Applied Deep Learning

DAT310

Lecture:
Deep Learning Best Practices

UNIVERSITY OF CALGARY | Start something.

Agenda

- General ML Pipeline
- Rules of Thumb
- Debugging Networks
- Improving Results
- Faster Networks

General ML Pipeline

General ML Pipeline

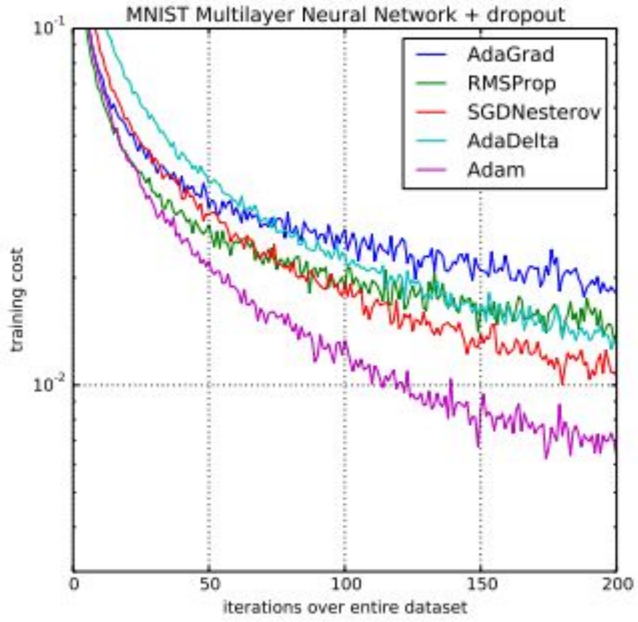
1. Get a dataset
2. Build a dataset and dataloader
3. Choose some models for the task
4. Train your models
    a. Need optimizer and loss function
    b. Compute loss w/ output of network, compute gradients using backwards
    c. Apply gradients with optimizer.step()
5. Evaluate your models

Rules of Thumb

Rules of Thumb

Optimizer

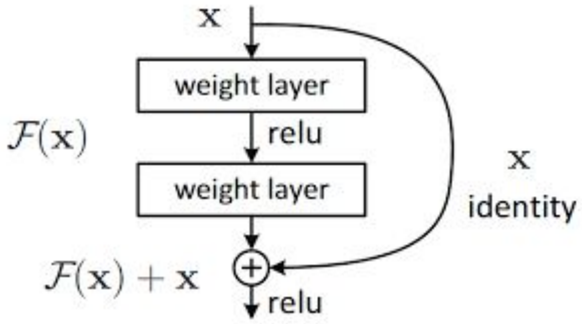
- Pick Adam as your optimizer
  - Combo of Momentum and RMS Prop
  - Automatically figures out a learning rate for you!



Rules of Thumb

Non Linearities

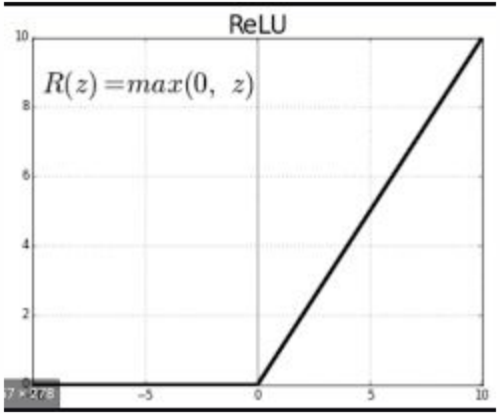
- Deeper VS Wider -> Deeper is better
  - Why? More non linearities
- How to make deeper?
  - Skip Connections

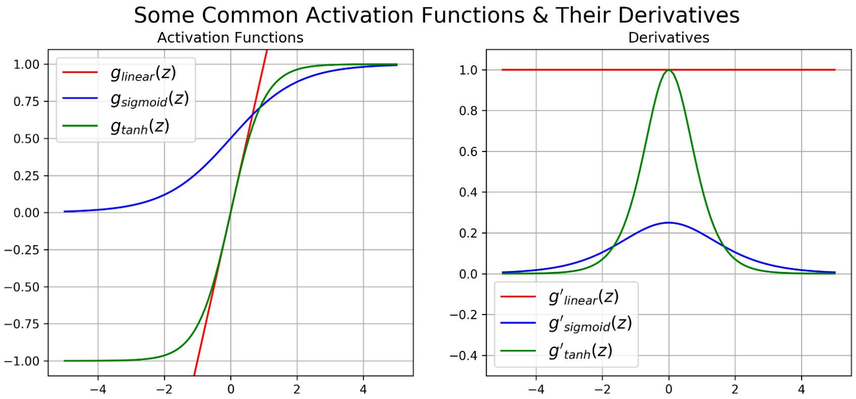


Rules of Thumb

Non Linearities

- Pick Relu
  - No exploding or vanishing gradients like tanh or sigmoid





Rules of Thumb

CV Architecture Choice

- For Computer Vision
  - CNNs -> Bias towards texture
  - Transformers -> Bias towards shape -> Model local visual structures

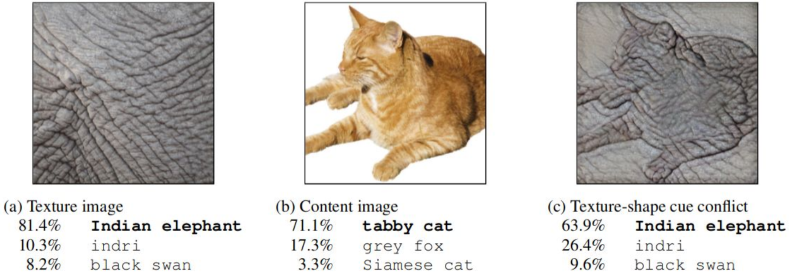
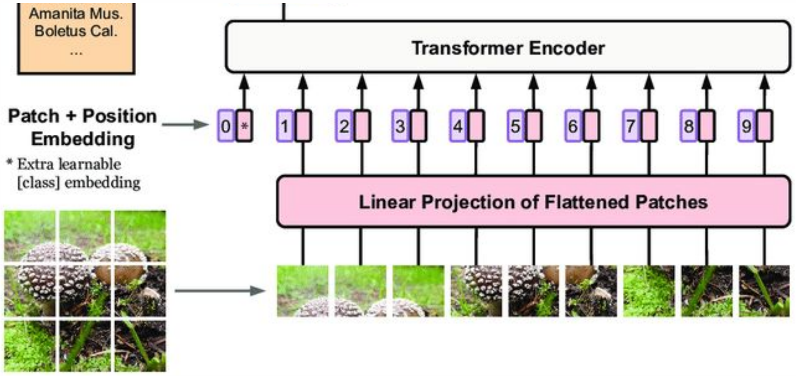


