UVA CS 6316: Machine Learning : 2019 Fall
Course Project: Deep2Reproduce @
https://github.com/qiyanjun/deep2reproduce/tree/master/2019Fall

Select Via Proxy: Efficient Data Selection For Training Deep Networks

Cody Coleman, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, Matei Zaharia.
Stanford University.

Reproduced and Presentation by Paola Cascante-Bonilla.

October 2019.

Motivation

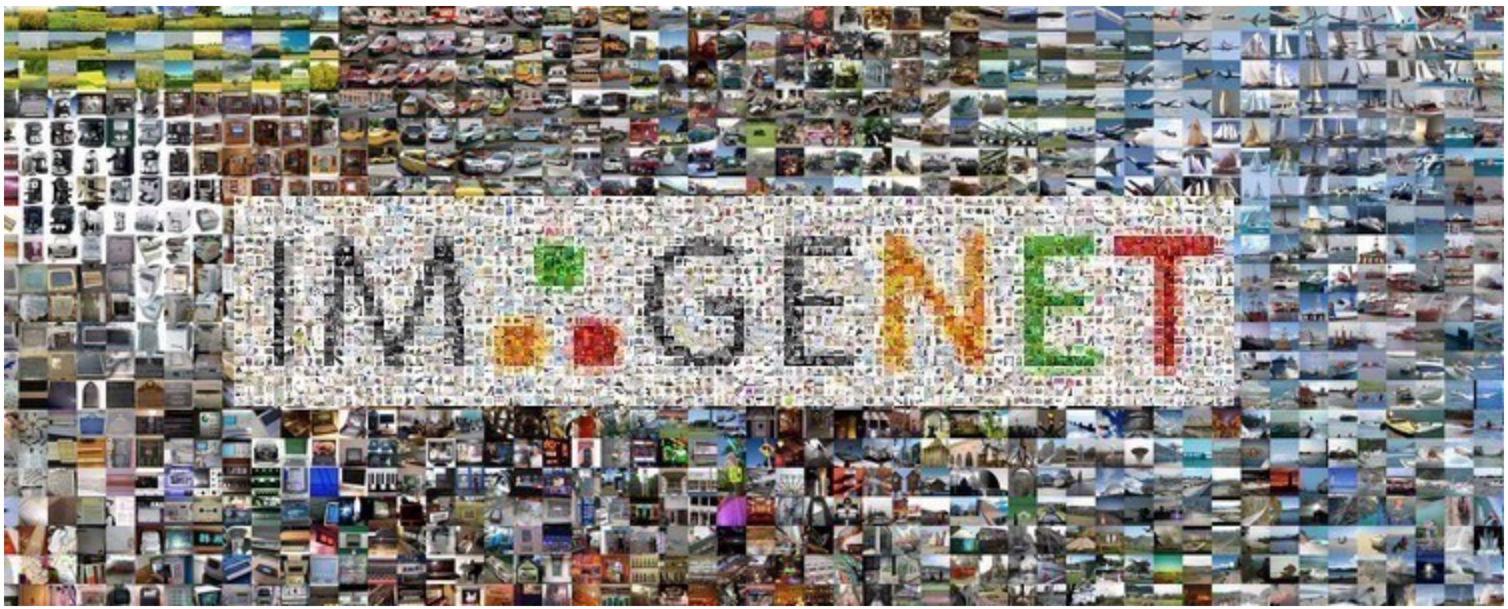
?

Sample of cats & dogs images from Kaggle Dataset

https://www.kaggle.com/c/dogs-vs-cats

Motivation

Large amounts of annotated data available in multiple domains!

