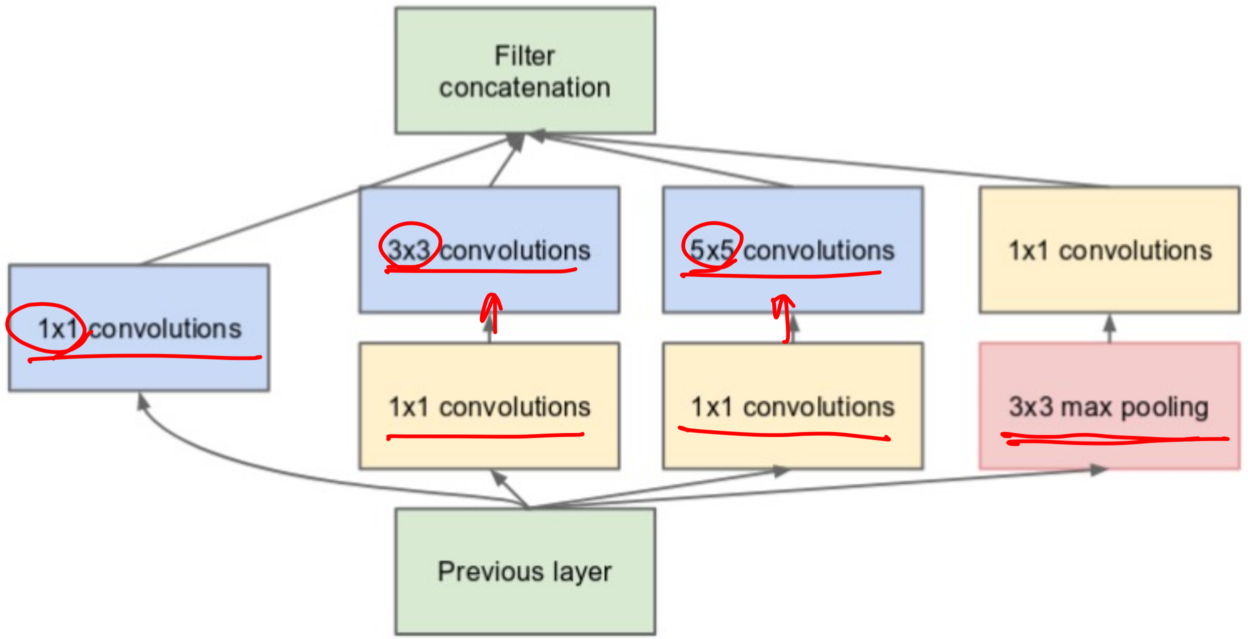
- Inception Modules are incorporated into convolutional neural networks (CNNs) as a way of reducing computational expense
- As a neural net deals with a vast array of images, with wide variation in the featured image content, also known as the salient parts, they need to be designed appropriately
- The most simplified version of an inception module works by performing a convolution on an input with not one, but three different sizes of filters (1x1, 3x3, 5x5)
- Also, max pooling is performed. Then, the resulting outputs are concatenated and sent to the next layer
- By structuring the CNN to perform its convolutions on the same level, the network gets progressively wider, not deeper



(a) Inception module, naïve version

How does it work ?

- To make the process even less computationally expensive, the neural network can be designed to add an extra 1x1 convolution before the 3x3 ad 5x5 layers
- By doing so, the number of input channels is limited and 1x1 convolutions are far cheaper than 5x5 convolutions
- It is important to note, however, that the 1x1 convolution is added after the max-pooling layer, rather than before



(b) Inception module with dimension reductions

How does it work ?

- The design of this initial Inception Module is known commonly as GoogLeNet, or Inception v1
- Additional variations to the inception module have been designed, reducing issues such as the vanishing gradient descent

RNN