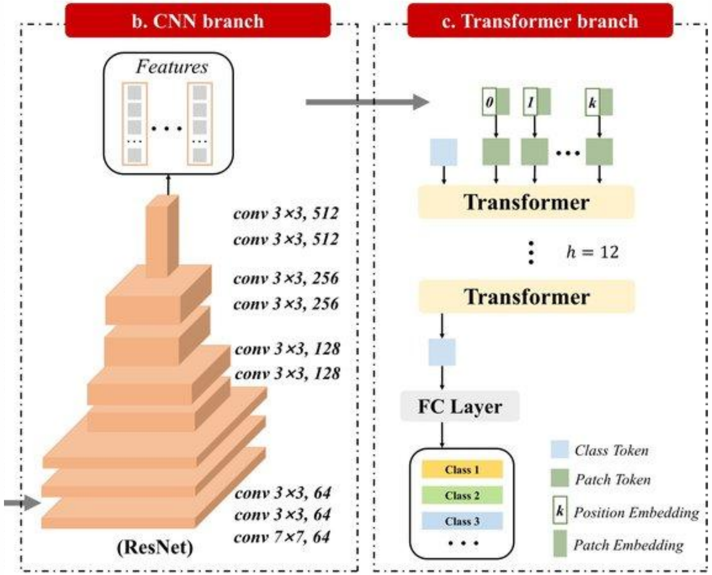
Figure 1: Classification of a standard ResNet-50 of **(a)** a texture image (elephant skin: only texture cues); **(b)** a normal image of a cat (with both shape and texture cues), and **(c)** an image with a texture-shape cue conflict, generated by style transfer between the first two images.



Rules of Thumb

CV Architecture Choice

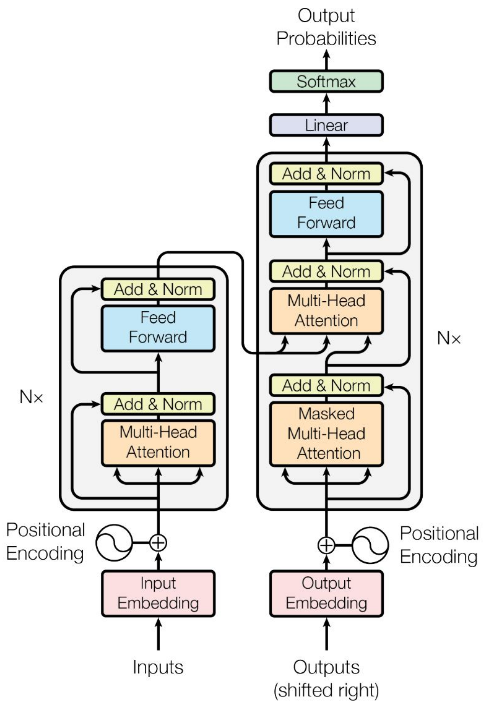
- Solution?
  - CNN Features into Transformer



Rules of Thumb

NLP Architecture Choice

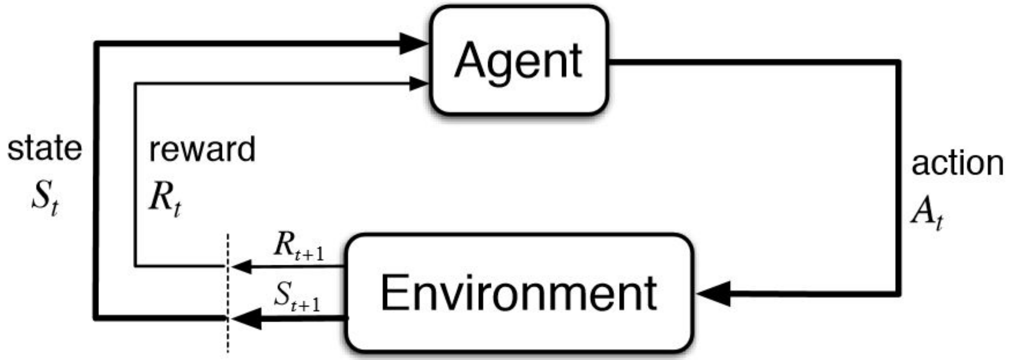
- No one uses RNNs anymore
- LSTM cells are better for long running sequences
- GRUs better for short
- Transformers better overall
  - Pick a transformer



Rules of Thumb

Reinforcement Learning

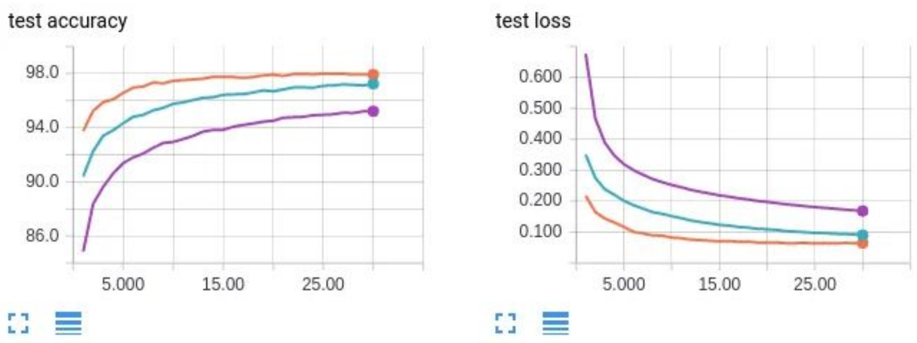
- When do you TYPICALLY use reinforcement learning?
    - No loss function with a gradient available
- You have an environment that produces a state
    - I.e a video game



