Knowledge Transfer and Adaptation

Module Intro, Deep Learning Tips and Tricks

Antonio Carta

antonio.carta@unipi.it

Plan for Today

- Intro to KTA module
- Deep learning tips and tricks
- Practical lab with PyTorch

Module Outline

Knowledge Transfer and Adaptation

- Deep learning tools for KTA
- Transfer learning and domain adaptation
- Multi-Task learning
- Self-supervised training and large-scale models
- Meta-learning, metric learning, few-shot learning
- PyTorch labs

Resources

- Deep Learning Book by Goodfellow et al.
  - https://www.deeplearningbook.org/
  - Mostly for preliminary knowledge
- CS330: course on Multi-Task and Meta-Learning by Chelsea Finn
  - http://cs330.stanford.edu/
  - Related topics with recordings on youtube
- Academic literature

Module Objectives

How can we reuse the same Deep Neural Network for multiple tasks?

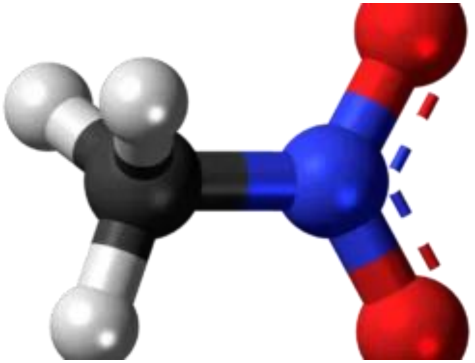
- Training a large model that can be reused
- Finetuning on downstream tasks
- Learning multiple tasks jointly, even with very small datasets

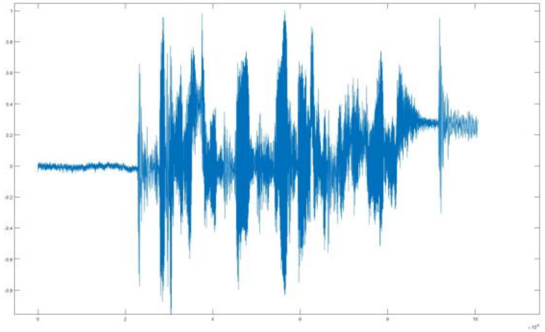
Deep Learning

Basic concepts and anatomy of vision models

Deep Learning – Intuition

- Classic ML models require **manual preprocessing**
- **Sensory input** (images, audio) and complex data (text, graphs)
  - require a lot of preprocessing to extract discriminative features
  - High dimensionality
  - Difficult to hardcode
  - Ideally we should learn it
- **Can we learn the feature extractor?**
  - This is the main problem solved by deep neural networks





Deep Learning

Intuition:

- Learning a deep neural network allows to automate feature extraction
- Stack multiple «layers» sequentially
    - Low layers capture low-level knowledge (e.g. texture)
    - High layers high-level knowledge (e.g. shapes, discriminative features)
    - Final layer should have simple and distinct clusters for each class (with supervised training)

Questions - When learning multiple tasks:

- Is it helpful to share layers?
- What happens if I train a network on a task and reuse it on a downstream task?
- How do I learn generic features that are helpful for a large class of tasks?