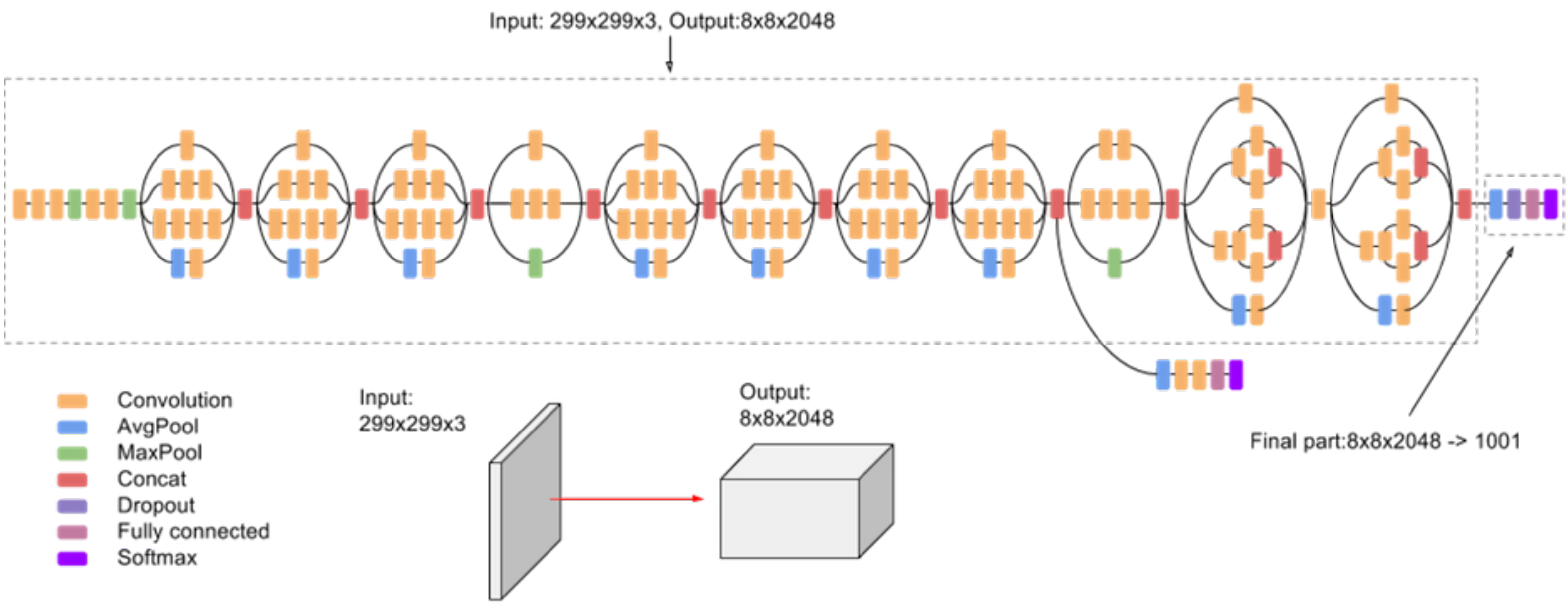


- Total number of non-empty synsets: 21841
- Total number of images: 14,197,122

http://image-net.org/about-overview

Motivation

Deep models to train all that data!



Top 1-Error of 4.2% on the ILSVRC 2012 classification benchmark [1]

[1] Rethinking the Inception Architecture for Computer Vision

https://cloud.google.com/tpu/docs/inception-v3-advanced

Motivation

But…

- Training large models over all available data can be computationally expensive

- Training deep networks can incur prohibitively long training times, measured in days, weeks, or even months

- This overhead impedes the development of new machine learning models and uses large amounts of computational resources

Background

- Subsampling training data is a common solution.

Not enough yellow lights!

- Core-set selection techniques is another solution.

https://www.thenewsletterpro.com/swimming-right-embracing-your-uniqueness/

Background

proxy [**prok**-see] SHOW IPA

EXAMPLES | WORD ORIGIN

SEE MORE SYNONYMS FOR *proxy* ON THESAURUS.COM

noun, plural **prox·ies.**

1. the agency, function, or power of a person authorized to act as the deputy or substitute for another.
2. the person so authorized; substitute; agent.
3. a written authorization empowering another person to vote or act for the signer, as at a meeting of stockholders.
4. an ally or confederate who can be relied upon to speak or act in one's behalf.

Related Work

Core-set selection.
Find a representative subset of points to speed up learning or clustering.

K-means, Bayesian Inference, SVM.

Subset selection.
Choose data points whose predictions have changed most over the previous epochs as a lightweight estimate of uncertainty [2].

Using Reinforcement Learning → Student/teacher model.

↓

accuracy and training time

[2] Active bias: Training more accurate neural networks by emphasizing high variance samples

Related Work

Heterogeneous active learning.
Use one model to select points for a different, more expensive model.

NLP Task -> CRF, Naïve Bayes, Maximum Entropy.

Optimization and Importance Sampling.
Based on Gradient norm, loss, focusing on more "hard" examples in later epochs.

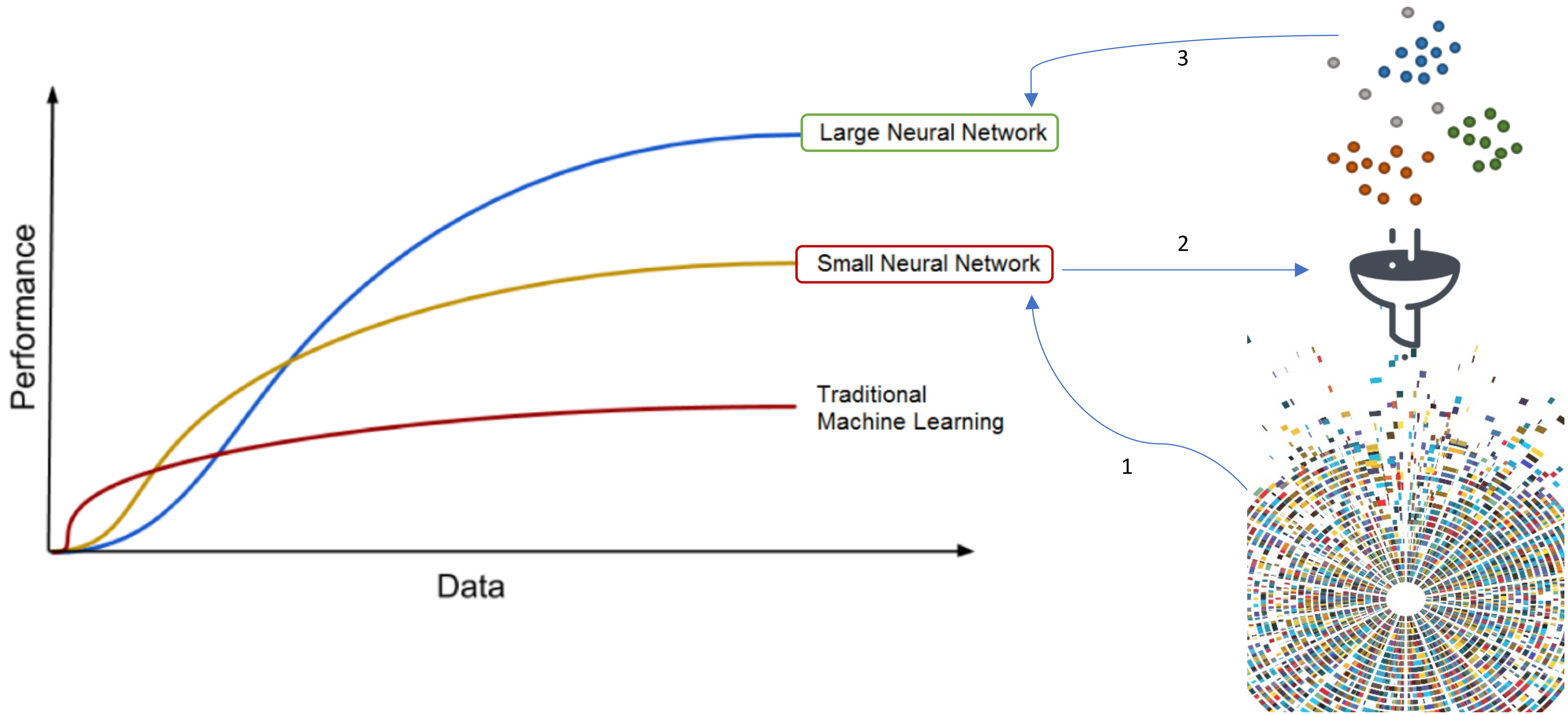
Claim / Target Task

Using a proxy model reduces the cost of selection by up to a 100×

It can be easily added to a training pipeline without modifying the training procedure of the target model