Rules of Thumb

Batch Size

- Bigger = Faster training, worse generalization
- Smaller = Slower training, better generalization



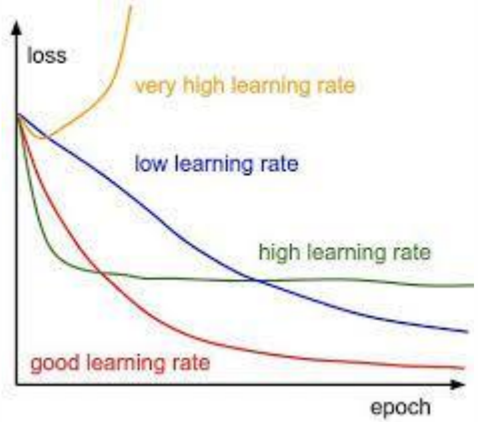
Testing loss and accuracy when the model is trained using different batch sizes.

- Purple = Batch Size 1024
- Blue = 256
- Orange = 64

Rules of Thumb

Learning Rate

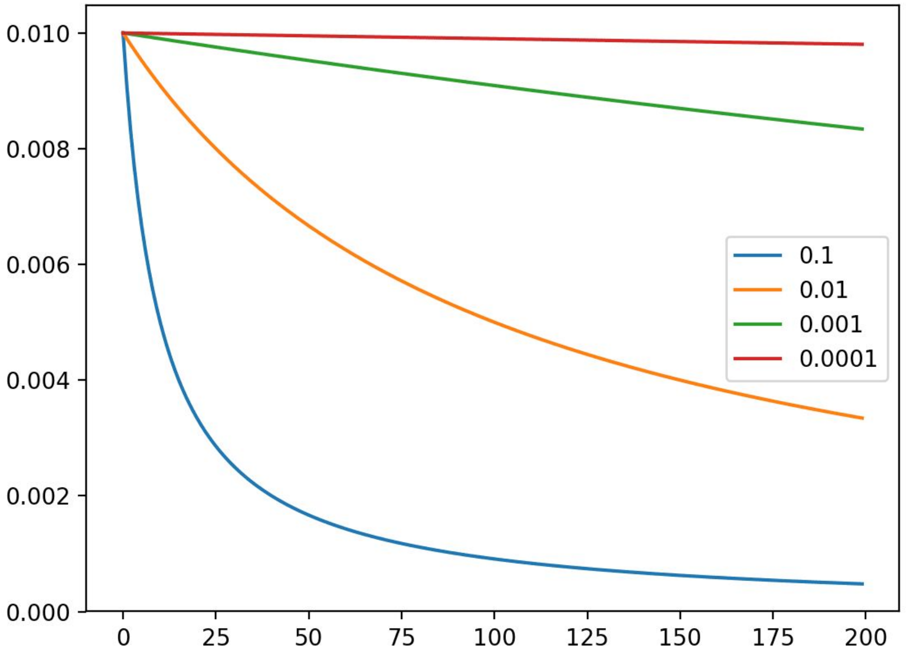
- Plot your loss per LR



Rules of Thumb

Learning Rate

- Plot your loss per LR



Rules of Thumb

UNIVERSITY OF CALGARY Start something.

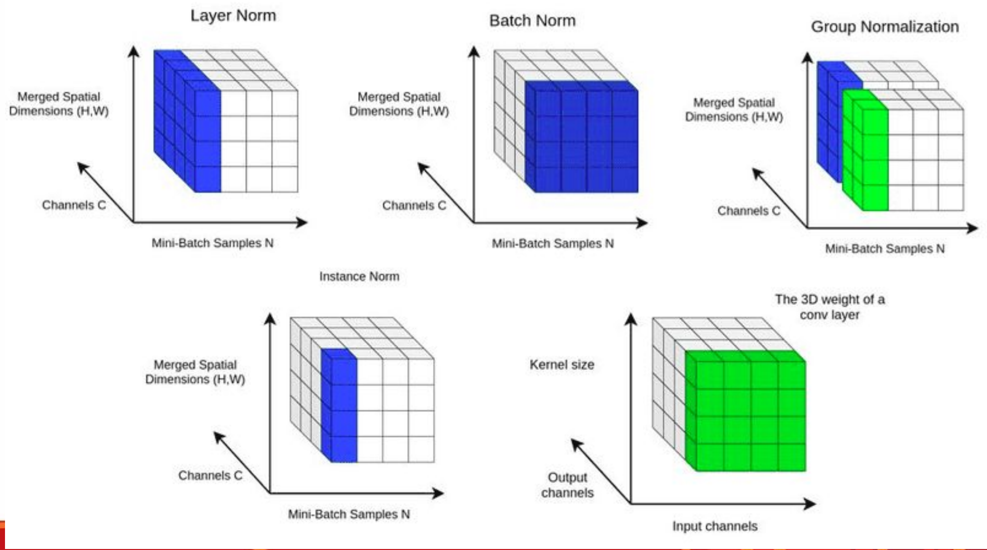
Transfer Learning

- Always use pretrained models!
  - ImageNet or COCO for CV
  - Language Models for NLP

Rules of Thumb

BatchNorm

- Always use BatchNorm (or some variation)!
  - Trains 6X faster
  - Less sensitive to LR
  - Helps w/ vanishing and exploding gradients



Rules of Thumb

Model Size VS Dataset Size

- If you are positive your implementation of the network and training is correct, but you still cannot overfit to just a small subset of training data…. Make your model deeper + wider
- As long as your training error is low, you can always decrease generalization error by collecting more training data

