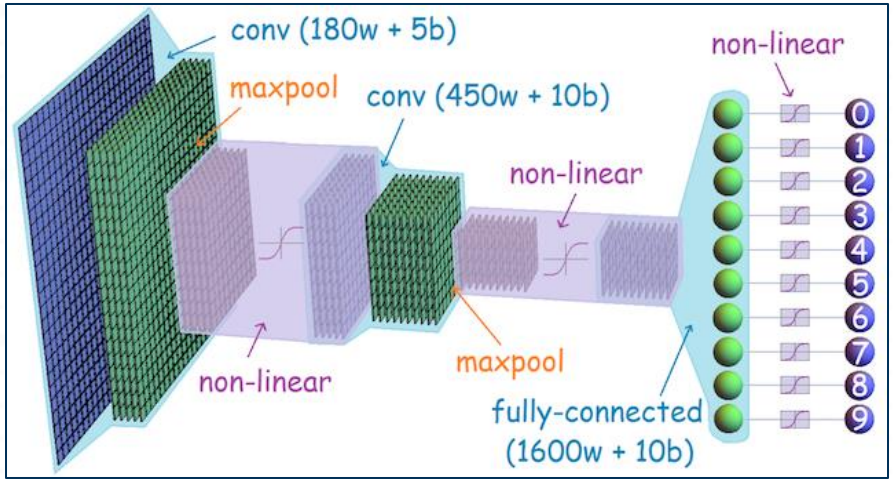
ECE 4252/8803: Fundamentals of Machine Learning (FunML)
Fall 2024

Lecture 17: Best Practices



OLIVES @GeorgiaTech

Georgia Tech

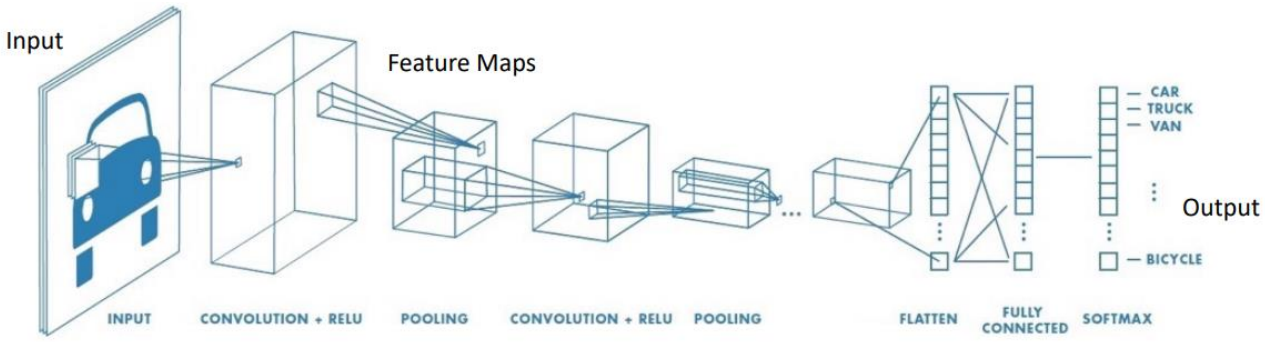
Which Architectures to Use?

- There is no one-size-fits-all solution here
- Use the architecture best suited to the application
- Broadly speaking, CNN architectures may be divided into two kinds:
  1. Those used for classification
  2. Those used for dense prediction tasks e.g., segmentation, reconstruction … etc.
- After having determined the application and the corresponding architecture type to use with it, search for baseline models made available to the public by people who worked on the problem before you
- Some places to look for: GitHub, PyTorch and Tensorflow tutorials … etc.
- Chances are you will already have a dozen solutions tackling the problem or a similar one; *no need to re-invent the wheel!*

Which Architectures to Use?

Classification Architectures

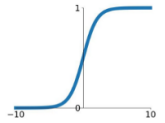
- CNN architectures for classification follow a typical pipeline processing input images via multiple 2D kernels across multiple, stacked layers to result in increasingly smaller activations
- Output of the last convolution layer unrolled and vectorized, passed to fully connected layers and finally a softmax layer to output pseudo-probabilities for the input belonging to different target classes
- Size of this logit vector equal to number of target classes
- Number of kernels should be steadily increased starting from the first convolution layer
- Experiment with different pooling options
- Experiment with different activation functions
- Experiment with dropout
- Experiment with skip connections (e.g., as in Resnet described in earlier lectures)



Activation Functions

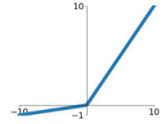
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



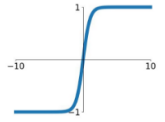
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

