Convolutional Neural Networks

Topics covered so far

- Convolutional Neural Networks
  - Locality, Translation invariance
  - Filters / Convolutions
  - Pooling layer
  - Architecture of CNN
  - Illustration of what CNNs learn

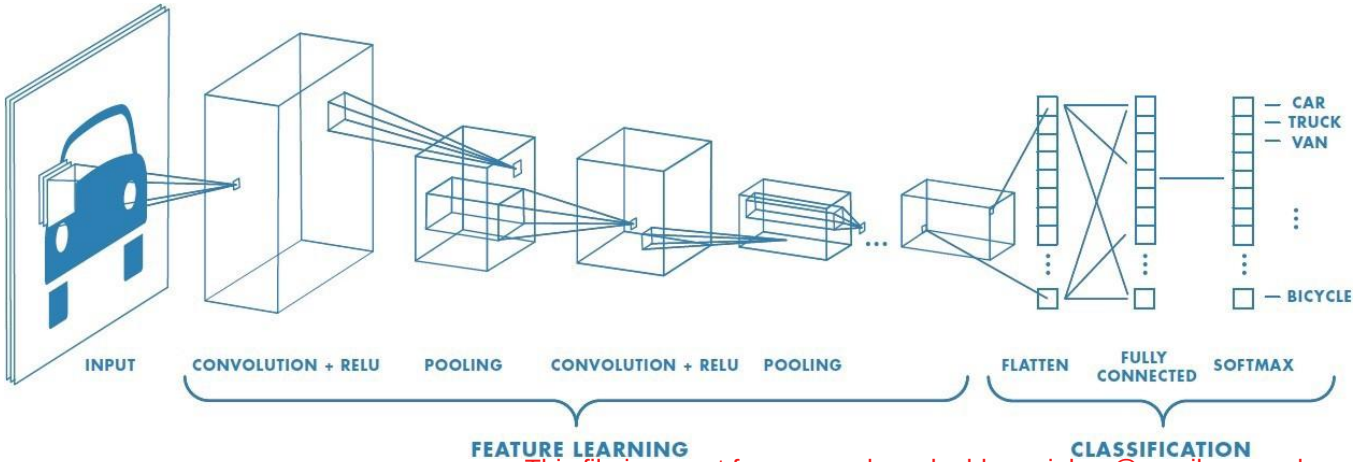
Discussion questions

1. What are convolutional neural networks and how are they different from ANNs?
2. How do filters/kernels work for feature detection in CNNs?
3. How do pooling, padding, and stride operations work in CNNs?
4. What is transfer learning and how can we use that in CNNs?

Convolutional Neural Networks

CNNs are a special type of neural network designed to work with the image data. CNNs use convolutional layers, i.e., hidden layers which perform convolution operations. They have some different characteristics to ANNs:

1. Unlike ANNs, CNNs capture the spatial structure of the image.
2. CNNs follow the concept of parameter sharing, i.e., one filter is applied over the whole image, because of which they are much more computationally efficient.



The first part in this architecture is the convolutional layer followed by the pooling layer and the second part is the fully connected layer. This whole architecture is called a convolutional neural network.

Image Source

The convolutional layer - Filter/Kernel

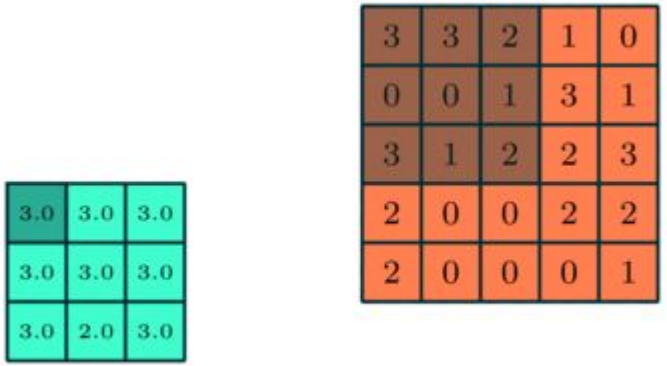
- A convolution operation uses a small array of numbers called a filter/kernel on the input image.
- Each filter is designed to identify a specific feature in the input space of the image, such as horizontal edges, vertical edges, etc.
- A CNN can successfully capture the spatial and temporal dependencies in an image through the application of relevant filters.
- The role of the CNN is to reduce the images into a form which is easier to process, without losing features which are important for getting a good prediction.

- This image shows how the convolution operation works in a CNN
  - It uses a 3x3 filter on a 5x5 image
  - The resulted feature is a 3x3 image which is the convolved feature

Image Source

Pooling layer in CNNs

- After a convolution operation, we usually perform pooling to reduce the dimensions of the feature map.
- It enables us to reduce the number of parameters, which both reduces the training time as well as the overfitting.
- Pooling layers downsample each feature map independently, reducing the height and width, but keeping the depth same.



The two most common types of pooling are - Max and Average:

- Max pooling just takes the maximum value, whereas average pooling takes the average value in the pooling window
- Contrary to the convolution operation, pooling has no parameters.
- In these gifs, we can see how max and average pooling work.