Rules of Thumb

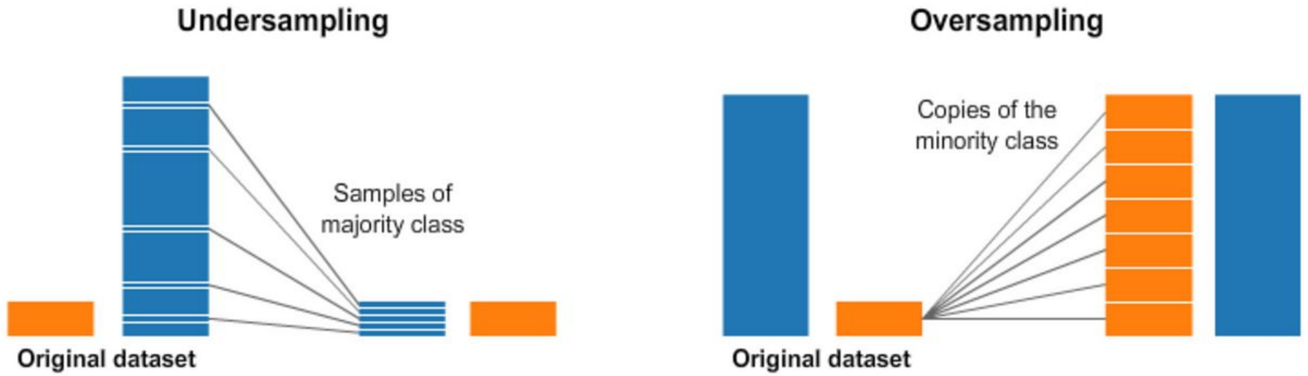
Output Activation

- Binary Classification: Sigmoid
- Multi Class: Softmax
- Regression: Whatever matches the scale of your outputs

Rules of Thumb

Imbalanced Data

- Oversample
- Undersample
- Weight the loss
- Use balanced accuracy and a confusion matrix to evaluate with



Rules of Thumb

Regularization

- L2 is less forgiving, set lambda small i.e 0.0001
- L2 is used by default in pytorch

Rules of Thumb

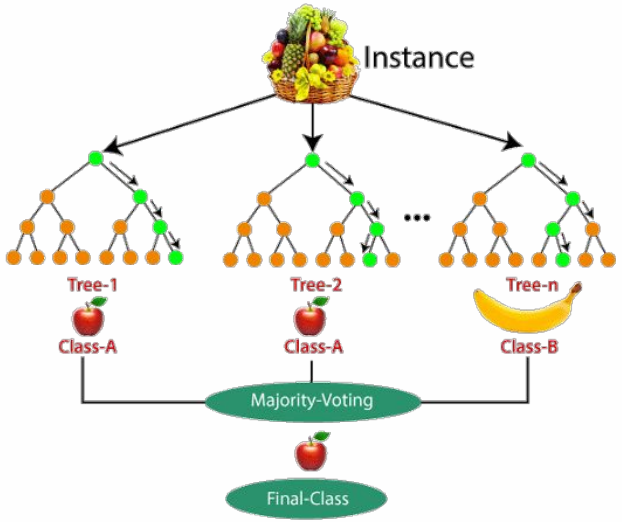
Gradient Clipping

- I loathe gradient clipping
- Prevents exploding grads
- Should fix exploding grads before using this
- Prevents gradients from being over a certain size
    - I find this cripples your learning

Rules of Thumb

Don’t forget about Classic ML Methods

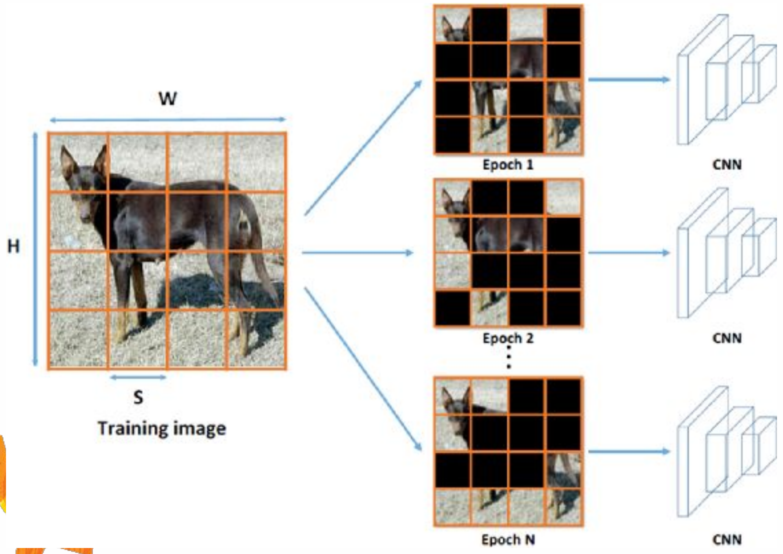
- Linear Regression, Logistic Regression, Random Forest, Gradient Boosting… all excellent models that are fast to train
- IME they do very well on tabular data, awful on text and images



Rules of Thumb

Look into NON Standard Data Augmentations

- CutMix
- Hide and Seek



	ResNet-50	Mixup [48]	Cutout [3]	CutMix
Image				
Label	Dog 1.0	Dog 0.5 Cat 0.5	Dog 1.0	Dog 0.6 Cat 0.4
ImageNet Cls (%)	76.3 (+0.0)	77.4 (+1.1)	77.1 (+0.8)	**78.6** (+2.3)
ImageNet Loc (%)	46.3 (+0.0)	45.8 (-0.5)	46.7 (+0.4)	**47.3** (+1.0)
Pascal VOC Det (mAP)	75.6 (+0.0)	73.9 (-1.7)	75.1 (-0.5)	**76.7** (+1.1)

Debugging Networks

UNIVERSITY OF CALGARY
CONTINUING EDUCATION

Start something.

