Program syllabus for : Classroom

Varun Kapoor

Kapoorlabs, Paris, France

February 2022

1 Introduction to AI based image analysis

1.1 Package installation

In this course we will use Goolge Colab to train the models and local Napari notebooks to create the annotation data. For installing Napari on the local notebooks, please do the following steps:
Local Installation

- Download Anaconda individual edition, after the download and installation is complete you can create virtual enviornment by either opening an anaconda powershell (windows) or by opening a terminal (Mac/Linux)
- From either of the terminals type: **create -n naparienv python==3.9**, this create a virtual enviornment using python 3.9 and inside this enviornment we can install all the python packages required for the computation.
- To activate the enviornment do **conda activate naparienv**
- Now we can pip install the required packages, run **pip install vollseg-napari** to install all the required packages

Colab Installation For the colab notebooks, no installation is required, all the notebooks have already been preset for the tasks of training the models and running the model predictions on the test datasets.

1.2 Creating training data for instance segmentation

To create the training data for instance segmentation we use Napari notebooks that creates a label layer with the annotations performed on the cells. Each cell is annotated with a unique label and all the cells in the given field of view are annotated, this is referred to as creating dense labels where all the pixels in the image are annotated as belonging to a cell or background (background pixels by default are valued at 0).

1.3 Creating training data for image classification

To create the training data for classification tasks we use a notebook which is used to mark the locations of cells of certain type. The program UI allows selecting an image/timelapse image from a directory then the users can select the cell type of interest to mark the locations and save the annotations performed using the Save button. As opposed to the classification tasks the labelling can be sparse, which means not all the given cell type in the field of view have to be annotated/marked but only few examples across all classes are sufficient to create a good training dataset which in this case comprises of csv file containing the location of the marked cells for each cell type. The same program can also be used to mark the location of action events, the succeeding program then creates either training patches in space on in space and time along with automatically generating the training labels for training the classification networks.

1.4 Bias Variance tradeoff

Consider solving a segmentation optimization problem. Task of the solver is to find the function parameters that can learn the mapping function to create a segmentation map. The function has many parameters, let us denote them by $\vec{theta}$. In determining these parameters there are two sources of errors, one is bias which measures the expected deviation of the function from the true value of the function. Other comes from sampling of the data that causes deviation from the expected estimator value. Good deep learning models have low bias and variance. To have deep learning models that have low bias and variance it is common to compare the mean squared error of the estimators (MSE).

$$\begin{aligned} \text{MSE} &= E[(\vec{\theta}_m - \vec{\theta})^2] = E[\vec{\theta}_m^2] + \vec{\theta}_m^2 - 2E[\vec{\theta}_m]\vec{\theta}_m \\ &= E[\vec{\theta}_m^2] - E[\vec{\theta}_m^2] + E[\vec{\theta}_m^2] + \vec{\theta}_m^2 - 2E[\vec{\theta}_m]\vec{\theta}_m \\ &= \text{Bias}(\vec{\theta}_m)^2 + \text{Var}(\vec{\theta}_m) \end{aligned} \tag{1}$$

From the above equation minimizing the mean squared error is equivalent to finding an estimation for the model that has low bias and low variance.

1.5 Keras features for model training

1.6 Using keras to create encoder-decoder networks

1.7 Model hyperparameters

1.8 Activation functions

Activation functions of the last layer are a crucial choice and depends on the task, if it is a segmentation task the final activation is linear as it estimates the mean value of the Gaussian distribution (assumption being that the data generating distribution is Gaussian). If the task is image classification the last activation function is either softmax or sigmoid. The activation functions are strongly coupled to the choice of the training loss. For segmentation task the training loss is mean absolute or mean squared error while for classification tasks is categorical or binary cross entropy.

1.9 Before Training: Over/Underfitting

Before starting the model training, it is essential to split the data into training and validation datasets. This is to perform cross validation to judge if the model is over/under fitting on the training dataset. Such information can be seen in the training and validation loss curves.
Your model is overfitting on the data if you see the validation loss curve rising instead of following the training loss curve. This gap between the two is called the generalization gap and is a result of high variance of the estimator 1.4. High variance comes due to large coefficients in the weight vector. Regularization is a technique used to penalize such large coefficients in the weight vector. Increasing the regularization coefficient adds to the penalty being higher, **increasing regularization coefficient helps avoid over fitting**. Your model can be overfitting on the data if the **training data is not enough**, **either increase the training data or use data augmentation** to avoid overfitting. It is also a good idea to check if your **model is too complex**, reducing the number of hidden units (reducing depth, number of convolutional filters) also helps in avoiding overfitting. You can also add a **keras callback for early stopping to avoid overfitting**. The opposite sitution to this is underfitting.
Your model is underfitting on the data if you see high training and validation loss that does not improve. This situation arises due to high bias of the estimator 1.4. In this case **increasing the model complexity** by adding more hidden neurons, increasing the network depth and or the number of convolutional filters would help. If you are using regularization then **decreasing the regularization coefficient** and increasing the training time simultaneously would avoid this problem.