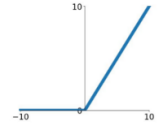


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

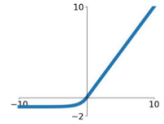
ReLU

$\max(0, x)$



ELU

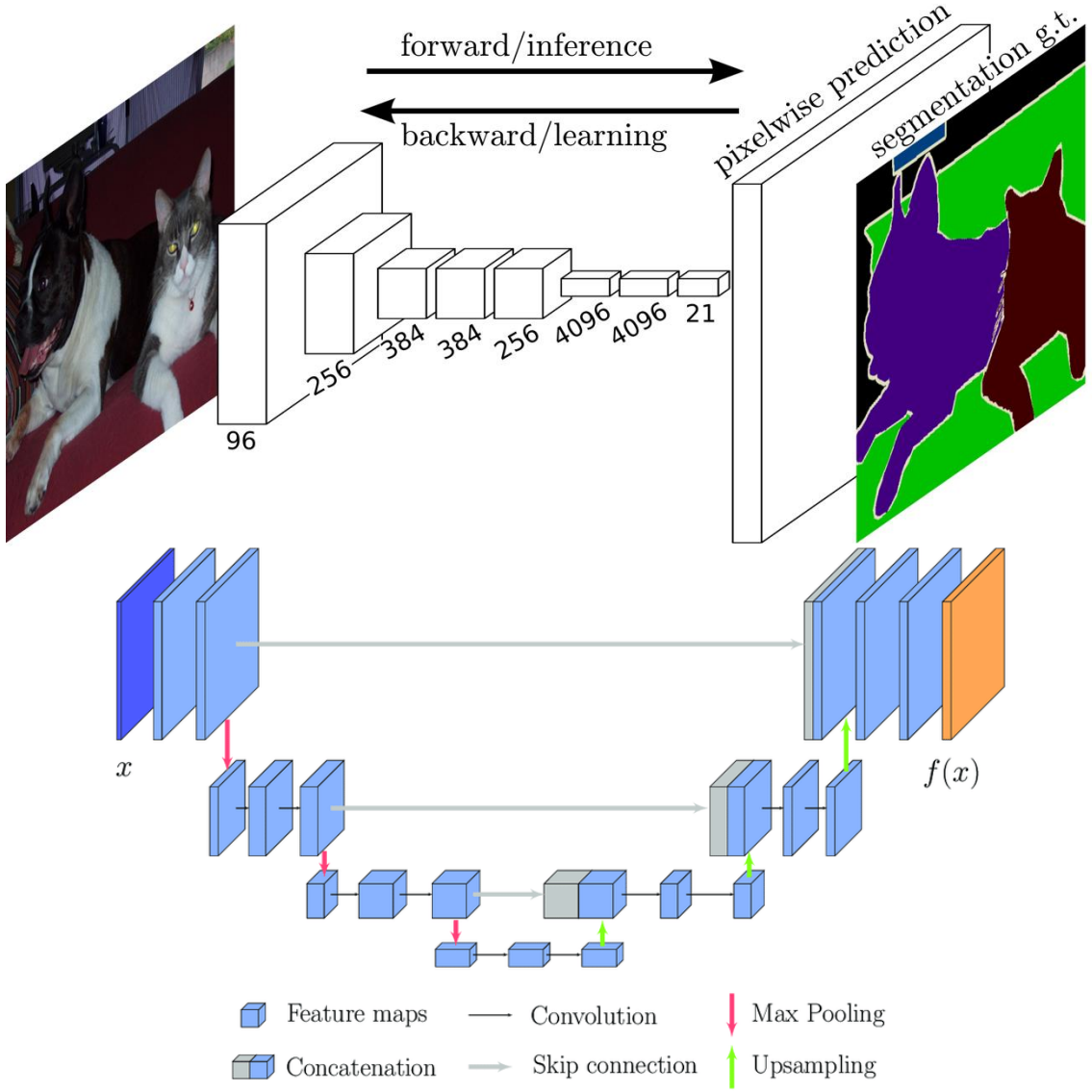
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Which Architectures to Use?

Dense Prediction Architectures – next 2 lectures

- Many applications require prediction of dense outputs e.g., image super-resolution, denoising, semantic segmentation etc.
- Suboptimal to use classification architectures since pooling operations would reduce input resolution and result in deteriorated output quality
- Hence use fully convolutional architectures
- Typically follow an encoder-decoder configuration
- Encoder similar to a classification type architecture without fully connected layers
- Decoder works to progressively upsample encoder output to desired output resolution via transposed/deconvolution layers



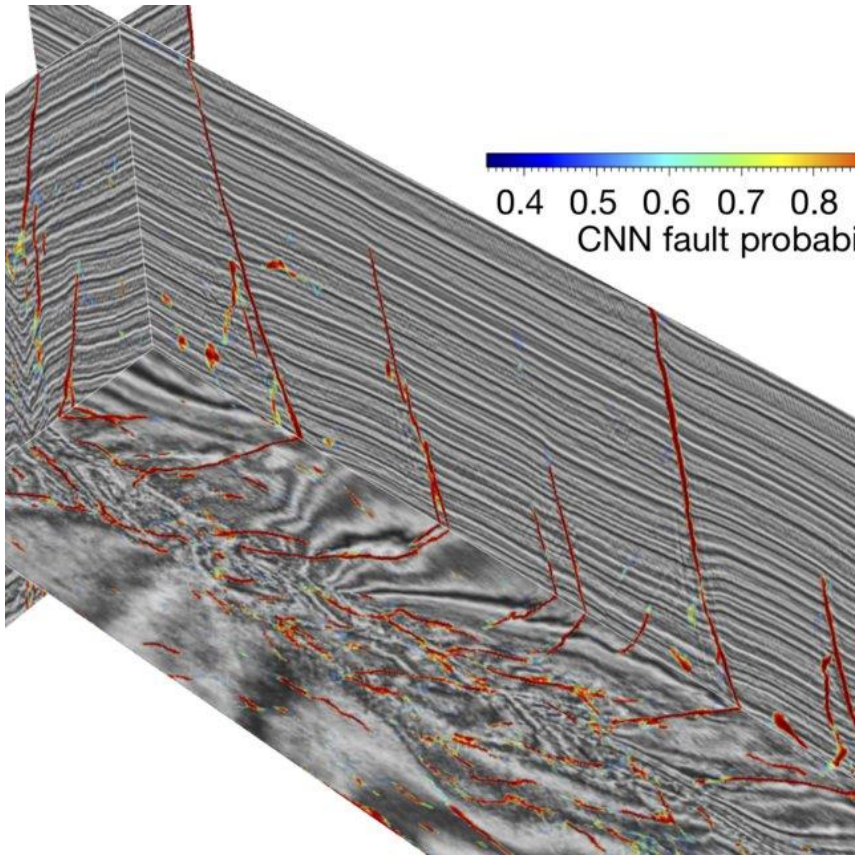
Which Architectures to Use?