Debugging Networks

1. Check dataloader
2. Overfit on a small subset of data
3. Play with LR
4. Are your parameters registered?
5. Check gradients

Debugging Networks

Check Dataloader

- For CV insure your images are loaded properly
- Insure you’re labels are aligned
- If you apply augmentations to your images, you MIGHT need to apply them to the labels
  - During segmentation for example

Debugging Networks

Overfit on a Small Subset

- Your network should be able to memorize a small subset of data
- This is MUCH faster than using the entire dataset
- Allows you to check that the network works, LR is correct, not too much regularization, etc.… QUICKLY

Debugging Networks

Play With LR

- Covered the plotting a lot already
- Make the plots!

Debugging Networks

Check Parameters are Registered

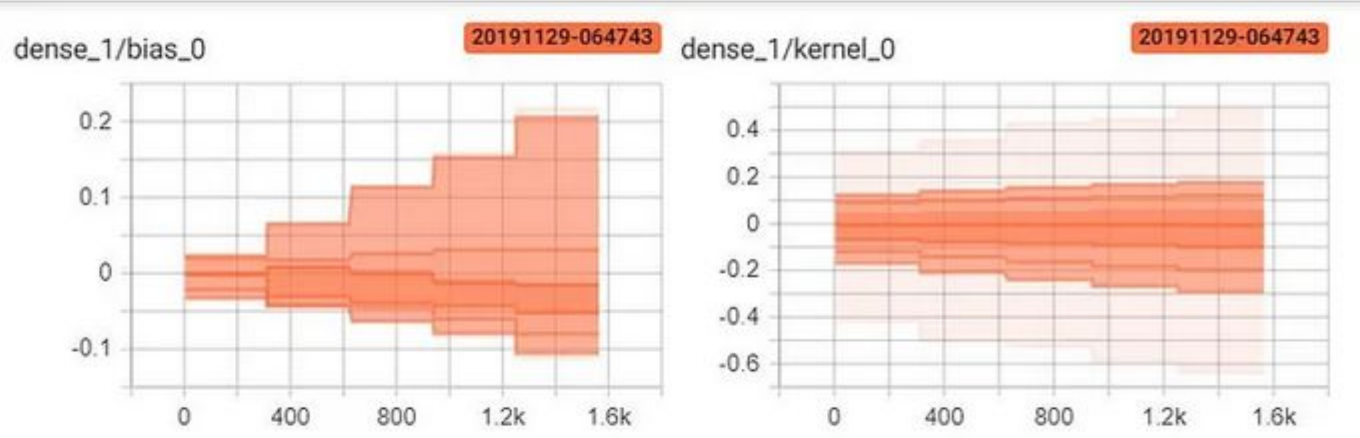
- Print out the parameters of your model
- Make sure all layers are there

- For parameter in model.parameters():
  print(parameter)

Debugging Networks

Check Gradients

- Can use tensorboardX to viz gradients
- Can check if exploding or vanishing
- Can check if proper size throughout the network



Improving Performance

Improving Performance

Hyperparameter Search

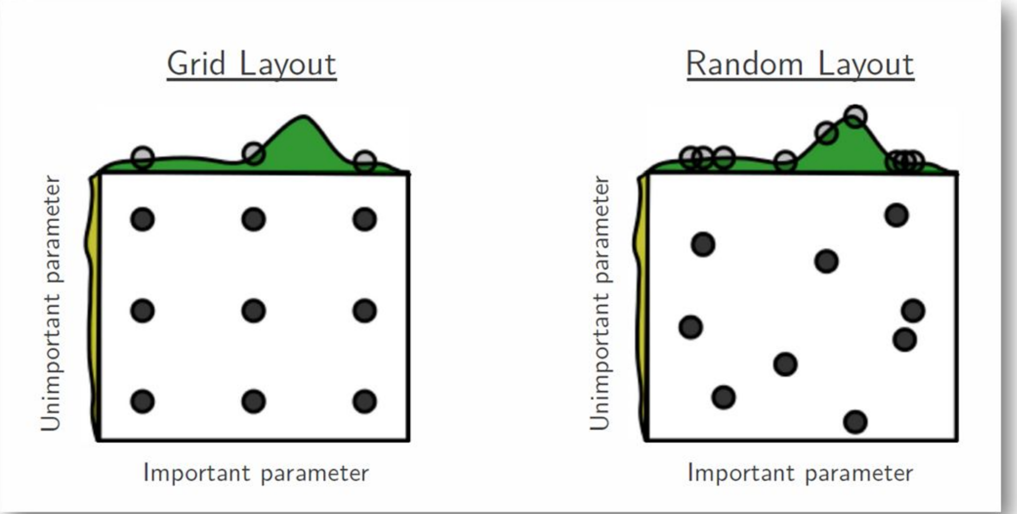
The learning rate is the most important hyperpameter. If bound by time, focus on tuning it.

- The learning rate can be picked by monitoring learning curves that plot the objective function over time.
- The optimal learning rate is typically higher than the learning rate that yields the best performance after the first ~100 iterations, but not so high that it causes instability..

Improving Performance

Hyperparameter Search

Random search typically converges to good hyperparameters faster than grid search.



Improving Performance

Check Examples w/ Largest Loss Values

- Save losses on validation or training
- Check the examples with the largest loss values
- Is there a pattern?
- If so, look for papers that address this, or get more data labelled!