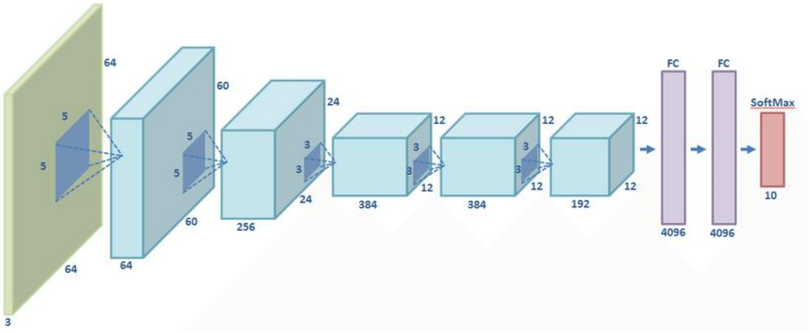
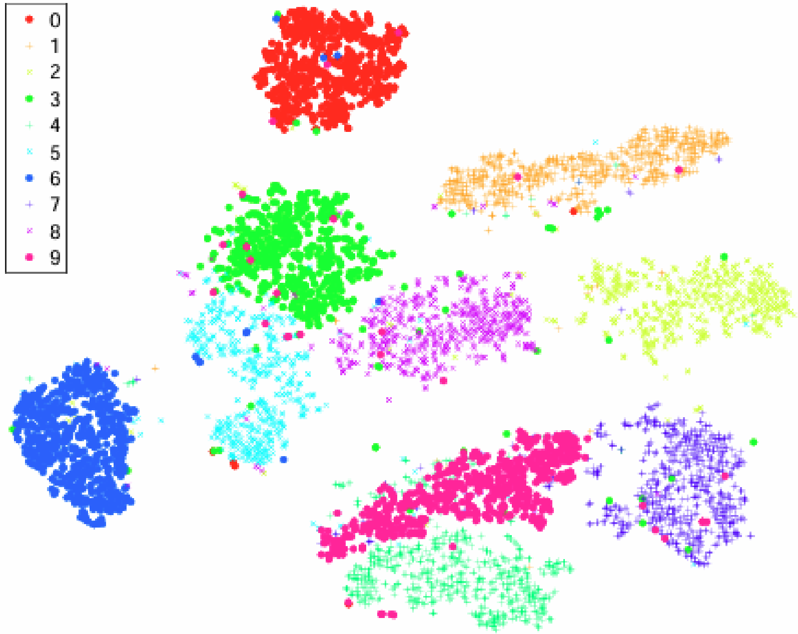


Image: https://www.researchgate.net/figure/Scheme-of-the-AlexNet-network-used_fig1_320052364

Embeddings

- Embeddings = latent representations
- Latent representations are the key novelty compared to «classic ML»
- Fundamental for transfer/sharing
- **IDEA**: we want embeddings that
  - Separate different, possibly unseen, classes
  - Are robust to domain drifts
  - Are robust to incremental training



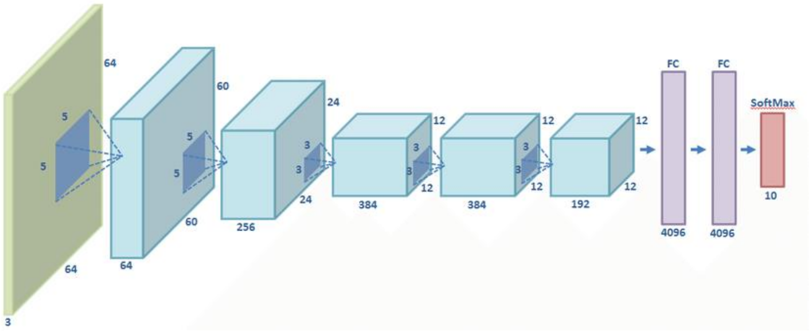
(a) Visualization by t-SNE.

Source: Visualizing Data using t-SNE

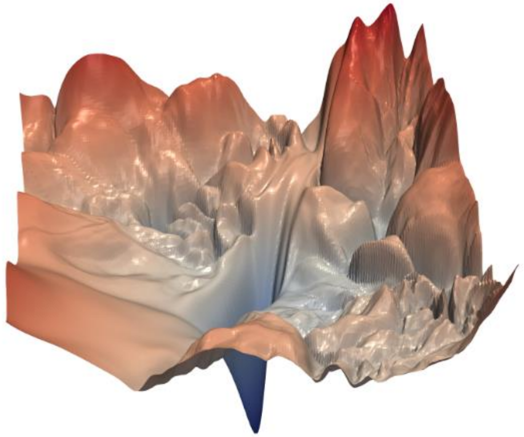
Learning a DNN

- **Algorithm**: stochastic gradient descent. At each step:
  - (forward) Compute output for current input
  - Compute loss
  - (backward) Compute gradients
  - Descent step

- **Forward** pass: input->output
  - Computes the output
- **Backward** pass: output->input
  - Computes the gradients



