Lecture 11: Midterm 1 review

CS 182/282A (“Deep Learning”)

2022/02/28

Today's lecture

- No lecture Wednesday! You have midterm 1 instead
- No new content today — just a collection of past slides and relevant questions that can be answered by reviewing the course content so far
- This is meant to get you started, if you haven't started yet, on your studying
    - But hopefully you have started studying…
- *Not* a substitute for studying on your own! E.g., doing the past/practice midterm
- We may not get through all slides, but you can review the rest on your own

Midterm 1 logistics

- For all students with standard accommodations (if you’re not sure, this is you):
  - Midterm time is 7-9pm — arrive promptly at 7pm, we begin promptly at 7:10
  - All 182 students, and 282A students with last names starting with Q-Z: Pimentel 1
  - 282A students with last names starting with A-P: Evans 60
- Students with DSP accommodations: make a private Piazza post if you have not yet received your specific logistics
- One double sided 8.5x11in cheat sheet is permitted

Introduction

The underlying themes

End-to-end learning and scaling

- Deep learning acquires representations by using high capacity models and lots of data, without requiring engineering features or representations
- We don’t need to know what the good features are, we can have the model figure it out from the data
  - This results in better performance, because when representations are learned end-to-end, they are better tailored to the current task
- Scaling is the ability of an algorithm to work better as more data and model capacity are added
  - Deep learning methods are really good at scaling

The underlying themes

Inductive bias vs. learning

- Inductive bias vs. learning can be thought of as “nature vs. nurture”: getting performance from designer insight vs. from data, respectively
- Inductive bias: the knowledge we build into the model to make it learn effectively
    - All such knowledge is “bias” in the sense that it makes some solutions more likely and some less likely
    - We can never fully get rid of the need for inductive biases!
- A common theme in deep learning for many applications:
  deep neural network models overtake the next best model after we figure out the right inductive biases for that application

ML review

The machine learning method

(or, at least, the deep learning method)

1. Define your model — which neural network, what does it output, …
2. Define your loss function — which parameters are good vs. bad?
3. Define your optimizer — how do we find good parameters?
4. Run it on a big GPU

Probabilistic models

- Often, it makes more sense to have the model predict output probabilities, rather than the outputs themselves
    - This can better capture uncertainty and also makes the learning process easier
- So instead of the model output $f_\theta(\mathbf{x})$ being a single y, it will instead be an entire distribution over all possible y
    - E.g., for digit recognition, the output will be 10 numbers between 0 and 1 that sum to 1
    - How is this done, mathematically and practically (in code)?

Negative log likelihood loss

- How is the negative log likelihood loss function motivated from maximum likelihood estimation?
- Why is it oftentimes called the cross-entropy loss function?
- What is another example of negative log likelihood loss for a different problem?
- How is this loss implemented practically (in code)?
- What is an example of another loss function that isn't negative log likelihood?

Gradient based optimization

- Deep learning relies on iterative optimization to find good parameters
  - Starting from an initial “guess”, continually refine that guess until we are satisfied with our final answer
- By far the most commonly used set of iterative optimization techniques in deep learning is (first order) gradient based optimization and variants thereof
  - Basically, move the parameters in the direction of the negative gradient of the average loss: $\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{N} \sum_{i=1}^{N} \ell(\theta; \mathbf{x}_i, y_i)$