Improving Performance

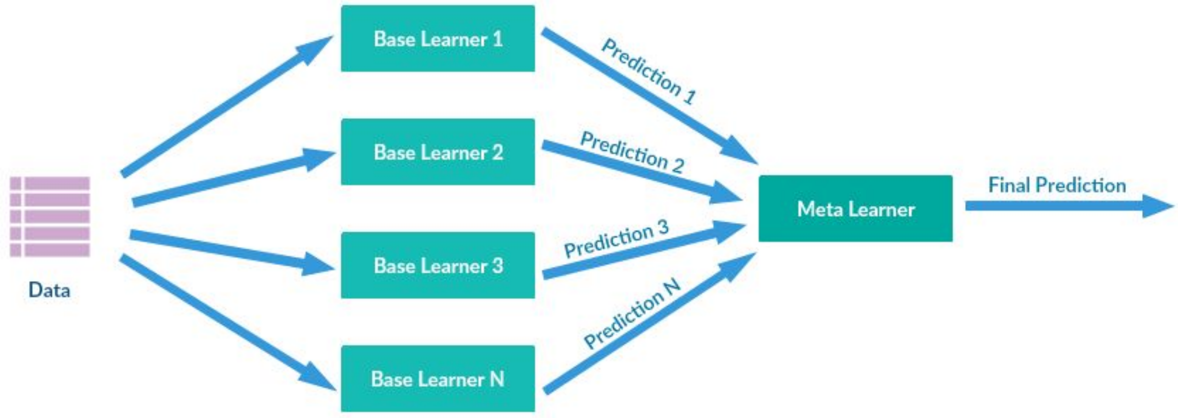
Early Stopping

- Compute validation loss + accuracy every epoch
- Save the best epoch, even if it's not the last one
- That's your model!

Improving Performance

Ensembling

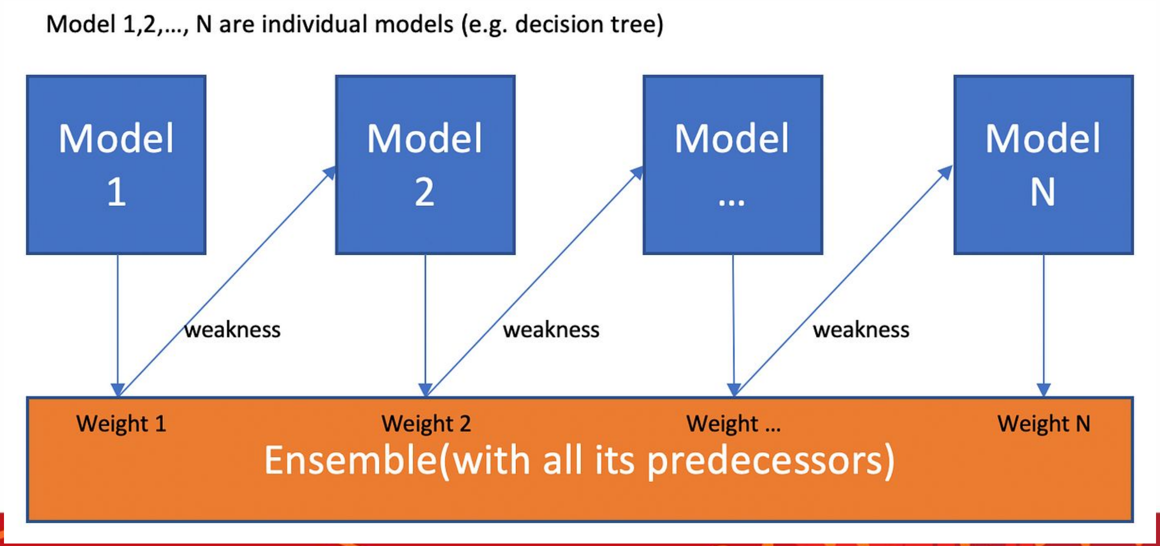
- Train multiple models and combine their predictions
  - Usually through averaging the output probabilities
- Great for increasing performance, bad in actual production
  - Too expensive to run!



Improving Performance

Boosting

- Boosting = Training a model on the errors of your previous model
- Then ensembling the two models
- Fancier version of ensembling



Improving Performance

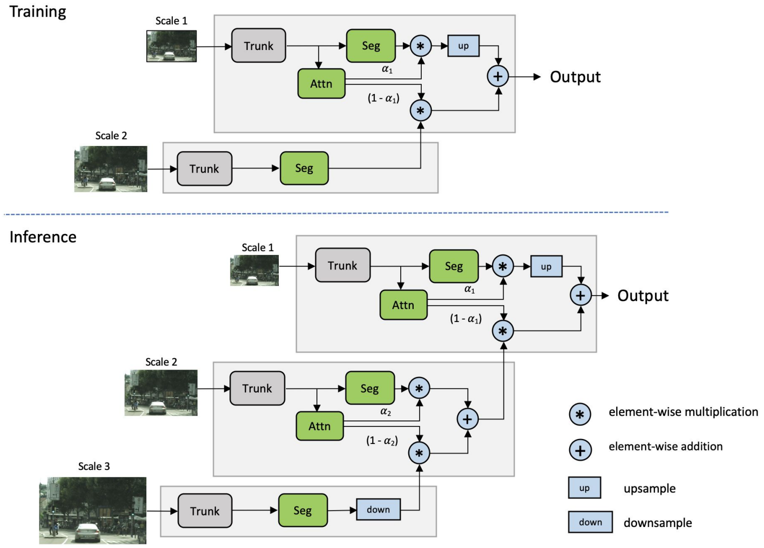
Increase Resolution

- Higher input resolution almost ALWAYS gives better results
- Can test / validate with a higher input resolution than trained with
    - Form of regularization