Loss Functions

- The choice of the loss function to use with the architecture should be dictated by the nature of the problem
- For classification tasks, a typical cross-entropy based loss would be the obvious choice
- For segmentation tasks, the cross-entropy loss would happen on per pixel basis for each pixel of the output
- For super-resolution, denoising problems, mean square error loss function or L1 loss would be a good choice
- Special considerations for when there is class imbalance (more on next slide)

Class Imbalance

- Regular cross entropy loss function fails in situations where one of the classes occurs much more frequently than other(s)
- Observe the picture to the right: the network is much more likely to predict no-fault pixels on the seismic volume compared to fault pixels
- Same situation with most anomaly, aberrant-event detection problems
- Regular cross-entropy loss would make the network predict the majority class just by virtue of its overwhelming presence in the training dataset



Class Imbalance

Modified Cross-entropy Loss

- **Regular cross entropy:**

$$L = \sum_i y_i \log \hat{y}_i,$$

where i refers to class number, y_i to the ground-truth label for the correct class, and $\hat{y}_i$ the softmax pseudo-probability predicted by the network

- **Weighted cross entropy loss:**

$$L = \sum_i w_i \times y_i \log \hat{y}_i,$$

Where w_i refers to weight assigned to class i. Depending on the class frequencies, assign more weight to infrequent/aberrant classes

- A simple rule of thumb:

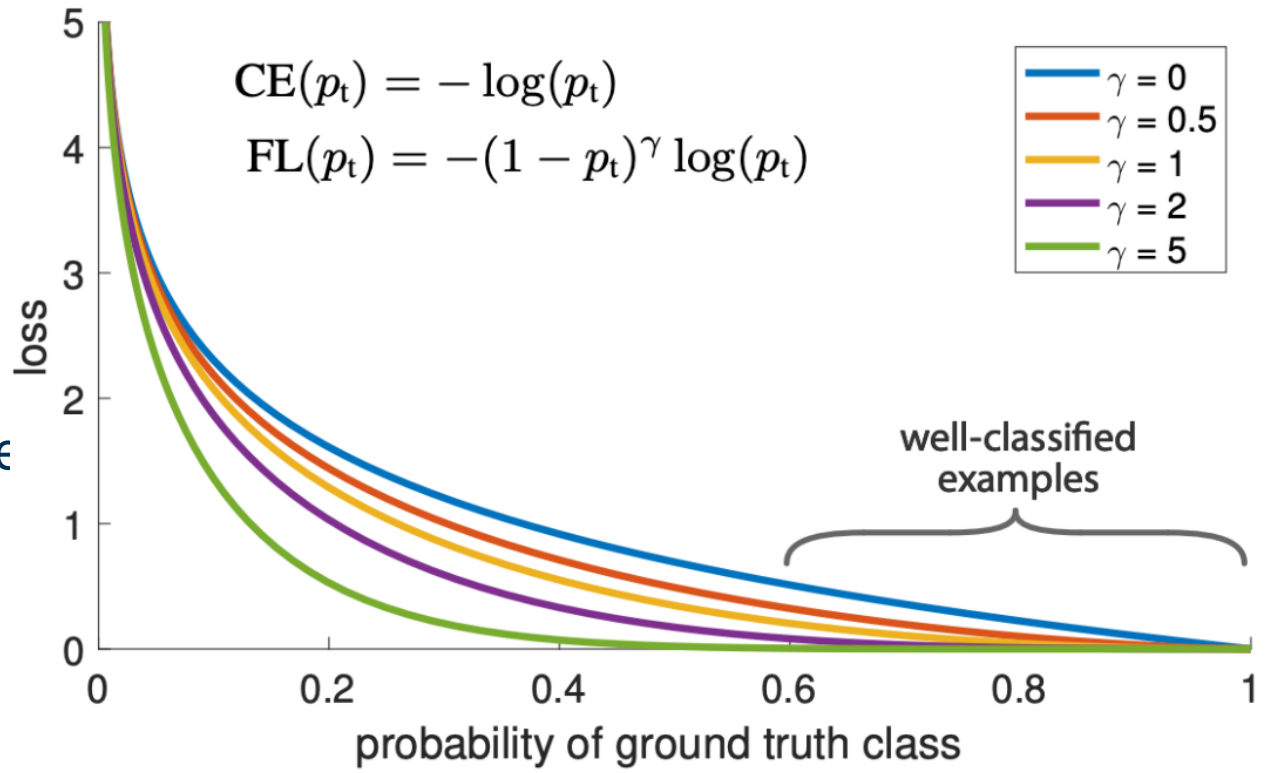
$$w_i = \frac{1}{f_i},$$

Where f_i refers to frequency of class i in the training set.