Loss landscape



DNN for image classification

- **CNN stack conv->BN->ReLU->pooling blocks**
  - Higher layers have a bigger receptive field
  - Pooling subsamples and reduces number of features
  - Typically number of channels increases for deeper layers

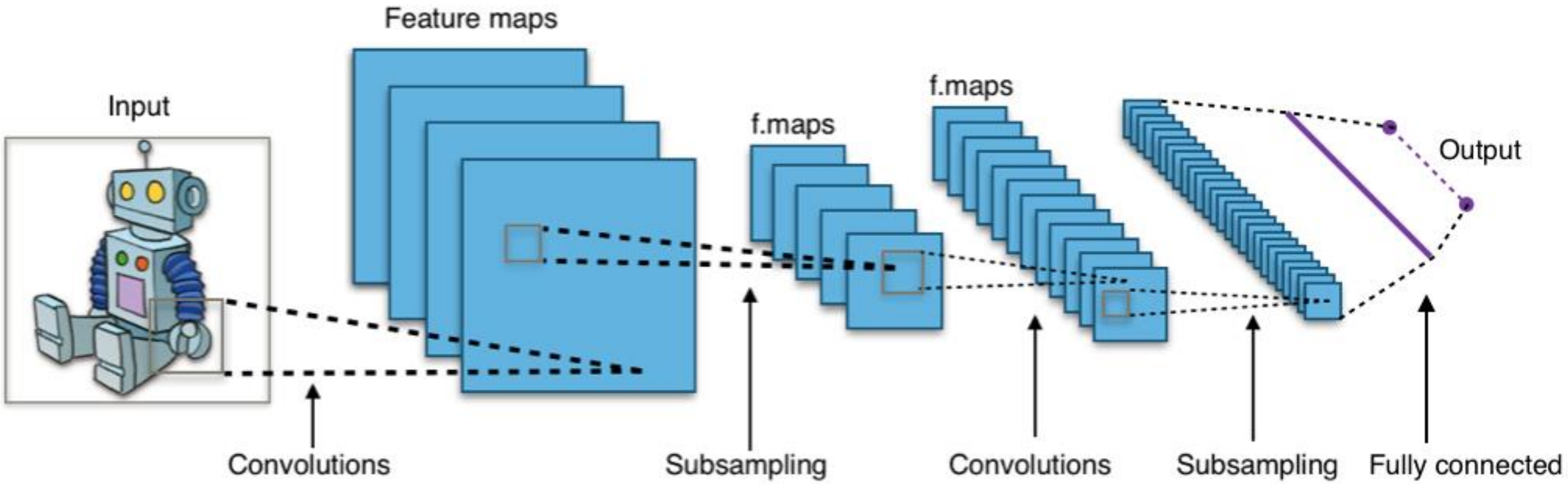


Image source: wikimedia

ResNet

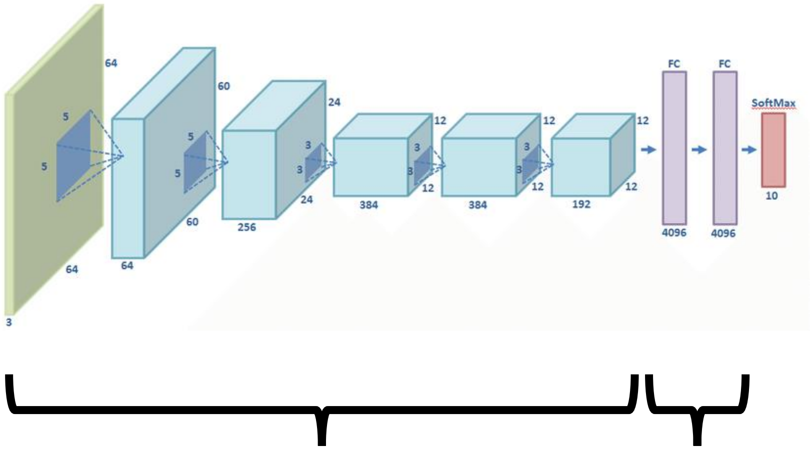
A popular example of computer vision model