The proxy is very fast to train and can substantially improve the training time of large deep models while maintaining the predictive performance

An Intuitive Figure Showing WHY Claim

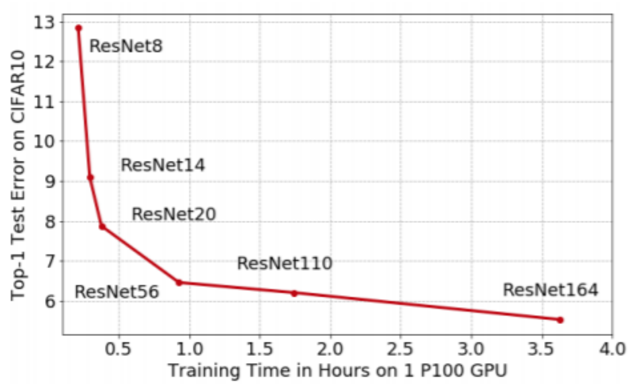


https://www.deeplearning-academy.com/p/ai-wiki-machine-learning-vs-deep-learning

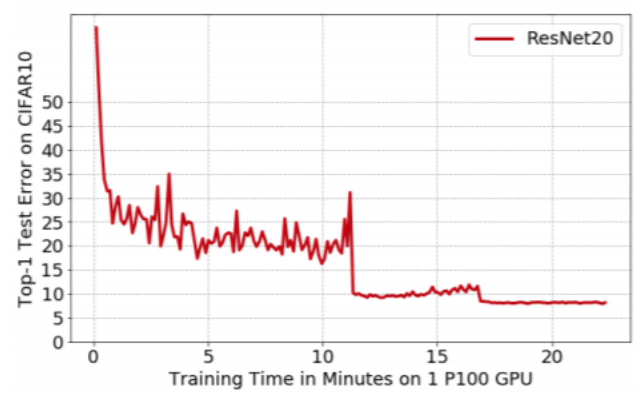
Proposed Solution

Creating a Proxy Model.

- Scaling down the target model



(a) Top-1 test error and training time on CIFAR10 for ResNet with pre-activation and a varying number of layers. There is a diminishing returns in accuracy by increasing the number of layers.



(b) Top-1 test error during training of ResNet20 with pre-activation. In the first 12 minutes, ResNet20 reaches 9.2% top-1 error, while the remaining 10 minutes are spent on increasing accuracy to 7.9%

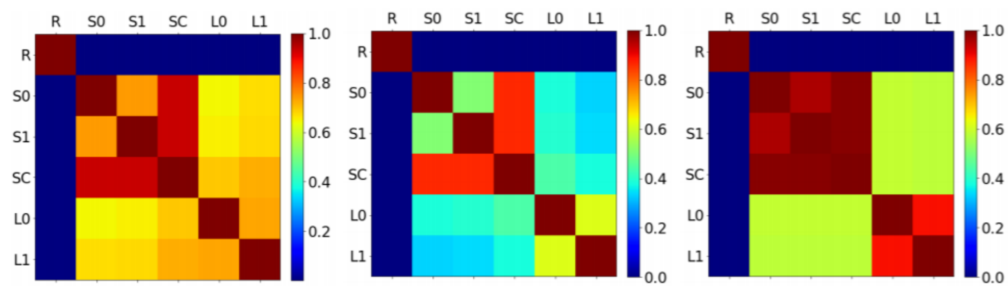
Figure 2: Top-1 test error on CIFAR10 for varying model sizes (left) and over the course training a single model (right), demonstrating a large amount of time is spent on small changes in accuracy.

Proposed Solution

Creating a Proxy Model.

- Training for smaller number of epochs

- Boosting the performance by ensembling small models.



(a) CIFAR10 (b) SVHN (c) Amazon Review Polarity

Figure 3: Pearson product-moment correlation of examples ranked by entropy calculated from different models on CIFAR10 using ResNet20 and ResNet164 with pre-activation (left), SVHN using ResNet20 and ResNet152 (center), and Amazon Review Polarity using fastText and VDCNN29 (right). S1 and S2 represent two separate runs of the small proxy model (e.g., ResNet20), while L1 and L2 represent different runs of the large target model (e.g., ResNet164). R gives a random order of points for reference. On all datasets, ensembling multiple small models together through rank combination (SC) increases the Pearson product-moment correlation with the large model.

Proposed Solution

Subset Selection via Proxy.
Use the proxy model to select the most uncertain data points around the decision boundary.

Quantifying uncertainty:

- Confidence, margin, and entropy
- For every data point *x* that provides *P(y|x)* for *x* to belong to class *y*, the uncertainty function *f* can be defined as:

$$f_{\text{confidence}}(x) = 1 - P(\hat{y}|x) \tag{1}$$

$$f_{\text{margin}}(x) = 1 - \min_{y \neq \hat{y}}(P(\hat{y}|x) - P(y|x)) \tag{2}$$

$$f_{\text{entropy}}(x) = -\sum_{y} P(y|x) \log P(y|x), \tag{3}$$

Proposed Solution

Training the target model on the subsets selected via proxy.
The set of uncertain data points can be used to train the large target model.

