**DL theory : Assingments-6**

1. What are the advantages of a CNN over a fully connected DNN for image classification? CNNs (Convolutional Neural Networks) are well-suited for image classification tasks due to their ability to learn spatial hierarchies of features. This allows them to learn both local and global features, making them more robust to variations in the position and scale of objects in an image. Additionally, CNNs use convolutional layers which reduces the number of parameters, as compared to fully connected DNNs, this reduces the risk of overfitting.
2. Consider a CNN composed of three convolutional layers, each with 3 × 3 kernels, a stride of 2, and "same" padding. The lowest layer outputs 100 feature maps, the middle one outputs 200, and the top one outputs 400. The input images are RGB images of 200 × 300 pixels. What is the total number of parameters in the CNN? If we are using 32-bit floats, at least how much RAM will this network require when making a prediction for a single instance? What about when training on a mini-batch of 50 images? The total number of parameters in the CNN is: (3x3x3+1)x100 + (3x3x100+1)x200 + (3x3x200+1)x400 = 33,600 + 180,400 + 720,800 = 1,034,800 Each feature map is a 2D array of size (200/2)/2 x (300/2)/2 = 25 x 37 So the RAM required for a single instance is 25x37x400x4 bytes = 6,4 MB When training on a mini-batch of 50 images, the RAM required would be 6.4MB x 50 = 320MB
3. If your GPU runs out of memory while training a CNN, what are five things you could try to solve the problem?
4. Reduce the batch size
5. Decrease the image resolution
6. Remove or decrease the number of filters in the layers
7. Use a more efficient data format, such as float16 instead of float32
8. Use a smaller network architectureTop of Form
9. Build your own CNN from scratch and try to achieve the highest possible accuracy on MNIST. To build a CNN from scratch to achieve the highest possible accuracy on MNIST, you can follow these steps:
10. Prepare the data by loading it into memory, normalizing it, and splitting it into training, validation, and test sets.
11. Define the architecture of the CNN using Keras, Tensorflow or Pytorch. The architecture typically consists of a series of convolutional, pooling, and fully connected layers.
12. Train the model by defining the loss function, optimizer, and metrics for evaluation.
13. Tune the hyperparameters of the model such as learning rate, batch size, and number of epochs to improve the performance.
14. Evaluate the model on the test set and compare the results with the state-of-the-art models.
15. Use transfer learning for large image classification, going through these steps: a. Create a training set containing at least 100 images per class. For example, you could classify your own pictures based on the location (beach, mountain, city, etc.), or alternatively you can use an existing dataset (e.g., from TensorFlow Datasets). b. Split it into a training set, a validation set, and a test set. c. Build the input pipeline, including the appropriate preprocessing operations, and optionally add data augmentation. d. Fine-tune a pretrained model on this dataset. e. Evaluate the model on the test set and compare the results with the pretrained model.
16. Transfer learning is a technique where a model trained on one task is used as the starting point to train a model on a second, related task. The idea is to leverage the knowledge learned from the first task to improve the performance on the second task. In the case of image classification, a pre-trained model on a large dataset such as Imagenet can be used as a starting point. The last fully connected layer of the pre-trained model can be replaced with a new fully connected layer, which is then trained on the new dataset. This allows the model to learn features specific to the new dataset, while still utilizing the knowledge learned from the pre-trained model.