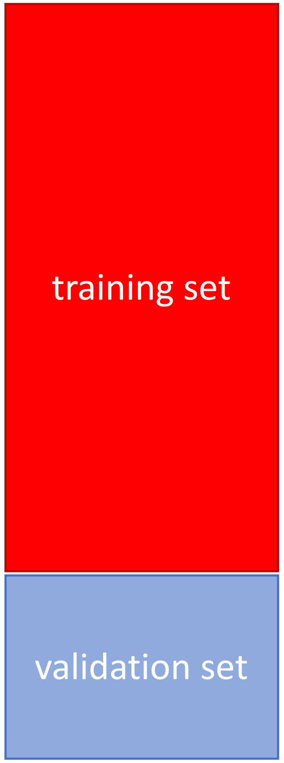
A small example: logistic regression

The “linear neural network”, if we’re being weird

- Given $\mathbf{x} \in \mathbb{R}^d$, define $f_\theta(\mathbf{x}) = \theta^\top \mathbf{x}$, where θ is a $d \times K$ matrix
- Then, for class $c \in \{0, \ldots, K-1\}$, we have $p_\theta(y = c \mid \mathbf{x}) = \text{softmax}(f_\theta(\mathbf{x}))_c$
- Loss function: $\ell(\theta; \mathbf{x}, y) = -\log p_\theta(y \mid \mathbf{x})$
- Optimization: $\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{N} \sum_{i=1}^{N} \ell(\theta; \mathbf{x}_i, y_i)$

The machine learning workflow

1. Define your model
2. Define your loss function
3. Define your optimizer
4. Run it on a big GPU



1. Learn θ on the training set
   - if the training loss is not low enough…
   - you are underfitting! increase model capacity, improve optimizer, …
   - and go back to step 1
2. Measure loss on the validation set
   - if the training loss is much smaller than the validation loss…
   - you are overfitting! decrease model capacity, collect more data, …
   - and go back to step 1
3. Not overfitting or underfitting? You're done

True risk and empirical risk

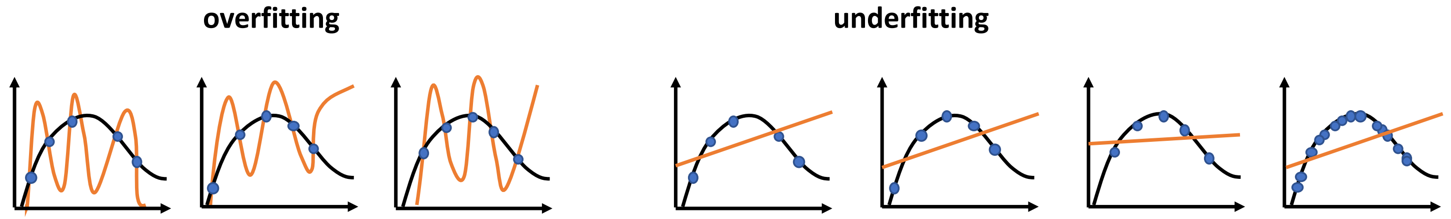
- Risk is defined as expected loss: $R(\theta) = \mathbb{E}[\ell(\theta; \mathbf{x}, y)]$
  - This is sometimes called true risk to distinguish from empirical risk below
- Empirical risk is the average loss on the training set: $\hat{R}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(\theta; \mathbf{x}_i, y_i)$
  - Supervised learning is oftentimes empirical risk minimization (ERM)
- Why are empirical and true risk different? How do we fix this?
- What do we call differences between the empirical and true risk?

Regularization

- Broadly speaking, a regularizer is anything we add to the loss function, optimization, and/or model that does not depend on the data
  - We add it to encode some prior belief about what a “good” model looks like — hence, it is a form of inductive bias
- A classic example is ℓ_2-regularization, which adds $\lambda||\theta||_2^2$ to the loss function
  - Why is this a good idea? Smaller parameters typically correspond to smoother functions that change less dramatically as the input changes
  - In classification, this is often (somewhat erroneously) referred to as weight decay