1. Select the data points around the approximate decision boundary learned by the proxy model.

2. Let the target model refine the decision boundary of the proxy model.

Implementation

Algorithm 1 SELECT VIA PROXY (SVP)

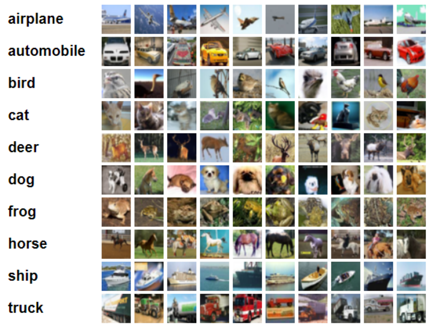
Input: Data set D, cardinality k, deep model architecture $\mathcal{M}$.
Output: Trained deep model $\mathcal{M}^t$.

1: Create a proxy model by scaling down the target model as described in section 3.1.
2: Train the small proxy model on the entire dataset D.
3: Calculate uncertainty of data points via the proxy model using uncertainty metrics from section 3.2.
4: Sort the examples in a decreasing order based on their uncertainty.
5: Train the target model $\mathcal{M}$ on the subset S of top k uncertain examples to get the final output $\mathcal{M}^t$.
6: **return** $\mathcal{M}^t$.

M:		M_t:
ResNet20	$\rightarrow$	ResNet164 [4]
fastText [5]	$\rightarrow$	VDCNN29 [3]

Data Summary

CIFAR-10



10 classes.
Train set: 50k images
Test set: 10k images

Balanced:
5k images per class

SVHN

10 classes.
Train set: 73257
Test set: 26032

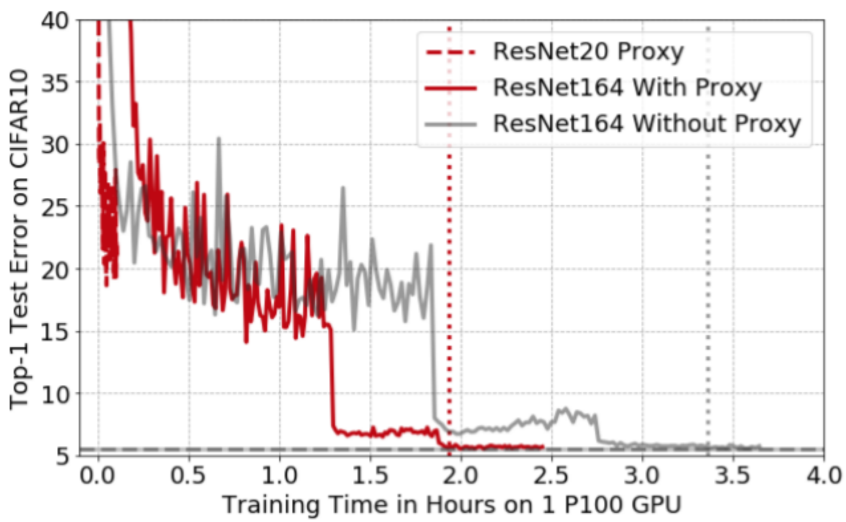
Extra training data: 531131

Amazon Review Polarity

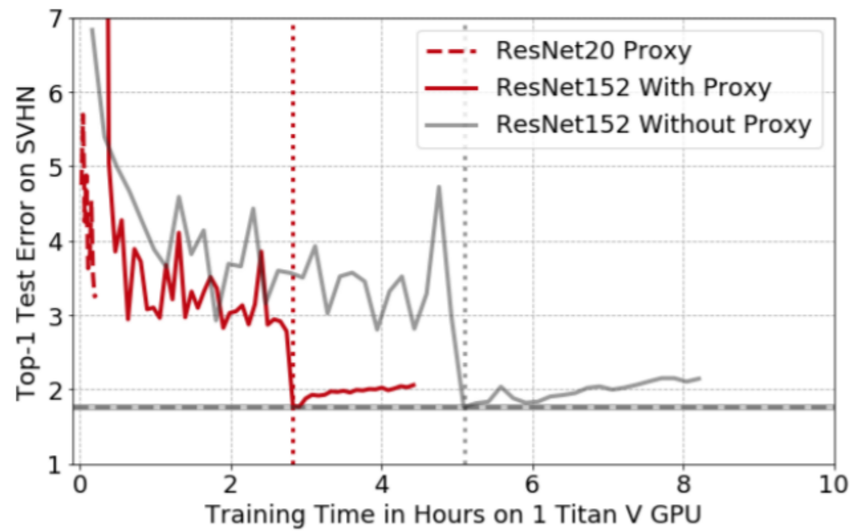
2 classes
Train samples: 3,600,000
Test samples: 400,000

Experimental Results

Training without data selection via proxy vs SVP algorithm:



(a) CIFAR10



(b) SVHN

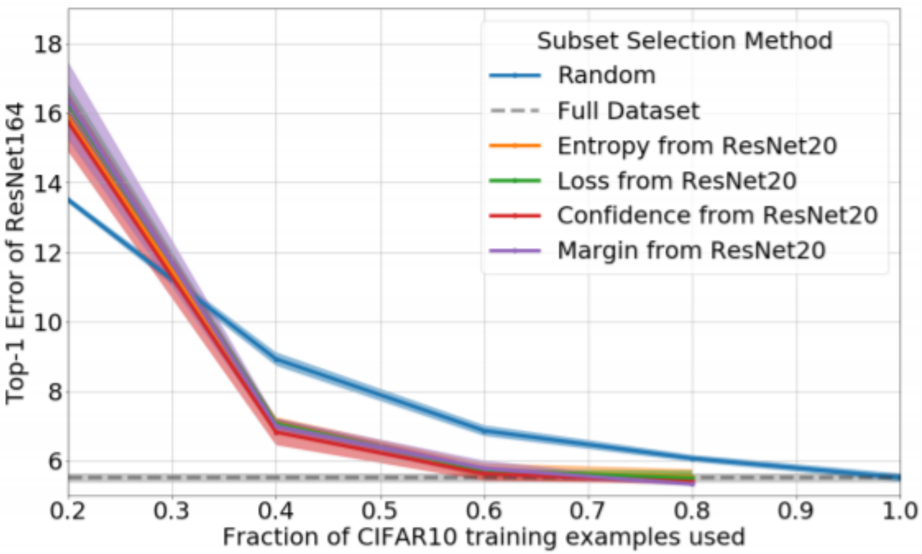
Experimental Results

Average Top-1 error and standard deviation for 3 runs of different proxy models across a range of subset sizes of the CIFAR10, SVHN, and Amazon Review Polarity datasets:

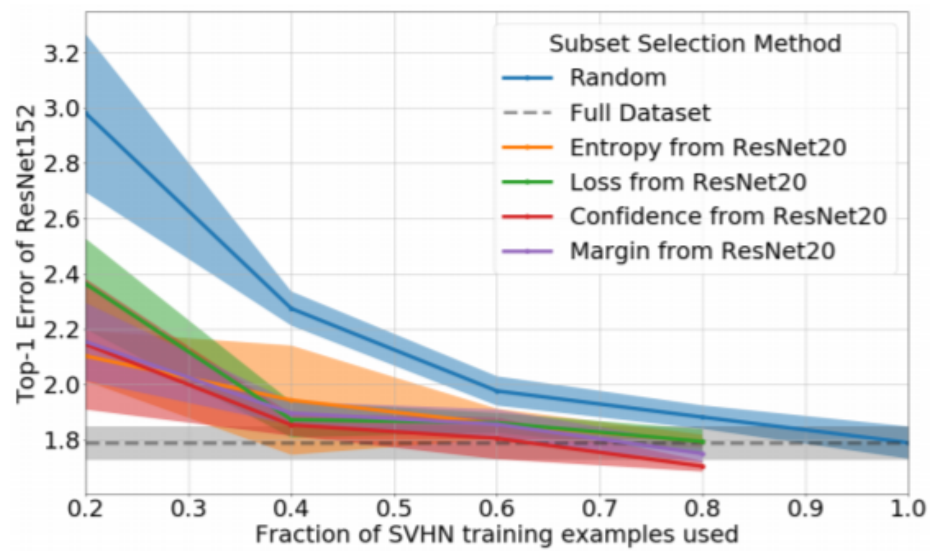
	Proxy			Fraction of Dataset			
Dataset	Architecture	Metric	Epochs (n_p)	0.4	0.6	0.8	1.0
CIFAR10	3xResNet20	Entropy	50	6.52 ± 0.21	5.46 ± 0.06	-	-
CIFAR10	1xResNet20	Entropy	50	6.83 ± 0.07	5.61 ± 0.09	-	-
CIFAR10	1xResNet20	Entropy	180	7.09 ± 0.17	5.71 ± 0.22	5.53 ± 0.23	-
CIFAR10	1xResNet164	Entropy	181	7.83 ± 0.32	6.31 ± 0.15	5.68 ± 0.25	5.48 ± 0.08
CIFAR10				8.93 ± 0.19	6.87 ± 0.16	6.07 ± 0.10	5.52 ± 0.12
SVHN	1xResNet20	Entropy	10	1.87 ± 0.03	1.72 ± 0.04	-	-
SVHN	1xResNet20	Entropy	50	1.94 ± 0.20	1.86 ± 0.05	1.79 ± 0.02	-
SVHN				2.27 ± 0.06	1.98 ± 0.05	1.88 ± 0.04	1.79 ± 0.06
Amazon Review Polarity	1xfastText	Entropy	5	4.39 ± 0.02	4.23 ± 0.02	4.16 ± 0.02	-
Amazon Review Polarity				4.89 ± 0.03	4.50 ± 0.05	4.28	4.13 ± 0.04

Experimental Analysis

Comparison of uncertainty metrics:



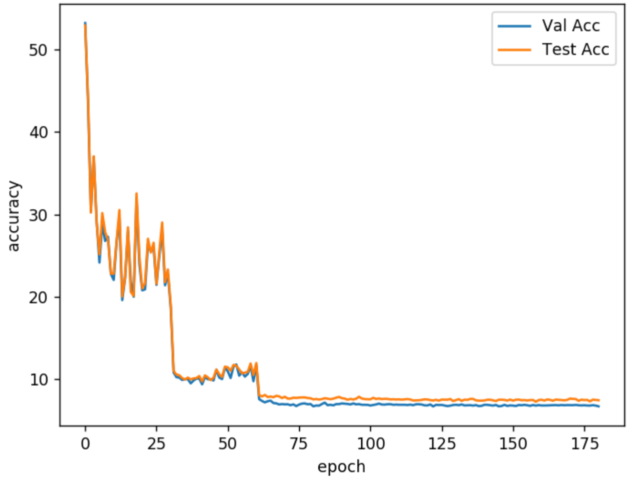
(a) CIFAR10



(b) SVHN

Experimental Results Reproduced

CIFAR10 – Resnet164 – 181 Epochs

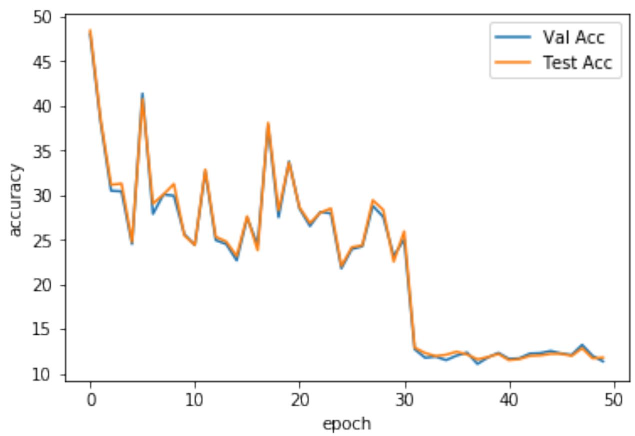


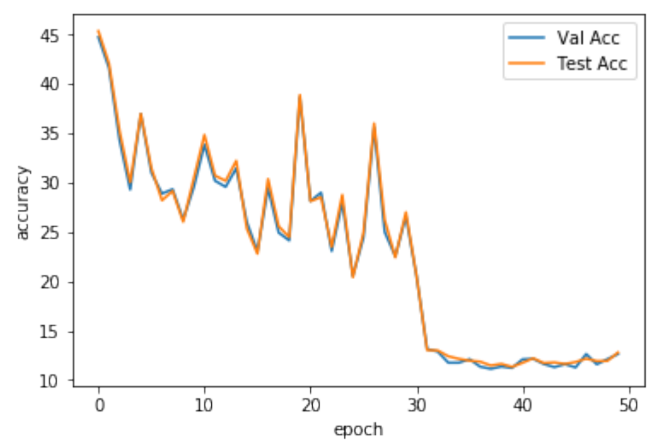
Using a Titan X GPU
- 40k as training set
- 10k as validation set
- 10k as test set

Accuracy achieved: 7.42% -> Time: 260 minutes
Reported accuracy: 5.52% -> Reported time: 240 minutes

Experimental Results Reproduced

CIFAR10 – Resnet20 – 50 Epochs





2x for Model Ensemble

Accuracy achieved: ~11.47% -> Time: ~7.45 minutes
Reported accuracy: 9.2% -> Reported time: 12 minutes

Experimental Results Reproduced

Dataset: CIFAR10

	Proxy			Fraction of Dataset			
Dataset	Architecture	Metric	Epochs	0.4	0.6	0.8	1
CIFAR10	2xResNet20	Entropy	50	8.94	7.83	7.84	--
CIFAR10	1xResNet20	Entropy	50	9.12	8.56	8.02	--
CIFAR10	1xResNet164	No Proxy	180	11.45	11.07	10.92	7.42

Details:

Splits:

- 40k training set
- 10k validation set
- 10k testing set

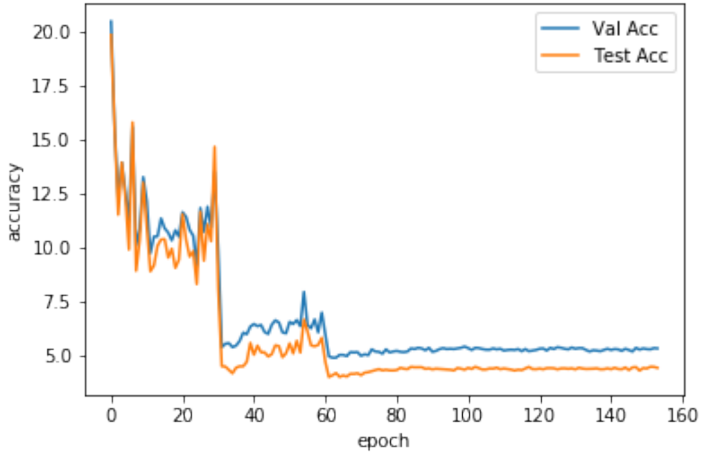
Learning rate: 0.1 with a decay step of 0.1 every 30 epochs

Batch size: 64

SGD: momentum 0.9 – weight decay: 0.0005

Experimental Results Reproduced

SVHN – Resnet152 – 150 Epochs

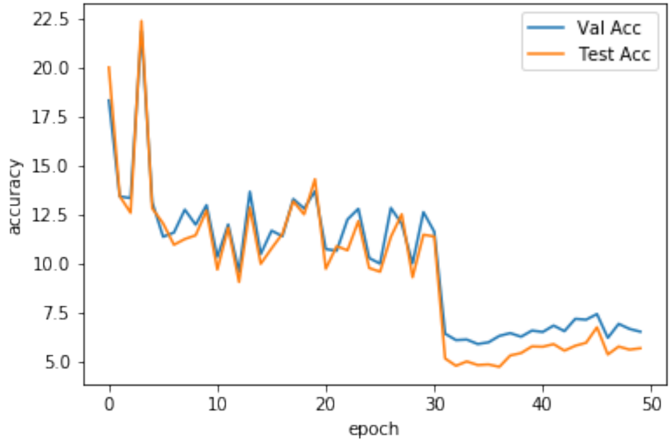


Using a Titan X GPU

Accuracy achieved: 4.09% -> Time: 294 minutes
Reported accuracy: 1.79% -> Reported time: 480 minutes

Experimental Results Reproduced

SVHN – Resnet152 – 50 Epochs



Using a Titan X GPU

Accuracy achieved: 4.81%	->	Time: 14.40 minutes (50 epochs)
Reported accuracy: 1.79%	->	Reported time: 13.4 minutes (10 epochs)

Experimental Results Reproduced

Dataset: SVHN

	Proxy			Fraction of Dataset			
Dataset	Architecture	Metric	Epochs	0.4	0.6	0.8	1
SVHN	1xResNet20	Confidence	50	4.12	4.61	4.78	--
SVHN	1xResNet152	No Proxy	50	5.37	5.07	4.96	4.81

Details:
Splits:
- 66,257 training set
- 7k validation set
- 10k testing set

Learning rate: 0.1 with a decay step of 0.1 every 30 epochs
Batch size: 64
SGD: momentum 0.9 – weight decay: 0.0005