- Convolutional network for image classification
- Convolutions
- BatchNorm
- pooling
- ReLU
- Residual connections
- Feedforward connections
- Softmax
- Crossentropy loss

CNN -> FNN -> classifier

Most networks have three parts:

- Convolutional blocks:
  - Conv->Relu->BatchNorm->Pooling
- FF layers
- Classifier



Convolutional blocks

Feedforward blocks

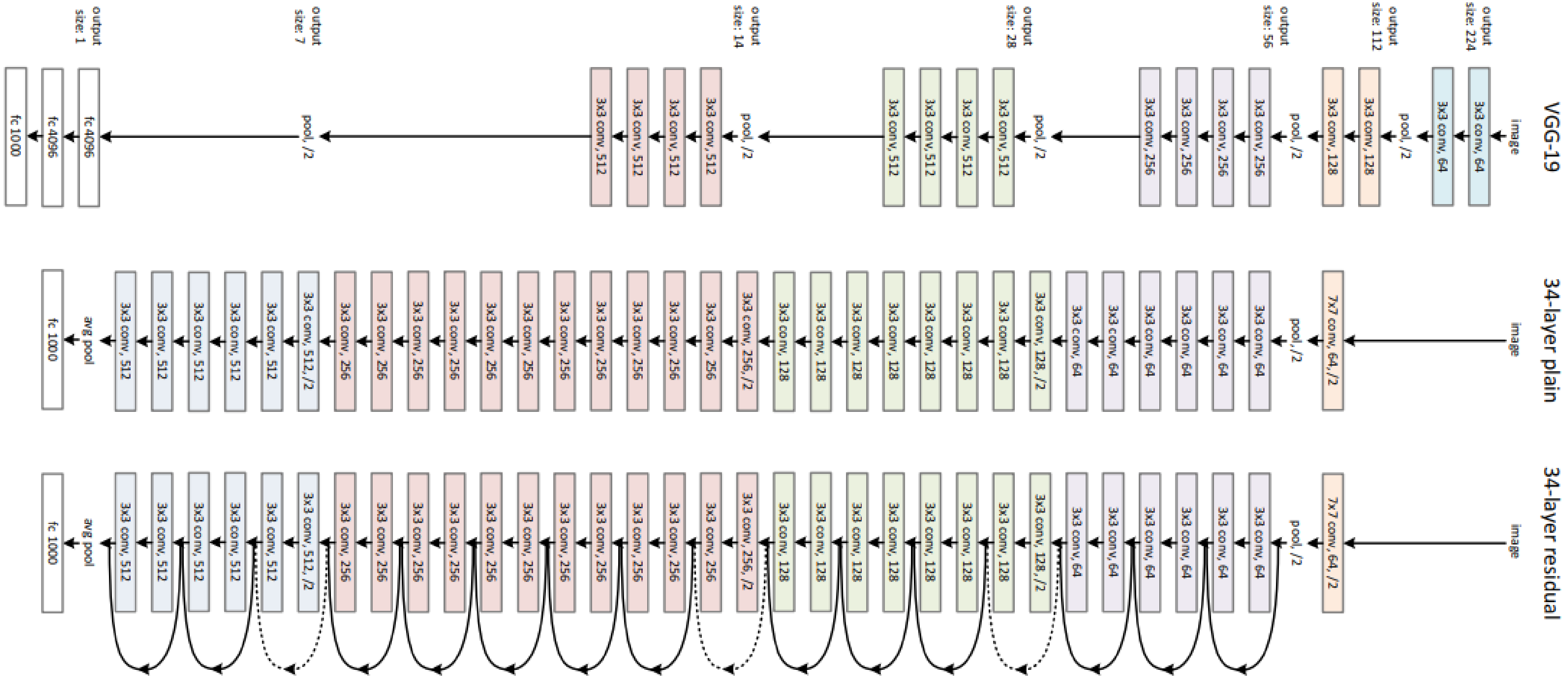
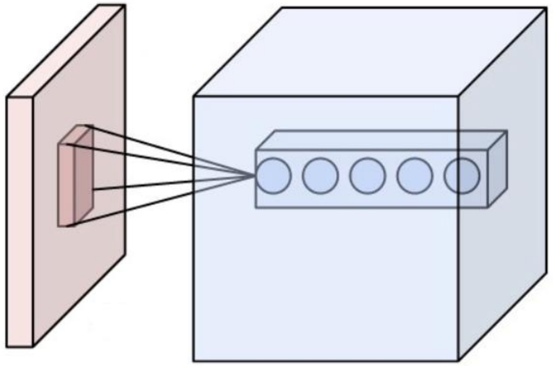