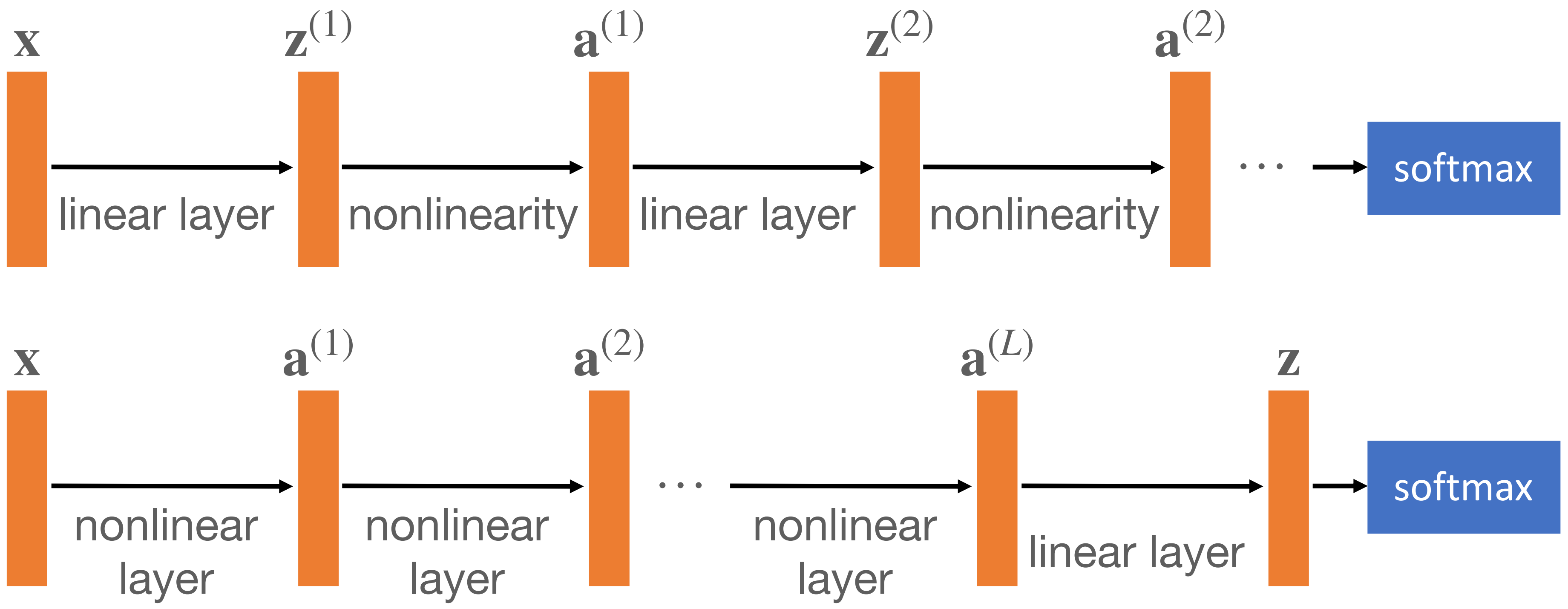
Bias and variance

- How are bias and variance defined, intuitively and mathematically?
- How do these concepts relate to overfitting and underfitting?
- How do we derive the bias-variance decomposition?
$\mathbb{E}[(f_{\theta(\mathcal{D})}(\mathbf{x}') - y')^2] = (\bar{f}(\mathbf{x}') - f(\mathbf{x}'))^2 + \mathbb{E}[(f_{\theta(\mathcal{D})}(\mathbf{x}') - \bar{f}(\mathbf{x}'))^2] + \sigma^2$



Neural network basics

Neural networks



Backpropagation

- First, we perform a forward pass and cache all the intermediate $\mathbf{z}^{(l)}, \mathbf{a}^{(l)}$
- Then, we work our way backwards to compute all the $\nabla_{\mathbf{W}^{(l)}}\ell,\ \nabla_{\mathbf{b}^{(l)}}\ell$
  - Going backwards allows us to reuse gradients that have already been computed
  - It also results in matrix-vector product computations, which are far more efficient than matrix-matrix product computations
- After all the gradients have been computed, we are ready to take a gradient step
- How does this compare to the method of finite differences?