Improving Performance

Multiscale Predictions

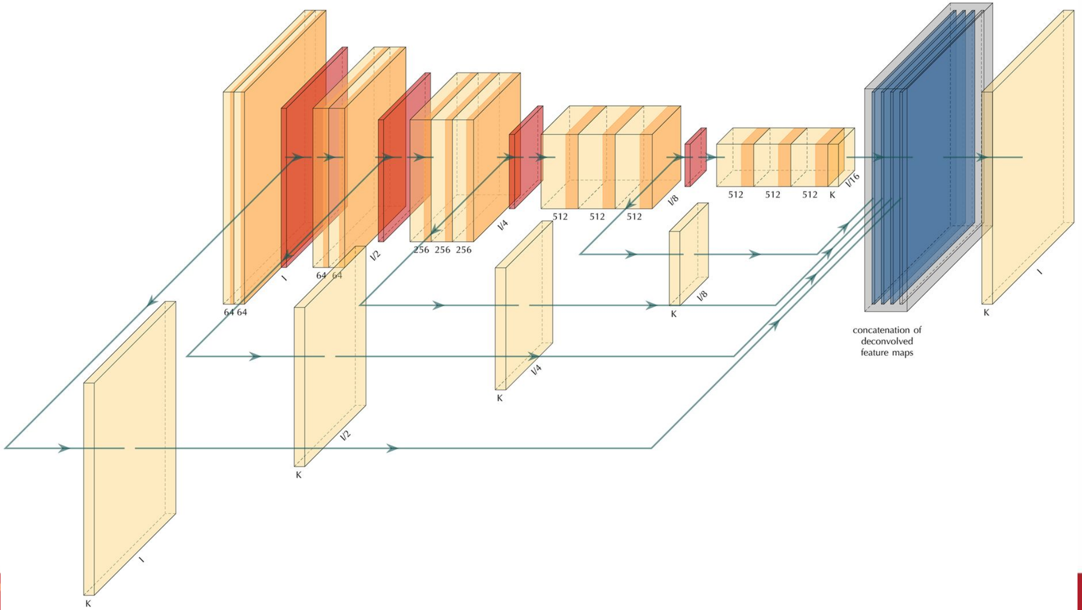
- Train and test using multiple resolutions
- Combine the results of multiple resolutions for your predictions



Improving Performance

Use output of all CNN blocks for Predictions

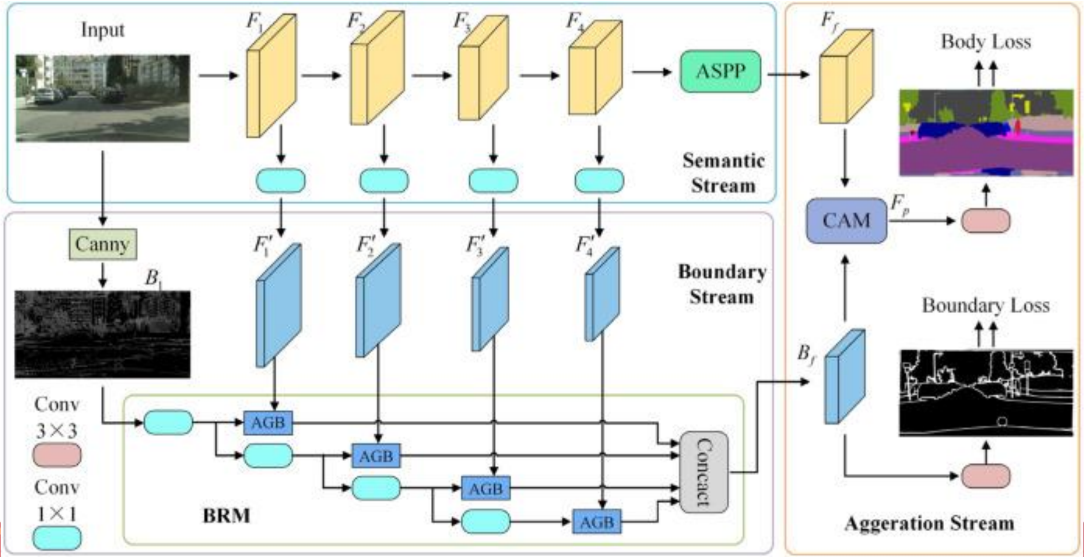
- Concat the output from the end of each block, then make predictions using that



Improving Performance

Auxiliary Losses

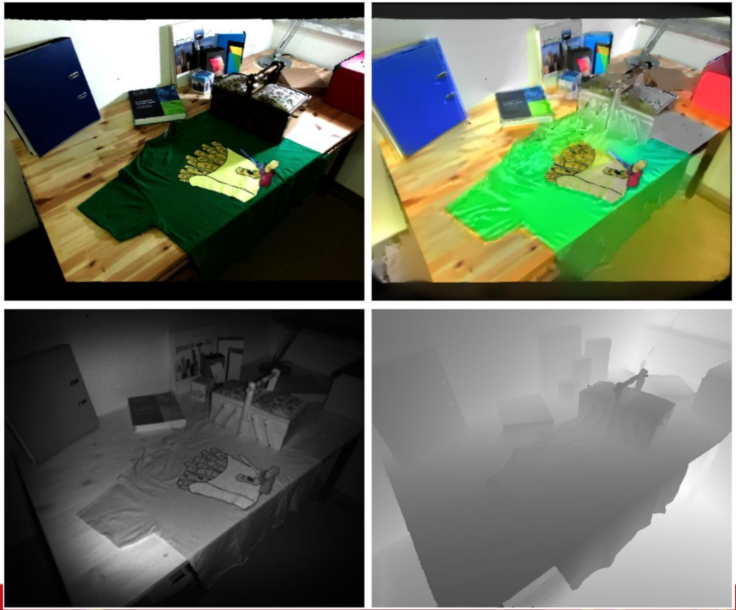
- Use a loss that addresses the weakness of your current model
  - Boundary loss + Segmentation loss for example



Improving Performance

Don’t just use RGB Input

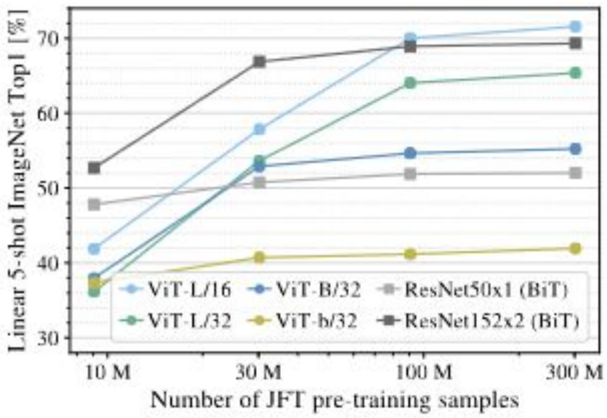
- Can use RGB + Depth + Edges + a Segmentation etc….
- Sometimes adding this extra channel improves performance



Improving Performance

When Transformers Excel

- Need AT LEAST 10 MILLION training examples
- Not much better than CNNs until over 100 MILLION training examples



Faster Networks

Faster Networks

UNIVERSITY OF CALGARY Start something.