Image source: https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/

Convolution

- **Basic processing block for images**
  - Input: <C,W,H>
  - output: <C,W,H>
  - Parameters: padding, stride, kernel size
- **Number of dimensions depends on domain**
  - 1D for sequences such as text or audio
  - 3D for videos
- **Increase number of channels for deeper layers**
- **Reference**: for a reminder of the math see https://github.com/vdumoulin/conv_arithmetic



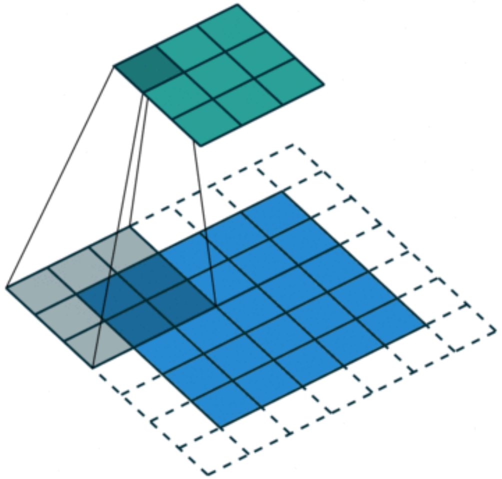


Image source: wikimedia

Batch Normalization

Problem: we know normalization is helpful. How can we normalize hidden representations?

- **Batch normalization** standardizes output of an hidden layer
- **Parameters**: γ and β are learned by backprop. μ and σ are running averages used during inference
- **Training behavior**:
  - Remove mean and std computed on the mini-batch
  - Scale and shift
  - Update mean and std running averages
- **Inference behavior** (it's different!):
  - Remove mean and std using running average
  - Scale and shift
  - *Inference is deterministic and order-independent*
- Many contrasting theories on why it's important. Not well understood theoretically yet.
- **Question: What happens when we change task? What if we train on multiple tasks?**

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Sergey Ioffe, Christian Szegedy

Dropout

Regularization method for DNN

- **INTUITION**: regularize via
  - Noise
  - Approximate ensembling
- **TRAINING**: (for each layer)
  - Sample a random mask. Each unit is masked with probability p
  - Mask units
- **INFERENCE**: (for each layer)
  - Scale weights by p
  - Use all the units (no masking)
- Inference is deterministic
  - In pytorch, remember to call `model.eval()`