Class Imbalance

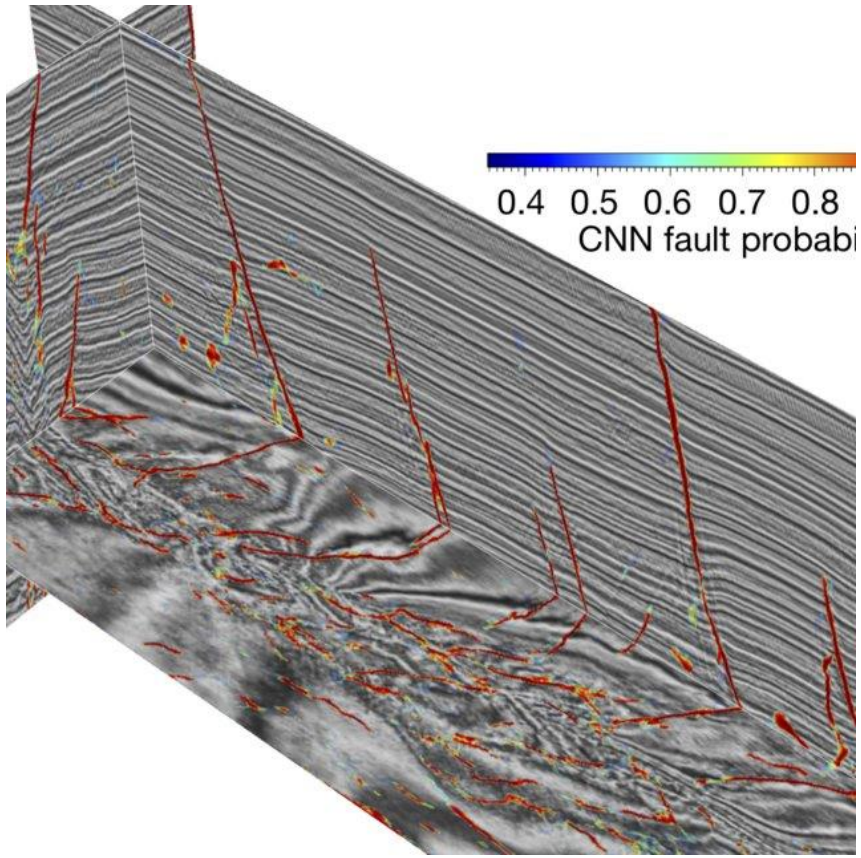
Focal Loss

- *Intuition:* downplay loss contribution of easy-to-classify examples while emphasizing that of harder training examples
- This leads to the optimization focusing more on getting the harder examples right
- Let p_t be pseudo-probability of ground-truth class predicted by the network.
- Focal loss works by adding a multiplicative factor of $(1 - p_t)^\gamma$ to the $\log p_t$ term in the regular cross entropy loss.
- γ is a hyperparameter. Larger values for γ result in more emphasis of harder training examples in the loss (with simultaneous de-emphasizing of easier ones)



Performance Metrics

- The performance metrics one uses should be appropriate for the application being faced
- Using the last example of anomaly/aberrant event detection, accuracy is a bad metric
- A model that is hard-coded to predict the majority class despite the input can still achieve very high accuracy (because it gets most predictions correct just by virtue of the sheer numbers of the majority class)



Performance Metrics

Precision and Recall

- Looking at both precision and recall for applications involving severe class imbalance can usually reveal a much more accurate picture of the model's underlying performance
- Taking our hard coded model that only predicts the majority class, it would score high on precision since it can predict every majority class sample as correct from within the test set
- However, it performs very badly on recall since it fails to detect any aberrant event
- The mean of the precision and recall, called the *F-score*, then reveals a true picture of the performance

$$F = \frac{2 \times p \times r}{p + r}$$

Performance Metrics

SSIM for Image Reconstruction

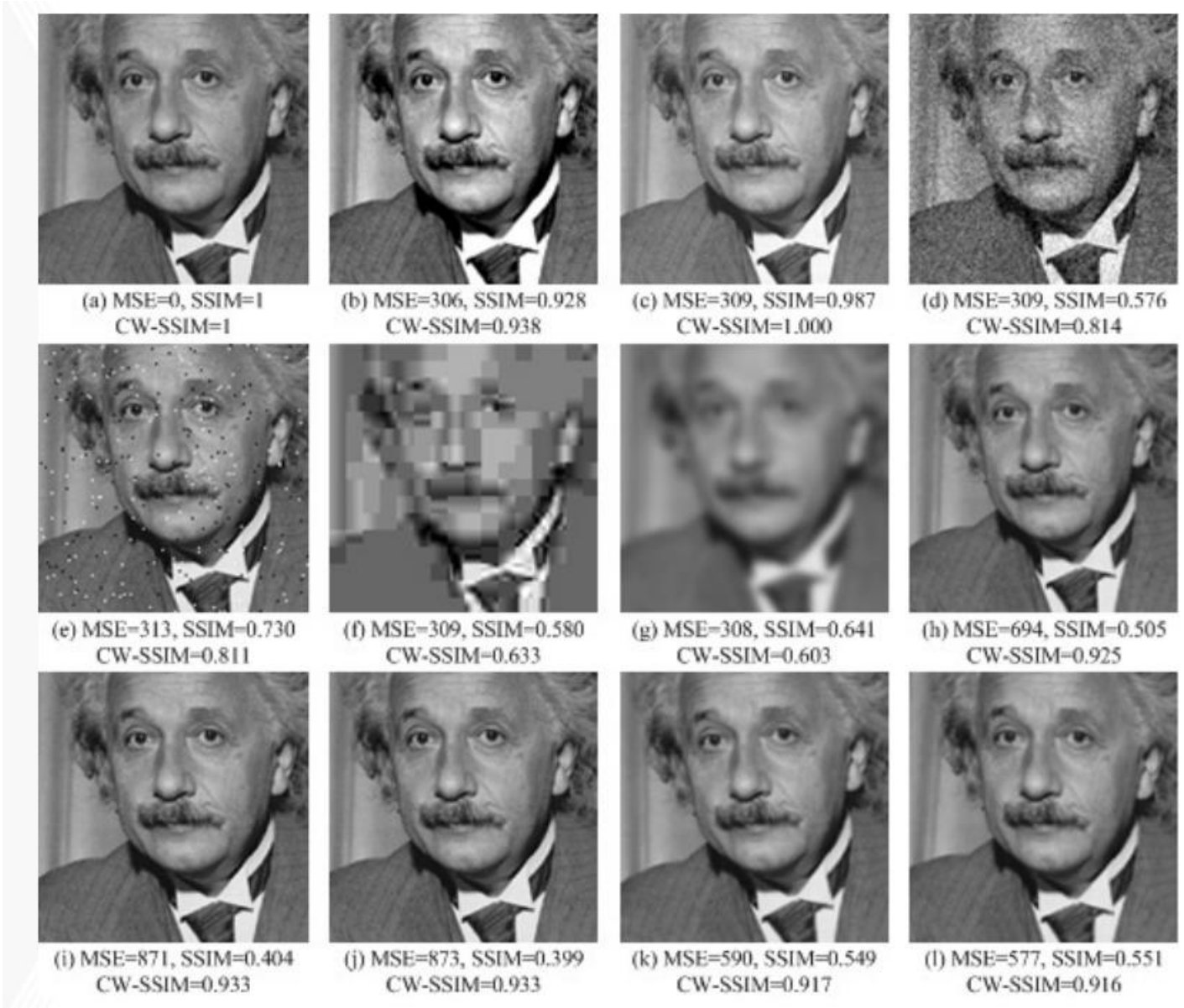
- Mean Square Error (MSE) and Peak Signal to Noise Ratio (PSNR) can at times be bad predictors of performance in certain dense prediction tasks like image denoising, super-resolution etc.
- Does not account for the spatial and structural characteristics of the image
- Structural Similarity-based Image quality Measure (SSIM) helps to navigate these situations much better
- Decomposes the image into luminance, contrast, and structure components and separately compares reference and query images for all three modalities before pooling the results into a single number