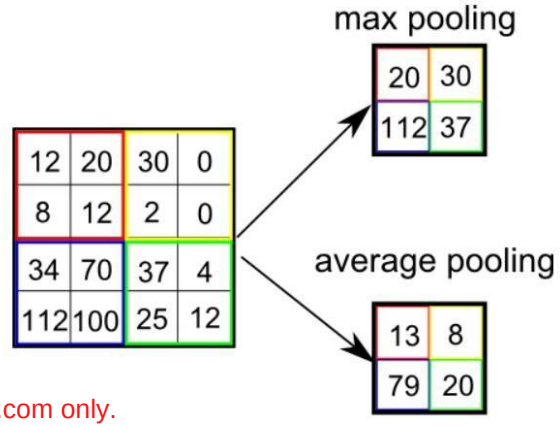
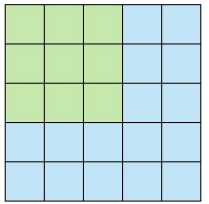


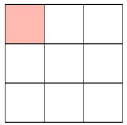
Image Source

Padding & Stride in CNNs

- **Stride** specifies how much we move the filter at each step. By default, the value of the stride is 1 and is represented by the first figure.
- We can also increase the value of stride if we want less overlap between the filters. It also makes the resulting feature map smaller since we are skipping over some locations.
- The second figure demonstrates the stride 2

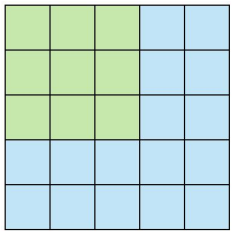


Stride 1

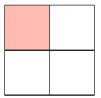


Feature Map

We see that after using stride 2, the size of the feature map is smaller than the input. If we want to maintain the same dimensions, we can use **padding** to surround the input with zeros.

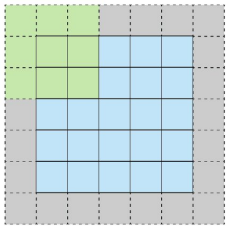


Stride 2

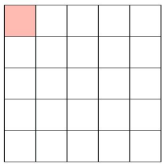


Feature Map

- The grey area around the input in the third figure is the padding.
- We either pad with zeros or the values on the edge, to match the dimensions of the feature map with the input.
- Padding is commonly used in CNNs to preserve the size of the feature maps, otherwise they would shrink at each layer, which is not desirable.



Stride 1 with Padding



Feature Map

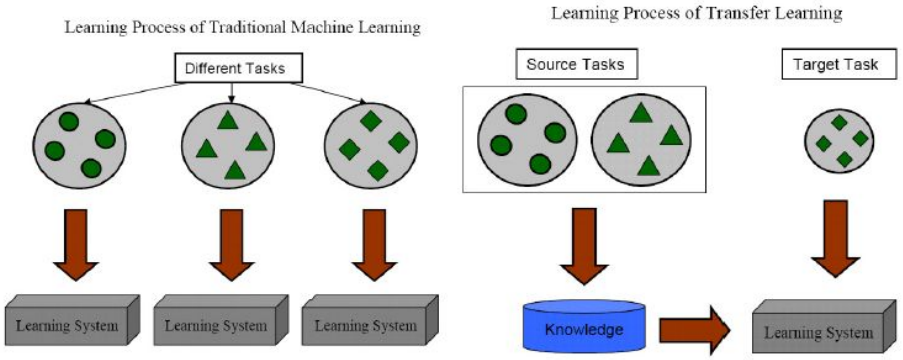
Image Source

Topics to be covered

- Transfer Learning

Transfer Learning

- If you are using a specific neural network architecture that has been trained before, you can use these pretrained parameters/weights instead of random initialization to solve the problem.
- It can help boost the performance of the neural network.
- The pretrained models might have trained on large datasets like ImageNet, and taken a lot of time to learn those parameters/weights with optimized hyperparameters. This can save a lot of time.
- If you have enough data, you can fine tune all the layers in your pretrained network but don't randomly initialize the parameters, leave the learned parameters as they are and learn from there.



Case Study

Appendix

Graph Neural Networks

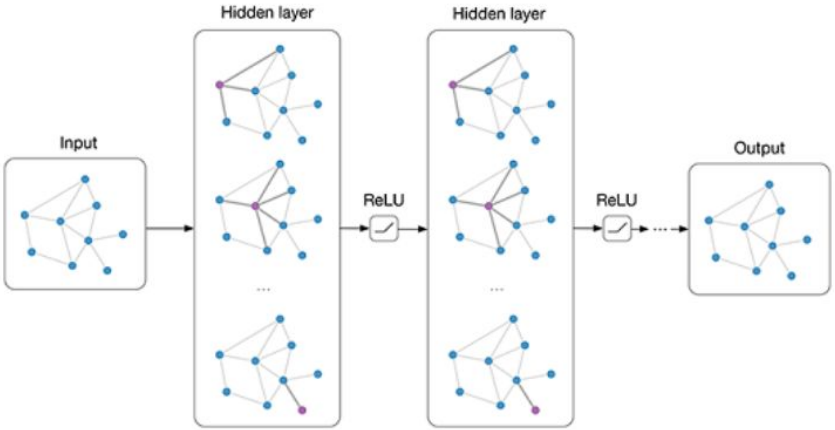
Graph Neural Network is a type of Neural Network that directly operates on the Graph structure. It takes graphs as an input.