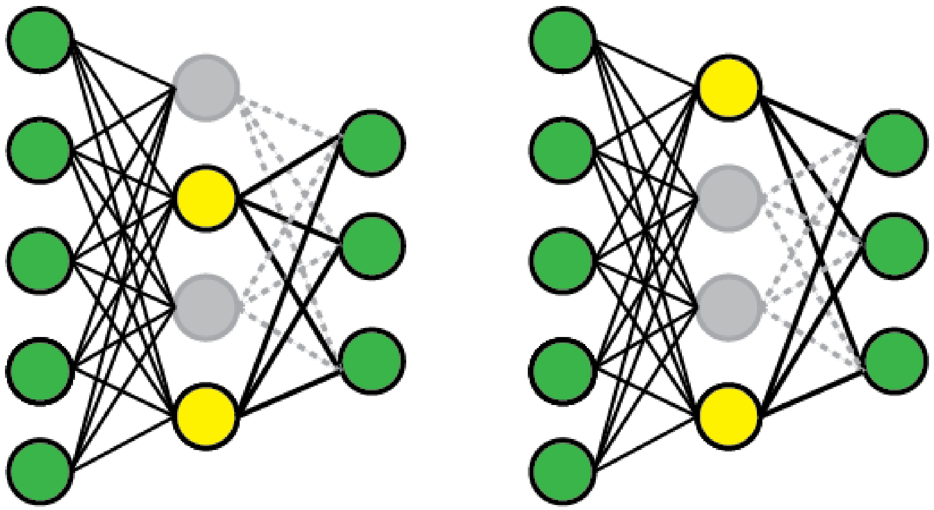
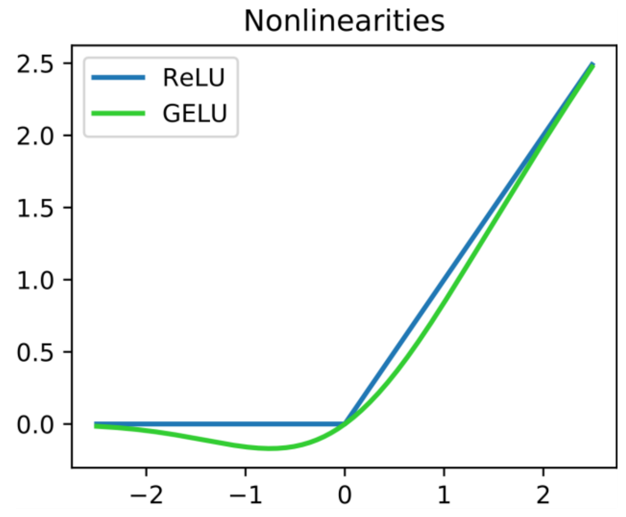


Image source: https://stats.stackexchange.com/questions/201569/what-is-the-difference-between-dropout-and-drop-connect/201578#201578

ReLU – Rectified Linear Units

- **Sigmoid and tanh saturate gradients**
  - A good init limits but doesn't solve the issue
- **ReLU**
  - It Has a better gradient flow
  - It is cheap to compute

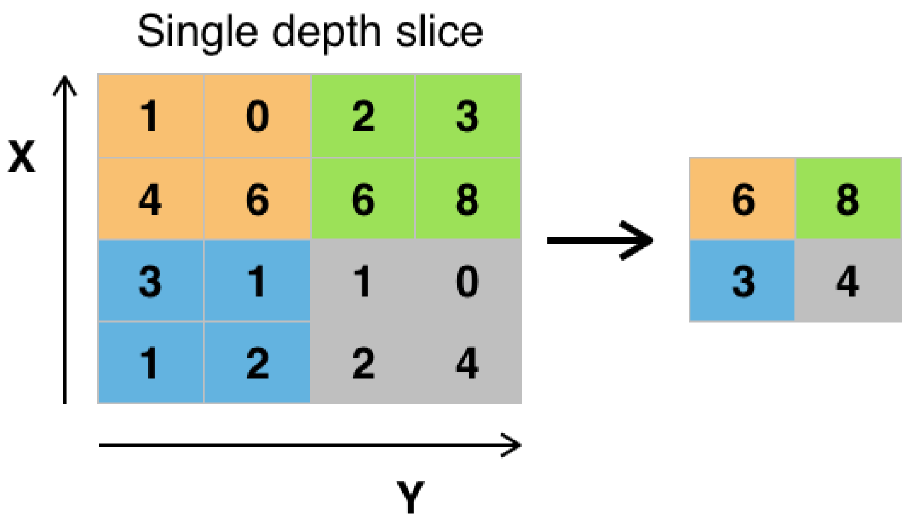


$$f(x) = x^+ = \max(0, x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases}$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{if } x < 0. \end{cases}$$

Pooling

- **Pooling is used for downsampling**
- **Aggregations**: max, mean, …
- **Reference**: for a reminder of the math see https://github.com/vdumoulin/conv_arithmetic



Example with max pooling

Image source: wikimedia

Residual Connection

- During the backward pass, gradient flow through
- **Problem**: gradient flow in DNN is bad.
  - Remember the vanishing/exploding gradients?
- Residual connections improve the gradient flow by allowing to skip layers
- $h^l = h^{l-1} + f(h^{l-1})$