Experimental Analysis

Comparison of uncertainty metrics:

```
#confidence = 1 - P(y'|x)
def get_correct_values(model, device, data_loader, indices):
    max_correct_results = {}
    model.eval()
    with torch.no_grad():
        for i, (data, target) in enumerate(data_loader):
            data, target = data.to(device), target.to(device)
            output = model(data)
            out_softmax = F.softmax(output, dim=1)
            pred = out_softmax.argmax(dim=1, keepdim=True) # get the index of the max log-probability
            for t, p in enumerate(pred):
              if p == target[t]:
                  max_correct_results[indices[(i*args.batch_size)+t]] = out_softmax[t][pred[t]].item()

    return max_correct_results
```

```
#entropy = -sum P(y|x) log P(y|x)

def get_correct_values(model, device, data_loader, indices):
    max_correct_results = {}
    model.eval()
    with torch.no_grad():
        for i, (data, target) in enumerate(data_loader):
            data, target = data.to(device), target.to(device)
            output = model(data)
            out_softmax = F.softmax(output, dim=1)
            pred = out_softmax.argmax(dim=1, keepdim=True) # get the index of the max log-probability
            entropy = (-out_softmax.detach().cpu().numpy()*np.log2(out_softmax.detach().cpu().numpy())).sum(axis=1)
            for t, p in enumerate(pred):
              if p == target[t]:
                  max_correct_results[indices[(i*args.batch_size)+t]] = entropy[t].item()

    return max_correct_results
```

Conclusion and Future Work

- A small proxy model can select a subset of data to train a large architecture while maintaining the predictive performance.
- On CIFAR10 and SVHN, the speed of training the proxy model leads to a 1.6× and 1.8× speed-up in end-to-end training time by selecting 60% and 50% of data respectively to train the target model on.
- Train on larger datasets.
  More experiments without making multiple passes over all the data.

References

[1] Rethinking the Inception Architecture for Computer Vision

[2] Active bias: Training more accurate neural networks by emphasizing high variance samples

[3] Very Deep Convolutional Networks for Text Classification

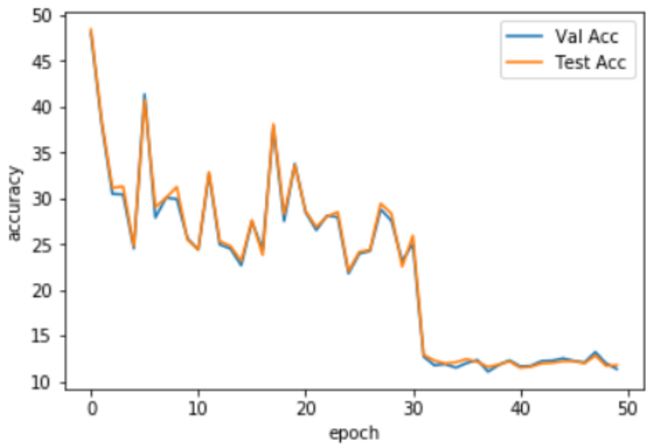
[4] Deep Residual Learning for Image Recognition

[5] Bag of Tricks for Efficient Text Classification

Appendix

```
model_state = torch.load('resnet20_100perct_best_50Epochs_Step30_PreActivations_lr0.1_Best.ckpt')
model.load_state_dict(model_state)
print ('100 Percent \nBest ResNet20 model - Test Accuracy:')
acc = test(args, model, device, test_loader, class_criterion_eval)
total_time = end_time - start_time
print ('Time: {}'.format(total_time))
print ('=======================')
x_epochs = range(0, args.epochs)
plot_acc_valtest(x_epochs, 100 - np.array(record_val_acc), 100 - np.array(record_test_acc))
```

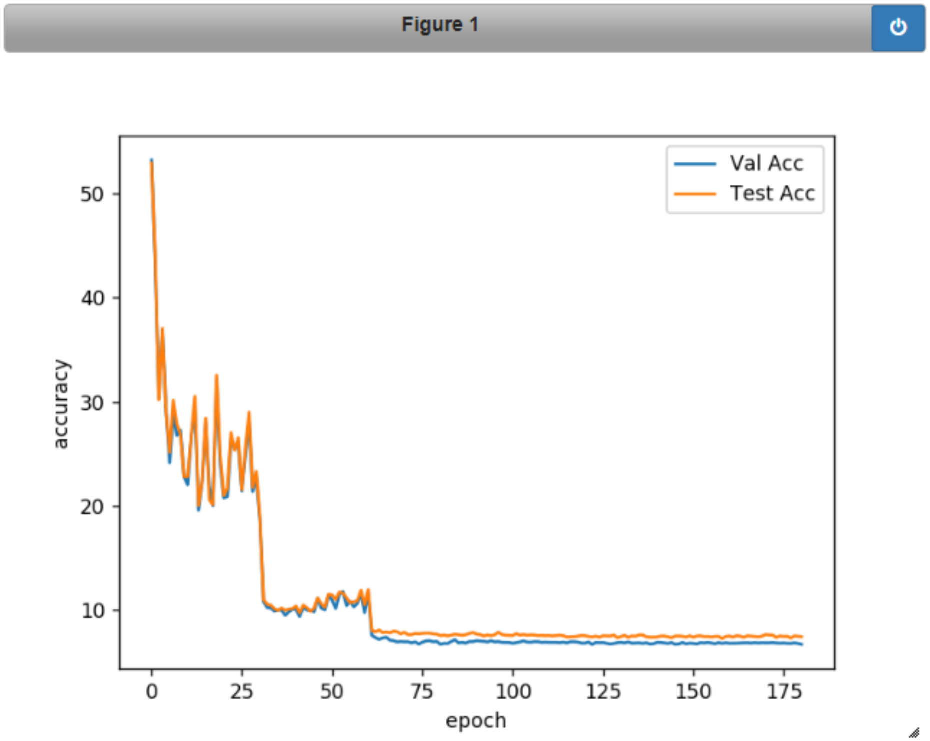
```
100 Percent
Best ResNet20 model - Test Accuracy:
Average loss: 0.3406, Accuracy: 8842/10000 (88.42%)

Time: 450.59966111183167
=======================
```