Automatic differentiation

- Why do we care about autodiff when we already implemented backpropagation for our simple neural network model?
- What is the difference between forward mode and reverse mode autodiff?
    - Which one is more useful for deep learning and why?
- What role do computation graphs play in autodiff?
- Go through the end of Matt’s slides as well as the coding example from lecture, make sure you understand the high level ideas

Neural network building blocks

Input standardization

- Input standardization is carried out for each dimension of the input separately
- For each training input, for each dimension d, we subtract the mean $\mu_d = \frac{1}{N}\sum_{i=1}^{N} x_d$ and divide by $\sigma_d = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_d - \mu_d)^2}$
- There are some variations on this, e.g., this is usually done per channel for image inputs rather than per dimension
    - And for discrete inputs, such as in language, this is typically not done at all

Batch normalization (BN)

- BN refers to normalizing $\mathbf{z}^{(l)}$ or $\mathbf{a}^{(l)}$ using statistics computed from the mini batch
- We can think of this as putting a BN “layer” either before or after the nonlinearity
- The BN layer also includes learnable scale and shift parameters
- Models with BN layers operate in two different modes: “train” vs. “test” or “eval”
  - Train mode: compute statistics using the mini batch
  - Eval mode: use an exponential moving average of the statistics computed during train time

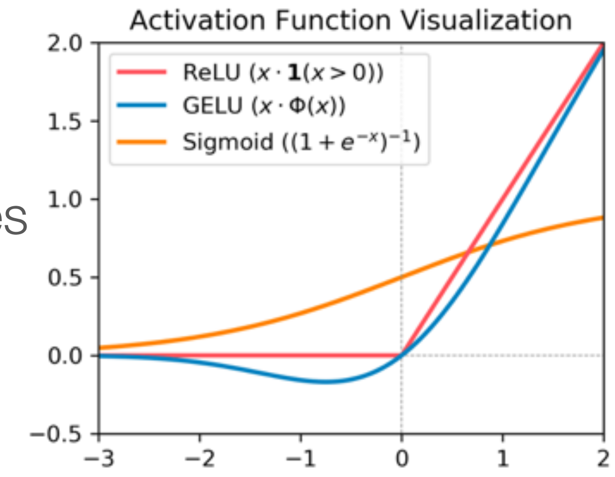
Layer normalization (LN)

And comparing BN to LN

- LN is basically the “transpose” of BN: compute the mean and standard deviation of $\mathbf{z}^{(l)}$ across the feature dimensions, rather than per dimension
    - Now, each data point will have different normalization statistics, but these statistics are shared across dimensions
- How is LN different from BN? How is it similar?
- What are the tradeoffs of BN vs. LN?
- What are their shared benefits or downsides?

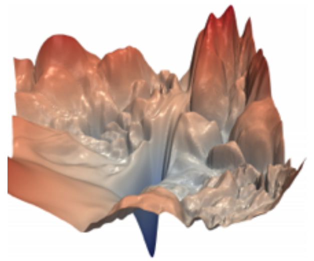
Comparing different common nonlinearities

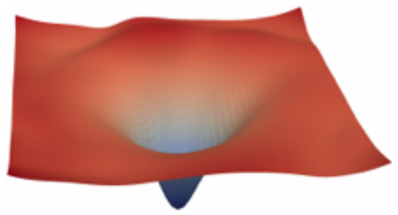
- Both sigmoid and ReLU are non negative and monotonically non decreasing
- sigmoid and GELU are smooth, which is sometimes important from an optimization perspective
- sigmoid is historically an important activation but is rarely the only nonlinearity used in today's neural networks



Skip connections

- Basically every state-of-the-art neural network uses skip connections
- Very simple high level idea: $\mathbf{a}^{(l)} = \sigma(\mathbf{z}^{(l)}) + \mathbf{a}^{(l-1)}$, rather than just $\mathbf{a}^{(l)} = \sigma(\mathbf{z}^{(l)})$
- This idea was popularized by residual convolutional networks (ResNets)
    - Allowed for training much deeper, more performant models
- The loss "landscape" of neural networks with residual connections looks much nicer