Performance Metrics

SSIM for Image Reconstruction

$$\text{MSE} = \left|\left|X - \hat{X}\right|\right|^2$$

- X is original image, $\hat{X}$ is reconstructed image. (a) MSE = 0 generally refers to the original image (or perfect reconstruction which is not achieved in practice)
- (b) – (g) MSE is around 310
- According to MSE, (g) = (b) and worse than (i)
- SSIM better represents the quality of reconstruction



Model Debugging

- A common occurrence with deep learning models is where they fail to deliver expected performances on training and/or test sets
- Sometimes, the poor performance might just be a consequence of the hard nature of the problem and indicate more research to be done in designing effective models, capturing better data etc.
- However, many times, causes of such problems stem from common issues that the practitioner failed to account for earlier
- Here we look at some of the most common issues that crop up in such situations and some of the ways to tackle them

Model Debugging

Low Model Capacity

- Model does not have the representational capacity to learn the mapping from input to output data
- Typical symptom includes underperformance on both training and test data (situation known as underfitting)
- Solution is to increase model capacity by increasing the depth of the model, number of learnable kernels etc.
- Make changes in small steps and monitor test and training performances after each change

Model Debugging

High Model Capacity

- When the model has very high representational capacity, it tends to perform well on training data, but poorly on test set—a situation commonly referred to as overfitting
- Solution is to steadily decrease model capacity (by either reducing the depth and/or the number of learnable kernels) and monitor training and validation error after each change

Model Debugging

Insufficient Data

- Sometimes, model underperformance (specifically on test set) may be caused by having insufficient data
- In this case, look for various data augmentation strategies allowing one to perform simple image transformations to multiply the quantity of available training data
- Common transformations include random cropping, horizontal and vertical flipping, random rotations, adding white gaussian noise, or any combination of these
- Transfer Learning from models pretrained on large datasets may also sometimes help
- It may also help to annotate new data if training data does not provide the complete picture of data distribution

Model Debugging

Too Few Data: Data Augmentations

Original Image

Rotation

Horizontal flip

Noise

Crop

Vertical Flip

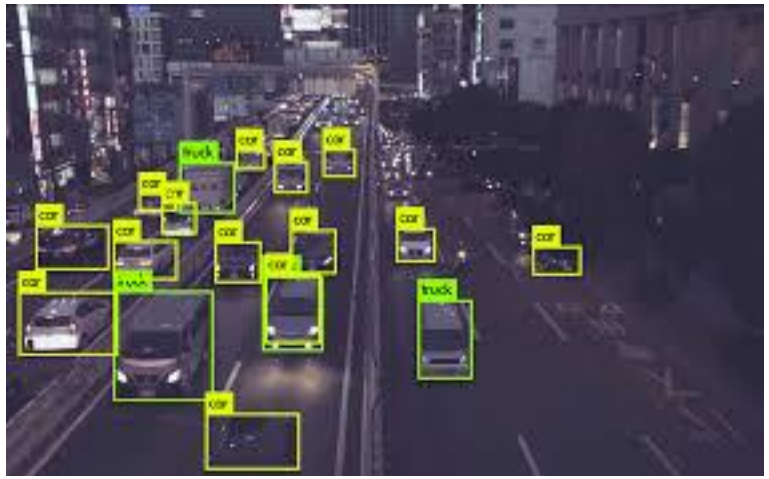
Blur

Transfer Learning

- It is known deep networks tend to learn common representations for various tasks in their earlier layers
- Can be exploited to transfer representations from networks trained on large datasets on one task (i.e., Image Classification on ImageNet) called the *source* to a different task called the *target* where quantity of labeled training data is much smaller
- Usually done by taking large pretrained network and then finetuning last layer (with all other layers frozen) on target dataset
- Helps avoid overfitting and trains a deployable network much faster
- Pretrained frozen backbone acts as a feature extractor while finetuned last layer acts to project the representations into the decision boundary for the target task
- Utility depends on how closely related the source and target datasets and/or tasks are

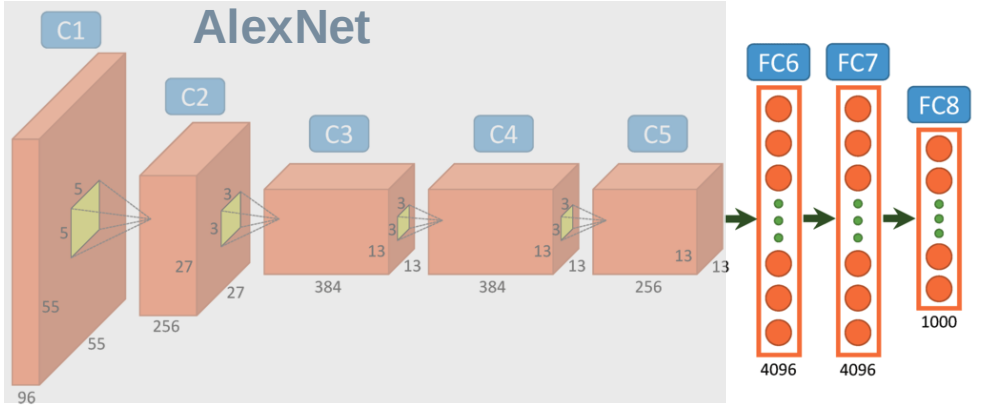
Transfer Learning

Source: https://gluon-cv.mxnet.io/



Source: https://www.move-lab.com/blog/tracking-things-in-object-detection-videos

Pretraining



Finetuning

Source: https://www.saagie.com/blog/object-detection-part1/

Visualizations

- Sometimes, numerical accuracy metrics can give a false sense of security regarding model performance
- It helps to visualize network outputs to qualitatively judge performance
- For image-based tasks, this involves looking at images superimposed with network predictions, visualizing sampled images (in case of dense predictions via VAEs, GANSs, Diffusion models) and reconstructions (e.g., image denoising applications)
- Visualizing the worst predictions made by the model sometimes reveals important trends e.g., model always makes mistakes on a particular kind of input
- Helps resolve issues by changing model/training data accordingly

