19) CNN

CNN Architecture

- Made of 3 layers:

① Convolution – learn kernels (filters) to create feature maps of data

② Pooling – Downsample feature maps

③ Fully Connected – Categorical/Continuous Prediction

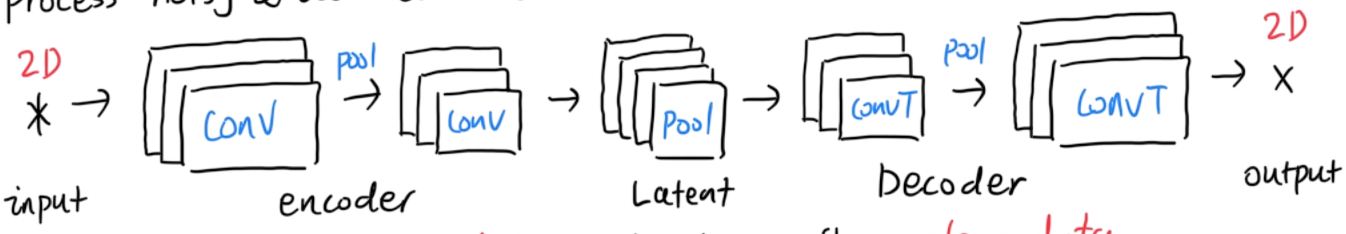
- Typical structure: $X \xrightarrow{2D}$ (Convolution → actFun → Pooling) ^{*N} $\xrightarrow{1D}$ Fully Connected → $\hat{y}$

  + RF of each pixel by: add wider (+ # feature maps) layers for – pixel size

CNN Autoencoder

- Used to process noisy & occluded images

· Structure: 2D



block

· Train workflow: Feed model noisy data → Backprop it on clean data

Custom Loss Function

- Exceeds the boundary of implementing PyTorch loss functions

· Code:

```
class myLossFunc(nn.Module):
    def _init_(self): super._init_()
    def forward(self, yhat, y):
        return formula result
```

① L1 loss: $L(\hat{y}, y) = m^{-1}\sum_{i=1}^{m}|\hat{y}_i - y|$

② L2 avg loss: $MSE + m^{-1}\left|\sum_{i=1}^{m}\hat{y}_i\right|$

③ Correlation: $\dfrac{\sum_{i=1}^{m}(\hat{y}_i - \mu_{\hat{y}})(y_i - \mu_y)}{(m-1)\,\sigma_{\hat{y}}\,\sigma_y}$

· ① is useful for image processing as it promotes sparse weights

② classification/LLMs distributed weights + small output

DL vs. Stats

- Model workflow: $\Theta \xrightarrow{\text{generates}} X$, $X \xrightarrow{\text{predict}} \hat{y}$ – mostly DL job, $X \to \hat{\Theta}$ – mostly stats job

  params, distributed data

Dropout in Convolution Layers

- Introduces noises to feature maps to help generalization

· Apply small dropout rate (.1~.25) so it won't break spatial dependence in weights