Li et al, NIPS '18


Li et al, NIPS '18

Weight initialization

A thought exercise

- What should we initialize our neural network parameters (weights) to? This question is less important with the advent of BN and LN, but it is still interesting to think about
- If $x_j \sim \mathcal{N}(0, 1)$ in each dimension j, and we initialize each $\mathbf{W}^{(1)}_{ij} \sim \mathcal{N}(0, \sigma^2_W)\ldots$
  - …then we get $\mathbb{E}[z_i^2] = \sum_j \mathbb{E}[(\mathbf{W}^{(1)}_{ij})^2]\mathbb{E}[x_j^2] = d\sigma^2_W$
- Therefore, picking $\sigma^2_W = \dfrac{1}{d}$ gives us outputs similar in magnitude to the inputs
  - We can do this at every linear layer, i.e., initialize each $\mathbf{W}^{(l)}$ with variance inversely proportional to the input dimensionality to that layer
- In practice: it's slightly more complicated, but it's done for you by deep learning libraries

Dropout

Correction: this is actually DropConnect

- Often, dropout (or DropConnect, or drop-*) is applied to our model during training
- DropConnect is very simple: randomly zero out some fraction p of the $\mathbf{W}_{ij}$
    - Can implement as element wise multiplication of each $\mathbf{W}^{(l)}$ with a boolean mask
- Drop-* builds redundancies into the model, such that it doesn't rely too much on any particular "pathways" through the network
    - Yet another example of inductive biases at work!
- Some care should be taken to make training vs. test output magnitudes consistent

Data augmentations, briefly

We'll talk more about this topic later in the course

- For some problems, data augmentations are an indispensable part of training
  - E.g., for image classification: we apply random flips and crops to the images
- This is useful for encoding invariances, e.g., flipping and cropping do not change the image class
  - Another inductive bias!
- For some domains, such as natural language, it is harder to come up with good data augmentation schemes

https://neptune.ai/blog/data-augmentation-in-python

Neural network ensembles

- If you have enough compute, training multiple neural networks is often useful
- Same concept as bagging for other machine learning models — an ensemble of models reduces variance and combats overfitting
    - Turns out, also very good at uncertainty quantification
- In theory: create different bootstrap samples of the dataset to train the models
    - In practice for neural networks: just train them all on all of the data
- In theory: when predicting, average all of their output probabilities together
    - In practice: just take a majority vote

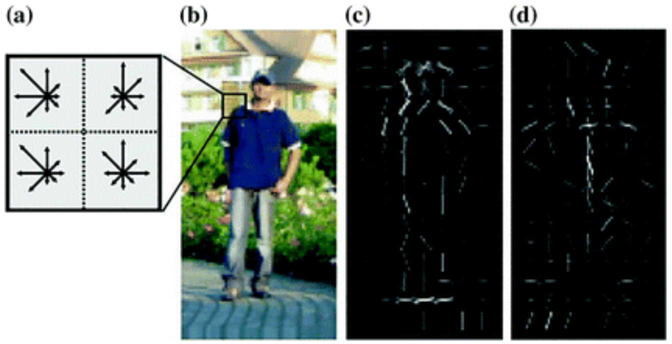
Hyperparameter optimization

- Typically, tuning hyperparameters goes from “coarse to fine”
  - E.g., first find the right order of magnitude for the learning rate, then zero in
- Hyperparameter search can be done with randomly sampled values or in a grid
- When grid searching, it is standard to space values evenly in log space
- For example, to cover [0.001, 0.01] approximately evenly, use:
  - [0.001, 0.003, 0.01] if grid searching with three values
  - [0.001, 0.002, 0.005, 0.01] if grid searching with four values