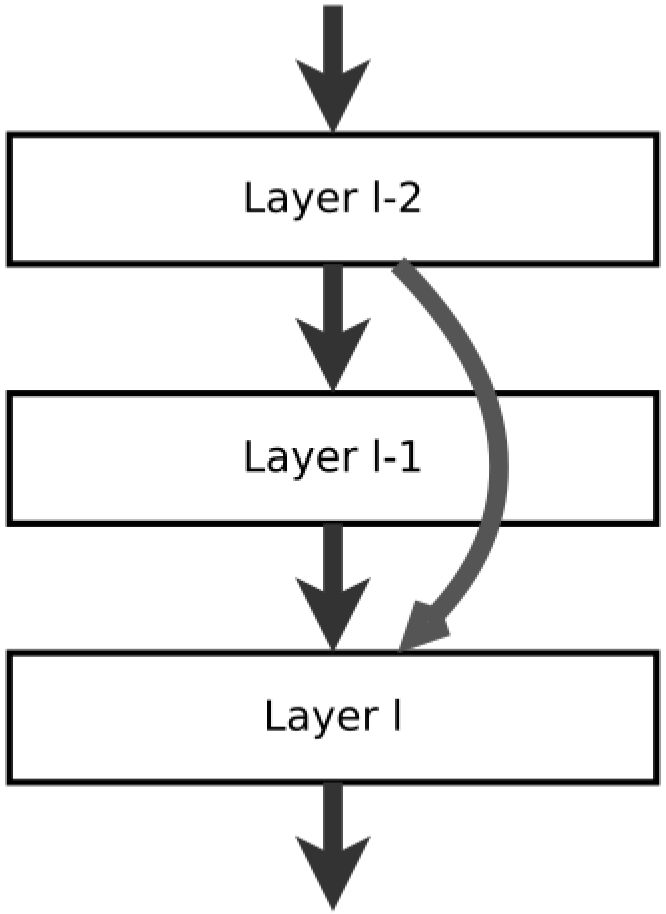


Image source: wikimedia

Classifier – Softmax and Crossentropy

- **Logits**: Output of the penultimate layer. Softmax input.
- **Softmax**: A smooth and differentiable argmax function
- **Crossentropy**: loss used for classification problems
- In practice, **we don't compute the softmax explicitly**
  - **Training**: Computing the crossentropy directly with the logits has better conditioning. We avoid the separate log and exponential operations
  - **Inference**: we only need to find the max logit. Softmax normalize units but doesn't change the ranking.
  - Always check the documentation to see if you need to use logits or softmax outputs (normalized probabilities)

softmax

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

crossentropy

$$H(P,Q) = -\sum_{x\in\mathcal{X}} p(x)\,\log q(x)$$

Tips and Tricks

Hyperparameter Optimization

- Full hyperparameter optimization is often unfeasible
- Most hyperparameters are not important. Some are very important (learning rate).
- Interaction between hyperparameters
  - Example: changing the batch size change the number of iterations per epoch
- **DNN Tuning guidelines** by Google Research team: https://github.com/google-research/tuning_playbook
- As a general rule, start from the best model in the literature and improve on it.

Model Initialization

- **Model initialization helps to stabilize the first epochs of training**
  - As a general rule, networks don't recover from bad initializations. We will see some results.
- Ignoring the weights initialization is a common error
  - **Symptoms**: training instability, network doesn't converge
- Always check your init: https://pytorch.org/docs/stable/nn.init.html

Learning Rate Scheduling

- Best results in computer vision are often a combination of basic SGD+momentum with learning rate scheduling
- pytorch: https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate
- General rule:
  - Start with a higher lr
  - Decrease slowly
- If you can, start from hyperparameters used for similar problems/datasets/architectures

Early Stopping + Model checkpointing

- Training is often unstable
- **Early stopping**:
  - Periodically evaluate on a validation set
  - If valid-score does not improve for `` `patience` `` epochs, stop training
- **Model Checkpointing**:
  - Periodically evaluate on a validation set
  - If valid-score improves, save a model checkpointing
  - After training, load the best checkpoint and use it for inference