Numerical Gradient Analysis

- Sometimes bugs occur with the way certain software packages implement their backpropagation methods or if one defined their own backpropagation code
- Good idea to check gradients computed via backpropagation to those computed numerically.
- One way to do this is by using **finite differences:**

$$f'(x) = \frac{f(x+\epsilon) - f(x)}{\epsilon},$$

where ϵ is a small constant.

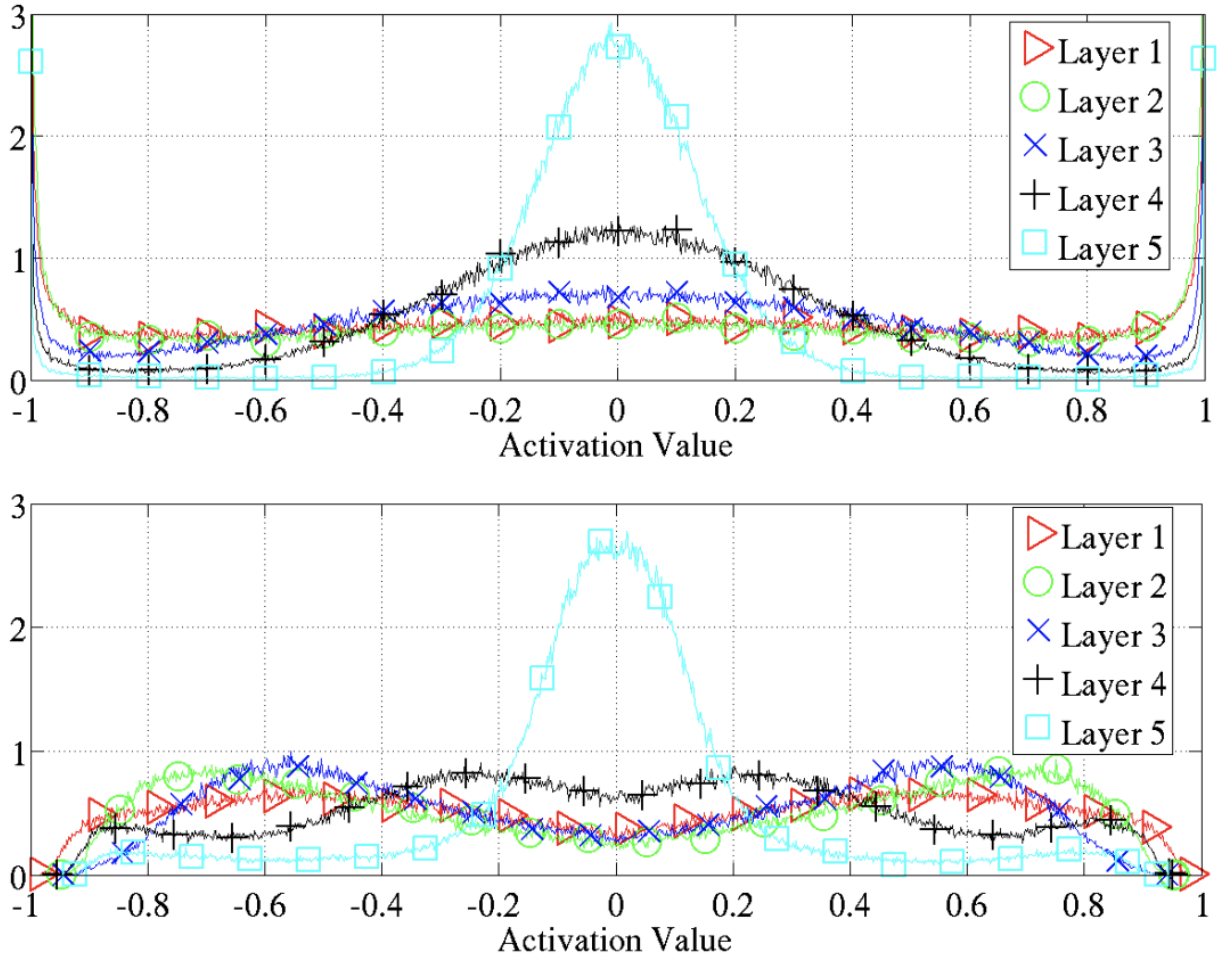
Visualize Activation Histograms

- Visualize histograms of network activations and gradients to reveal dead neurons and to reveal any underlying problem with optimization
- The pre-activation value of a neuron can tell us if the units saturate and how often they do.
- In a deep network, gradients may quickly grow or vanish; can hamper optimization
- Monitor magnitudes of parameter updates and compare that to the magnitudes of the parameter themselves
- Former should be something like 1% of the latter.
- Too large or too small would indicate problems with optimization

Visualize Activation Histograms

- Top: Hyperbolic tangent nonlinearity
- Bottom: Softplus nonlineraity

- For the hyperbolic tangent, activation values are saturated in the lower layers
- For softplus, the values are less saturated, specifically around (-0.6, -0.8) and (0.6, 0.8)



Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010.