Convolutional networks

The key idea behind conv nets

- The key idea behind reducing the massive number of parameters is the observation that many useful image features are local
    - E.g., edge information, used by many once-popular hand designed features
- We won't go so far as to hand design the features, but we will place limits on the features that can be learned via the architecture
    - Inductive biases at work
- Useful nonlocal information can also be captured by stacking multiple convolutions together

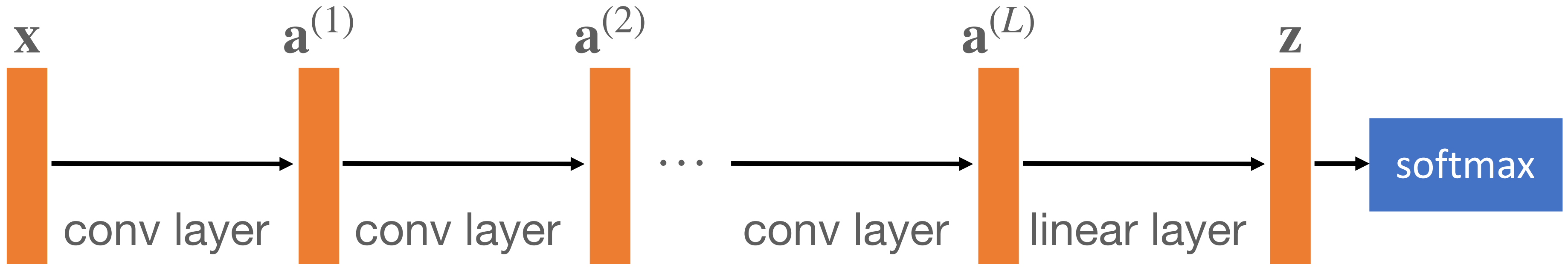


The convolution layer

- Convolution is performed with a filter — a tensor with dimensions $[K, K, O, I]$ (e.g., $[3, 3, 4, 3]$) — and a O-dimensional bias term
- (2D) convolutions take in an input of size $[I, H, W]$ (or $[H, W, I]$, depending on the convention) and output a tensor of size $[O, H', W']$
    - What are H' and W'?
- Because the output has similar dimensions, we can stack convolutions on top of each other to make deep convolutional networks

Convolutional networks

- A simple convolutional network repeats the convolution $\rightarrow$ BN $\rightarrow$ ReLU recipe L times to process the input image into a representation $\mathbf{a}^{(L)}$
- We flatten or pool $\mathbf{a}^{(L)}$ into a one dimensional vector, pass it through one or more linear layers, and then (for classification) get our final probabilities with softmax



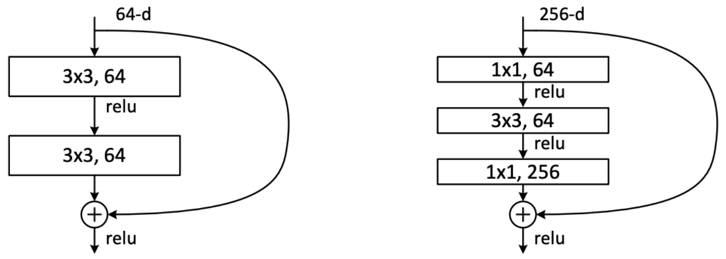
ImageNet image classification

- ImageNet consists of $224 \times 224 \times 3$ images evenly covering 1000 classes
  - There are 1.2M training images and 50000 evaluation images
- ImageNet-22K is a larger version of ImageNet (roughly $10 \times$ larger) with 22000 classes, increasingly used these days due to expanding compute budgets
- It is common for computer vision applications to start from a network pretrained on ImageNet

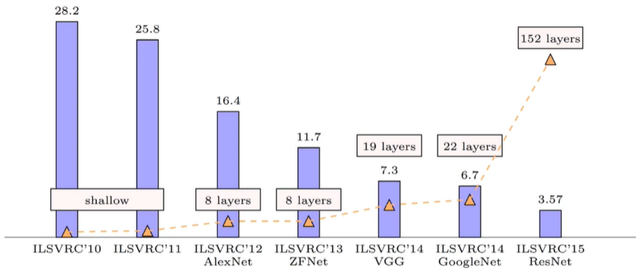
Skip connections in convolutional networks