Pin Memory

- Dataloader(dataset, pin_memory=True)
- Pin_memory transfers to the GPU faster

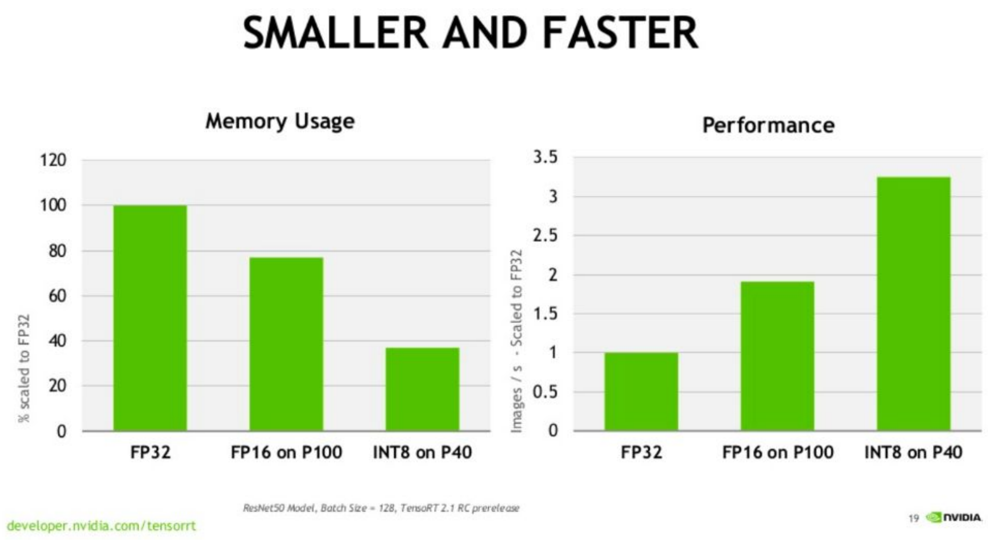
Setting for the training DataLoader	Time for one training epoch
{'num_workers': 0, 'pin_memory': False}	8.2 s
{'num_workers': 1, 'pin_memory': False}	6.75 s
{'num_workers': 1, 'pin_memory': True}	6.7 s
{'num_workers': 2, 'pin_memory': True}	4.2 s
{'num_workers': 4, 'pin_memory': False}	4.5 s
{'num_workers': 4, 'pin_memory': True}	4.1 s
{'num_workers': 8, 'pin_memory': True}	4.5 s

PyTorch 1.6, NVIDIA Quadro RTX 8000

Faster Networks

Half Precision

- By default Pytorch uses float 32
- Half precision is faster, and uses half as much memory
    - model.half()
- Does take a minor performance (accuracy) hit
    - Very minor



Faster Networks

Zero Grads Properly

```
model.zero_grad()

# or

optimizer.zero_grad()
```

- executes memset for every parameter in the model
- backward pass updates gradients with "+=" operator (read + write)

```
for param in model.parameters():
    param.grad = None

# or (in PyT >= 1.7)

model.zero_grad(set_to_none=True)
```

- doesn't execute memset for every parameter
- memory is zeroed-out by the allocator in a more efficient way
- backward pass updates gradients with "=" operator (write)

Faster Networks

cuDNN Benchmark

For convolutional neural networks, enable cuDNN autotuner by setting:

```
torch.backends.cudnn.benchmark = True
```

- cuDNN supports many algorithms to compute convolution
- autotuner runs a short benchmark and selects algorithm with the best performance

Example:
nn.Conv2d with 64 3x3 filters applied to an input with batch size = 32, channels = width = height = 64.

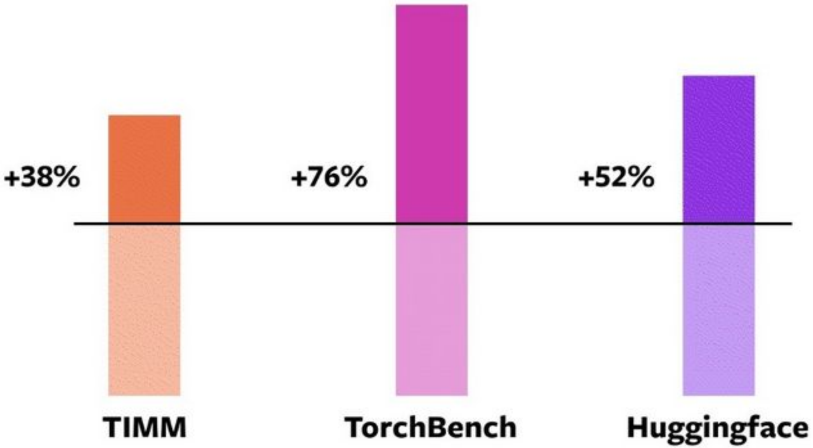
Setting	`cudnn.benchmark = False` (the default)	`cudnn.benchmark = True`	Speedup
Forward propagation (FP32) [us]	1430	840	1.70
Forward + backward propagation (FP32) [us]	2870	2260	1.27

PyTorch 1.6, NVIDIA Quadro RTX 8000

Faster Networks

Pytorch 2.0

- Pytorch 2.0 allows you to compile your model
- Model = torch.compile(model)
- Provides a massive speed up



Speedups for torch.compile against eager mode on an NVIDIA A100 GPU