practical methodology

- neural nets are going to be beasts that won't train easily. should we:
  - gather more data?
  - increase or decrease model capacity?
  - add or remove regularizers?
  - improve the optimization algorithm?
  - debug the software implementation?
- steps:
  - have a metric → loss function, % correct classification etc and have a target value for each of these metrics.
  - establish a working end to end pipeline ASAP, including estimation of above metrics
  - instrument the system with additional visualizations, tensorflow summaries etc, to determine bottlenecks in performance → overfitting, underfitting, bad data etc.
  - repeatedly make incremental changes gathered from metrics above.
- figure out the properties needed for your specific application: error rates, precision, recall etc.
- coverage is the fraction of examples for which the ML system is able to provide a confident enough answer. we can trade off coverage for accuracy.
- First, start with a standard neural net, CNN, RNN with LSTM etc. Use early stopping, dropout and batch norm → they are simple and effective.
- Some domains, like NLP are known to benefit from unsupervised pretraining (embeddings). pretraining can also help vision models if amount of available data is not much.
- Remember, while using a CNN, a wider kernel results in narrower output dimension, reducing model capacity unless we use some form of padding.
- how to debug?
  - if performance on training set is poor, the algorithm is not using the training data already available → no point collecting more data.
  - if large models and carefully tuned optimization don't work to reduce training error, problem might be quality of training data, or collecting a richer set of features.
  - if test set performance is much worse than training set performance, then gathering more data is probably the single best thing to do. dataset augmentation also helps here.
  - reducing the size (and hence capacity) of the model might also be a thing to try out if you can't get a ton more data.
  - consider pretraining/transfer learning if you can't find more data in your specific problem.
  - plotting curves on training and test set is a must!
- selecting hyperparameters:
  - learning rate is perhaps the most important one to tune.
  - manual tuning of scale of hyperparameter and zeroing in on a good one after a reasonable one is found.
  - The test error is sum of training error and gap between training and test error. Neural nets typically perform best when training error is very low (high capacity model) and the test error is primarily driven by gap between training and test error. the goal is to reduce this gap without increasing training error too much.

    - Grid search and random search are two strategies for automated hyperparameter search. random search is more scalable for 3 or more hyperparameters since it wastes less time trying all possible settings of other hyperparameters when one of them is really bad so as for others to not matter.
- Debugging strategies:
    - visualize the model results → seems obvious but don't get lost in quantitive metrics alone!
    - visualize the worst mistakes → there's often a pattern you might just catch.
    - fit a tiny dataset → if you can't even do this, there might be a software bug.
    - compare back-propagated derivatives to numerically computed derivatives.
    - monitor histograms of activations and gradients
- If training and test error are similar, but you're not happy with overall model performance, it is probably under-fitting (increase model size, maybe throw in some regularization!) or a pattern with what it is getting wrong!