Visualize Activation Histograms

Weight initialization: Xavier Initialization

- Xavier initialization ensures that the activation and gradient variances remain consistent when forward propagation with activations and backward propagation with gradients

$$W = U\left(\frac{\sqrt{6}}{\sqrt{n_{inp}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{inp}+n_{out}}}\right)$$

U = Uniform Distribution
n_{inp} = number of inputs to a layer
n_{out} = number of outputs from a layer

- More recently: Kaiming initialization for ReLU nonlinearities

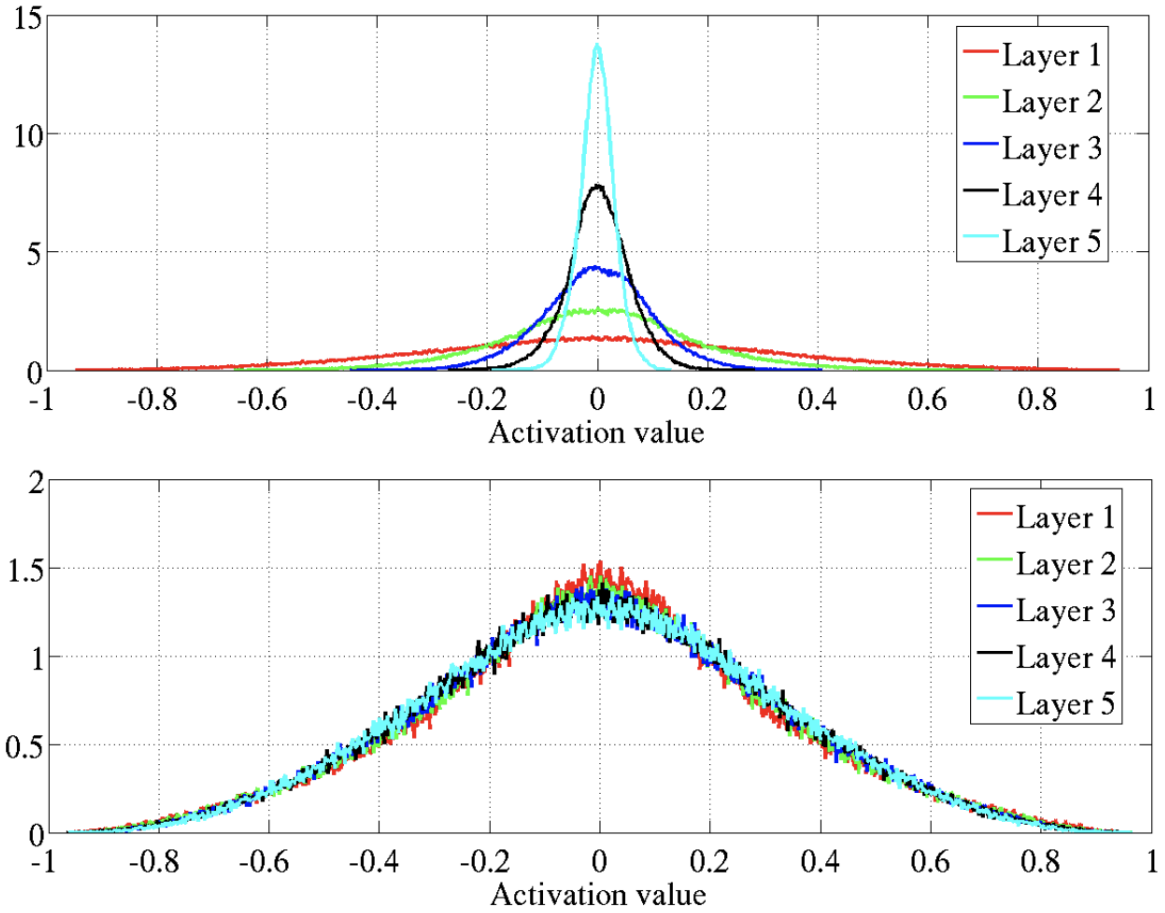


Figure 6: *Activation values normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized initialization (bottom). Top: 0-peak increases for higher layers.*

Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010.

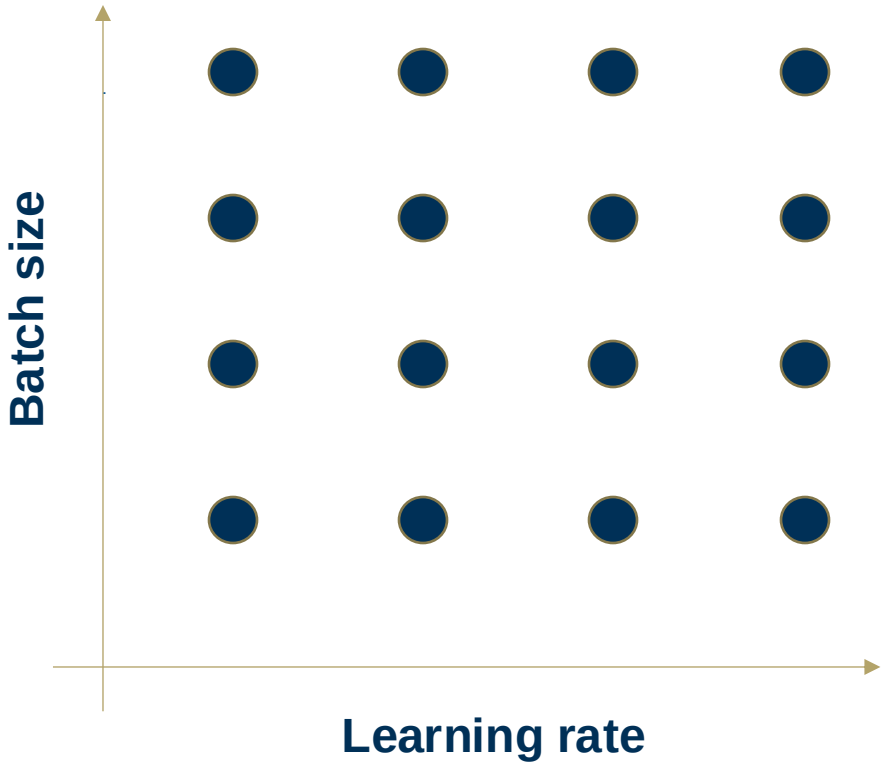
Tuning Hyperparameters

- Tuning hyperparameters like learning rate, batch size (for batch gradient descent), hidden layers, kernels etc., can at times lead to significant gains in network performance
- While one can always manually tune hyperparameters, there are more effective strategies available that automate the process: **grid search** and **random search**