We need GNNs because CNNs do not perform well on graph data due to the following reasons:

- Graphs do not have a fixed order of neighbours
- They do not have a fixed node ordering
- They are arbitrary in size

GNNs are used when we want to:

- Classify the nodes
- Classify the graphs
- Predict the edges and other features



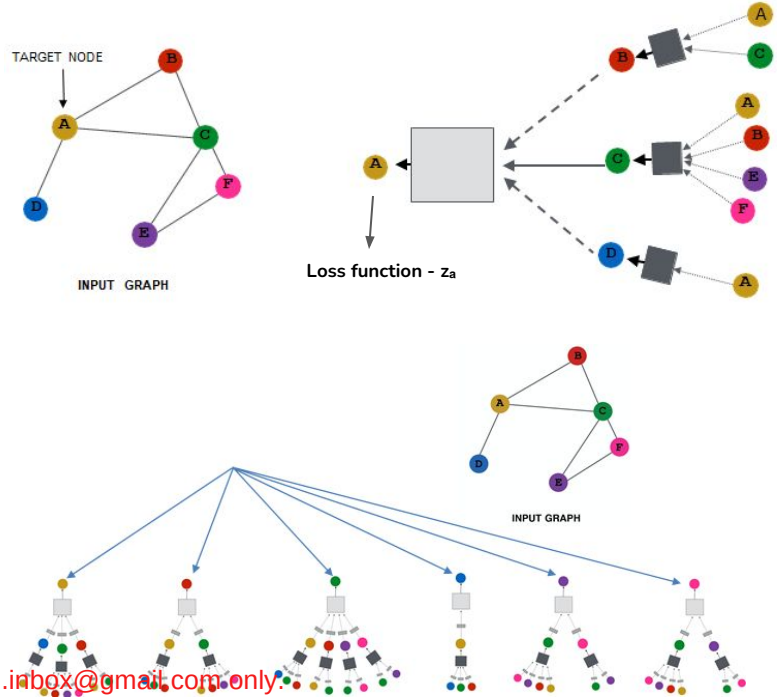
This image shows how GNNs work and predict the output.

Learning a GNN

Each node has a set of features defining it. In the case of social network graphs, this could be age, gender, country of residence, political leaning, and so on. Each edge may connect nodes together that have similar features. It shows some kind of interaction or relationship between them.

GNNs work in the following steps:

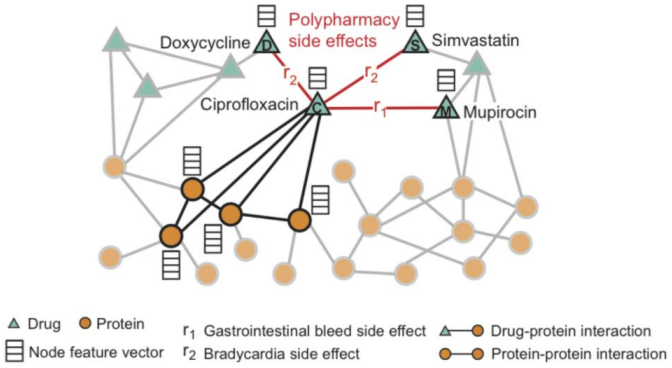
- We generate node embeddings based on local neighborhoods.
- The intuition behind neighborhood aggregation is that nodes aggregate information from their neighbors using neural networks.
- Then we specify a loss function on the output embedding for a target node (node A for example)
- We define a computational graph for each target node and each node defines a unique computational graph; we then train this on all the data points.
- Then we can generate embeddings for nodes that we have not trained on.

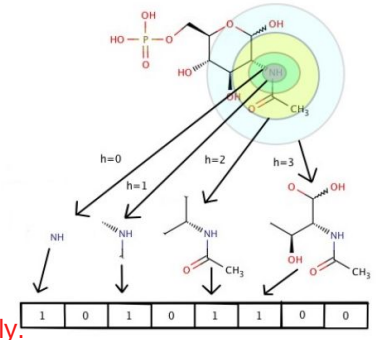


Applications of GNNs

These are some of the real-life applications of GNNs:

1. **Predicting side effects due to drug interactions:** We can predict what happens when two or more than two drugs interact with each other.
2. **Node Importance in Knowledge Graphs:** Using knowledge and product graphs to capture the relationships between product data and the critical context that humans have but machines lack. Such graphs enable machines to excel at downstream applications like product recommendations and question answering.
3. **Using GNNs in chemistry:** GNNs can be applied in both scenarios: learning about existing molecular structures as well as discovering new chemical structures. This has had a significant impact in computer-aided drug design.





Great Learning

Happy Learning !