Appendix

```
model_state = torch.load('resnet164_100_best_181Epochs_Step30_PreActivations_lr0.1_Best.ckpt')
model.load_state_dict(model_state)
print ('100 Percent \nBest ResNet181 model - Test Accuracy:')
acc = test(args, model, device, test_loader, class_criterion_eval)
total_time = end_time - start_time
print ('Time: {}'.format(total_time))
print ('======================')
x_epochs = range(0, args.epochs)
plot_acc_valtest(x_epochs, 100 - np.array(record_val_acc), 100 - np.array(record_test_acc))
```

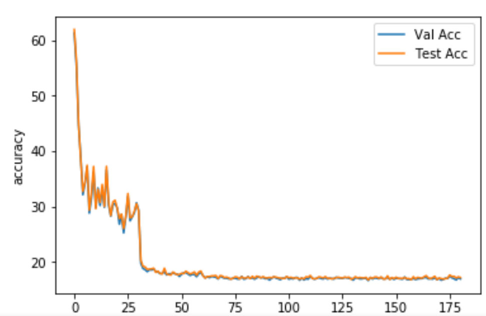
```
100 Percent
Best ResNet181 model - Test Accuracy:
Average loss: 0.3159, Accuracy: 9258/10000 (92.58%)

Time: 15605.456568479538
======================
```



CIFAR10
0.4 of dataset

```
args.epochs = 181

for i, perc in enumerate([train_index_filtered_40perc, train_index_filtered_60perc, train_index_filtered_80perc]):
  train_sampler_perc = SubsetRandomSampler(perc)

  train_loader_perc = torch.utils.data.DataLoader(
        train_dataset, batch_size=args.batch_size, sampler=train_sampler_perc, **kwargs)

  model = PreActResNet164Basic().to(device)
  optimizer = optim.SGD(model.parameters(), lr=args.lr, momentum=args.momentum, weight_decay=0.0005)
  scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1)
  best = 0
  record_val_acc = []
  record_test_acc = []
  pred_percent = pred_percent_values[i]

  start_time = time.time()
  for epoch in range(1, args.epochs + 1):
      train(args, model, device, train_loader_perc, optimizer, epoch, class_criterion_train)
#       acc = evaluate(args, model, device, valid_loader, class_criterion_eval)
#       if acc > best:
#         torch.save(model.state_dict(),"resnet164_{}_best_181Epochs_Step30_PreActivations_lr0.1_Best.ckpt".format(pred_percent))
#         best = acc
      print ('\nLoss and Acc | Val Set')
      val_acc = evaluate(args, model, device, valid_loader, class_criterion_eval)
      record_val_acc.append(val_acc)
      if val_acc > best:
        torch.save(model.state_dict(),"resnet164_{}_best_181Epochs_Step30_PreActivations_lr0.1_Best.ckpt".format(pred_percent))
        best = val_acc

      print ('Loss and Acc | Test Set')
      test_acc = test(args, model, device, test_loader, class_criterion_eval)
      record_test_acc.append(test_acc)
      scheduler.step()
  end_time = time.time()

  best_resnet164 = torch.load('resnet164_{}_best_181Epochs_Step30_PreActivations_lr0.1_Best.ckpt'.format(pred_percent))
  model.load_state_dict(best_resnet164)

  print ('{} Percent \nBest model - Test Accuracy:'.format(pred_percent))
  acc = test(args, model, device, test_loader, class_criterion_eval)
  total_time = end_time - start_time
  print ('Time: {}'.format(total_time))
  print ('=====================================\n======================================')
  x_epochs = range(0, args.epochs)
  plot_acc_valtest(x_epochs, 100 - np.array(record_val_acc), 100 - np.array(record_test_acc))
```

```
Time: 15630.942370176315
=====================================
=====================================
```



CIFAR10
0.8 of dataset

```python
args.epochs = 181

for i, perc in enumerate([train_index_filtered_80perc]):
  train_sampler_perc = SubsetRandomSampler(perc)

  train_loader_perc = torch.utils.data.DataLoader(
        train_dataset, batch_size=args.batch_size, sampler=train_sampler_perc, **kwargs)

  model = PreActResNet164Basic().to(device)
  optimizer = optim.SGD(model.parameters(), lr=args.lr, momentum=args.momentum, weight_decay=0.0005)
  scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1)
  best = 0
  record_val_acc = []
  record_test_acc = []
  pred_percent = pred_percent_values[i]

  start_time = time.time()
  for epoch in range(1, args.epochs + 1):
      train(args, model, device, train_loader_perc, optimizer, epoch, class_criterion_train)
#       acc = evaluate(args, model, device, valid_loader, class_criterion_eval)
#       if acc > best:
#         torch.save(model.state_dict(),"resnet164_{}_best_181Epochs_Step30_PreActivations_lr0.1_Best.ckpt".format(pred_percent,
#         best = acc
      print ('\nLoss and Acc | Val Set')
      val_acc = evaluate(args, model, device, valid_loader, class_criterion_eval)
      record_val_acc.append(val_acc)
      if val_acc > best:
        torch.save(model.state_dict(),"resnet164_{}_best_181Epochs_Step30_PreActivations_lr0.1_Best.ckpt".format(pred_percent))
        best = val_acc

      print ('Loss and Acc | Test Set')
      test_acc = test(args, model, device, test_loader, class_criterion_eval)
      record_test_acc.append(test_acc)
      scheduler.step()
  end_time = time.time()

  best_resnet164 = torch.load('resnet164_{}_best_181Epochs_Step30_PreActivations_lr0.1_Best.ckpt'.format(pred_percent))
  model.load_state_dict(best_resnet164)

  print ('{} Percent \nBest model - Test Accuracy:'.format(pred_percent))
  acc = test(args, model, device, test_loader, class_criterion_eval)
  total_time = end_time - start_time
  print ('Time: {}'.format(total_time))
  print ('=====================================\n=====================================')
  x_epochs = range(0, args.epochs)
  plot_acc_valtest(x_epochs, 100 - np.array(record_val_acc), 100 - np.array(record_test_acc))
```

```
Time: 18102.828154325485
=====================================
=====================================
```