Tools you may need

timm

- PyTorch library for computer vision architectures
- Pretrained models and state-of-the-art architectures
- Very comprehensive, even for recent models
- https://github.com/huggingface/pytorch-image-models
- Alternative: torchvision
  - Official pytorch repo
  - Less architectures

Computer Vision - Data augmentations

- Albumentations - https://github.com/albumentations-team/albumentations
- Kornia - https://github.com/kornia/kornia
- Faster dataloader: https://github.com/libffcv/ffcv

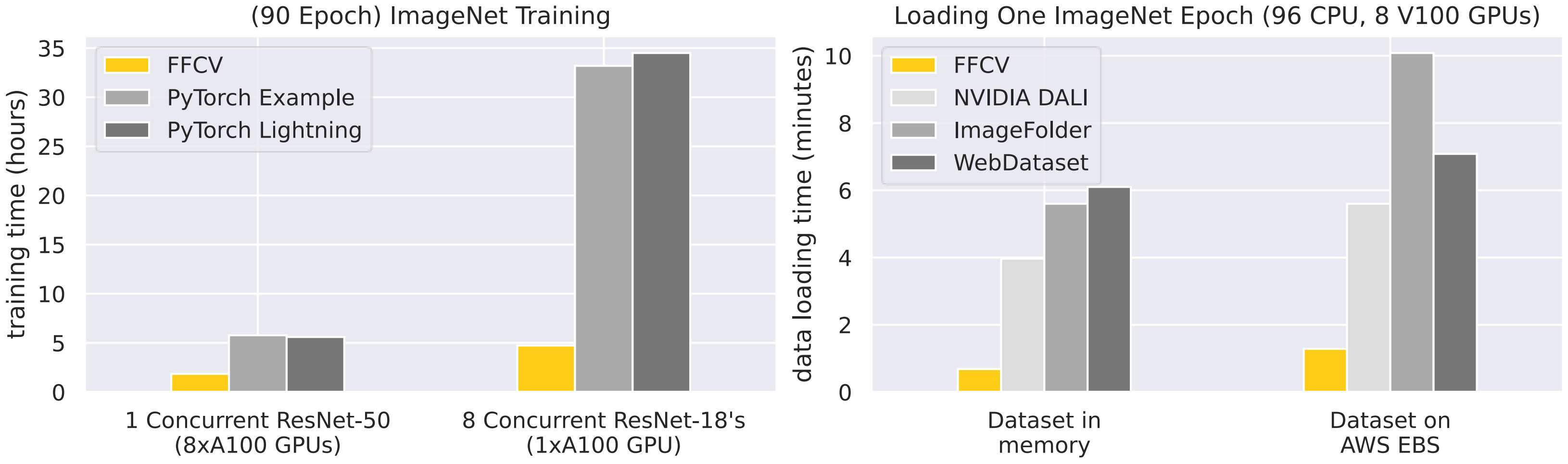


Image: https://github.com/libffcv/ffcv

Training frameworks

- Huggingface/timm: https://huggingface.co/docs/timm/training_script
  - Huggingface also has libraries for NLP
- fastai - https://github.com/fastai/fastai
- Pytorch lightning: https://www.pytorchlightning.ai/
  - Good support for distributed training and mixed precision
- Avalanche: https://avalanche.continualai.org/
  - We will use Avalanche for CL methods

Logging and Visualization

- Logging tools:
  - Many different companies: weights and biases, cometml, clearml…
  - Suggested: tensorboard. Everything is local. Easily integrated everywhere.
- Hiplot for visualizing model selection results: https://pypi.org/project/hiplot/

Conclusion

Notebook

- Deep learning notebook
- Dependency: Avalanche 0.5.0
  - pip install avalanche-lib==0.5.0
  - If you have problems install it in a new environment

Take-Home Messages

- Deep learning model are designed to extract high-level features from low-level sensory inputs (e.g. high-dimensional images)
- learned latent representations can also be reused, opening up many new applications

Next-Lecture

Multi-Task learning

- Definition
- Design choices
- challenges