He et al, 2015

- Recall the general idea behind skip connections: $\mathbf{a}^{(l)} = \sigma(\mathbf{z}^{(l)}) + \mathbf{a}^{(l-1)}$
- This idea was popularized by residual networks (ResNets), a convolutional architecture that implemented the idea slightly differently (and in two ways)
- This allowed for better training of deeper networks, which are more performant



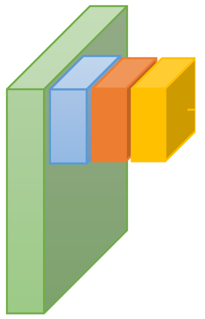
He et al, 2015

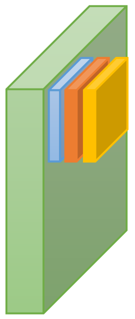


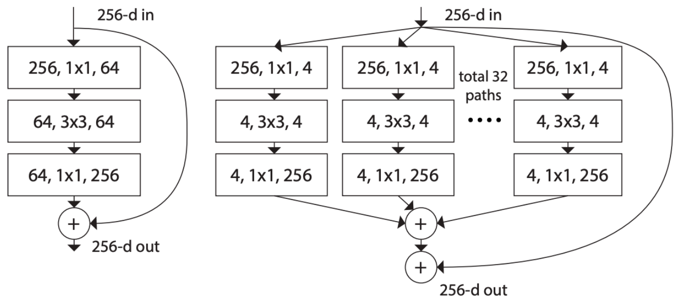
Depth wise (or grouped) convolutions

E.g., Xie et al, 2016

- In depth wise (resp. grouped) convolutions, the filter and input are split by channels (resp. groups of channels), convolved separately, then concatenated
- We can increase the number of channels and maintain roughly the same computational complexity with this technique, and performance often improves



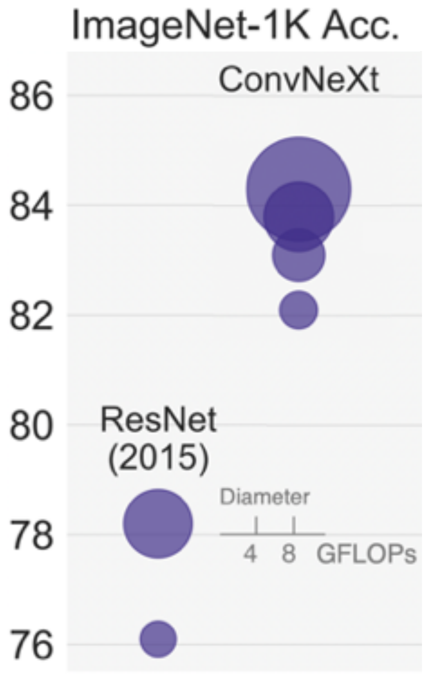




A recent state-of-the-art example

Liu et al, 2022

- ConvNeXt is a recent state-of-the-art conv net that aggregates several methods to achieve improved performance
- Improved training techniques (cosine learning rate schedule, AdamW, lots of data augmentation) turn out to help significantly
- Using depth wise convolutions (and proportionally increasing the number of channels) also significantly improves accuracy
- Some other changes, such as swapping BN for LN and swapping ReLU for GELU, provide smaller gains but appear to not be as important



Computer vision

- Make sure to review Prof. Malik’s lecture and absorb the high level ideas
- What are some computer vision problems other than image classification that researchers have tackled?
- What is the current “frontier” of problems that are being tackled?
  - And what are the high level ideas behind how to tackle these problems?
- What are the “problems of the future”? What are the challenges facing the field of computer vision and AI as a whole?