Tuning Hyperparameters

Grid Search

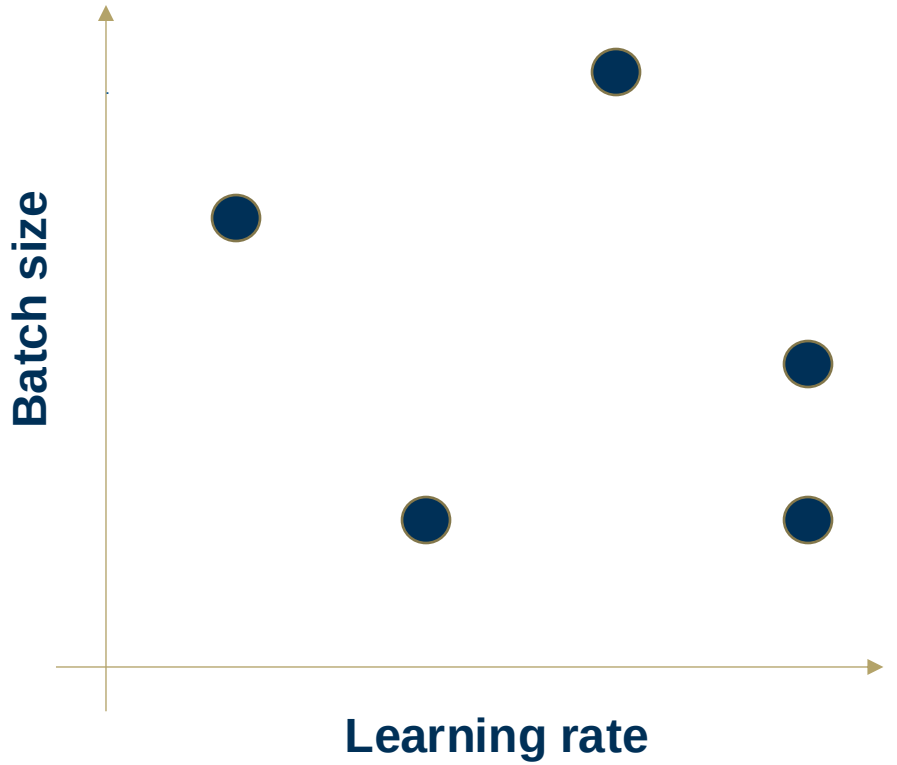
- User defines a finite set of values to take for each hyperparameter
- Software constructs a grid containing all possible configurations of hyperparameter values defined by the user
- Network training and inference procedure ran for each configuration to search for the best set of values
- **Con:** *Is computationally expensive!*



Tuning Hyperparameters

Random Search

- User defines a probability distribution for each hyperparameter setting
- Software randomly samples a configuration of hyperparameters each time a network is trained
- Tends to converge faster than grid search and is less computationally expensive



Logistics for 8803 Section

- Project: Biomarker Analysis on OLIVES dataset
- Logistics: Groups of 3
  - For a grad course, 20% project grade is roughly 40-50 hrs per person. Hence, project outputs must reflect 120-150 hrs of combined work
  - Grading is done on individual contributions
- Dataset download:
  - https://zenodo.org/records/7105232
  - Available on AI makerspace
  - **Hugging Face:** https://huggingface.co/datasets/gOLIVES/OLIVES_Dataset
- Goal: Detect Biomarkers given OCT images, and clinical labels
  - You are free to choose your train/test splits
  - You are free to choose your methods/evaluation
- Challenges you will be evaluated on
  - How will you handle multi-modal data (images, text etc.)?
  - What methods will you employ for evaluation?
  - **What analysis will you perform on results?**
  - How good are your results?

Logistics for 8803 Section

- Some resources:
  - Original paper: https://arxiv.org/pdf/2209.11195
  - 2023 SPS VIP competition: https://alregib.ece.gatech.edu/competitions/2023-vip-cup/
  - Competition structured dataset: https://zenodo.org/records/8040573
  - Example paper from the competition: https://arxiv.org/pdf/2310.14005
- Group members
  - Form groups here: https://docs.google.com/spreadsheets/d/1LriuGQuiF4RDL8DbOlg-ZXMCrhokEjFPCT0AlLJfOPs/edit?usp=sharing
- Deliverables: (More instructions when we approach the deadline)
  - 25 Oct: “Progress” Report (PDF), needs to have the following
    - Team members **(Representative team member uploads the PDF)**
    - Problem description
    - Proposed approach (no results necessary)
    - No specific format
  - 29 Nov: Term Project Paper (PDF)
    - Max 4 pages of content + unlimited pages for references
    - IEEE conference style paper format
    - Code on Github/HuggingFace

Appendix A: Notations

- x_i: a single feature
- $\boldsymbol{x}_i$: feature vector (a data sample)
- $\boldsymbol{x}_{:,i}$: feature vector of all data samples
- $\boldsymbol{X}$: matrix of feature vectors (dataset)
- N: number of data samples
- $\boldsymbol{W}$: weight matrix
- $\boldsymbol{b}$: bias vector
- $\boldsymbol{v}(t)$: first moment at time t
- $\boldsymbol{G}(t)$: second moment at time t
- $\boldsymbol{H}(\boldsymbol{\theta})$: Hessian matrix
- P: number of features in a feature vector
- α: learning rate
- Bold letter/symbol: vector
- Bold capital letters/symbol: matrix