Knowledge Extraction with No Observable Data

Jaemin Yoo, Minyong Cho, Taebum Kim, and U Kang

Data Mining Lab.

Seoul National University

NeurIPS 2019

Outline

- **Introduction**
- Proposed Approach
- Experimental Settings
- Experimental Results
- Conclusion

Model's Knowledge

- Consider an ML model in supervised learning
  - Trained for a dataset $\{(x_i, y_i) | \; i = 1, 2, \dots\}$
  - Learned $p(y|x)$ of a label y given a feature x

- It must have some knowledge about the data
  - *How much labels y_1 and y_2 are related*
  - *How much x is close to y_1 than to y_2*
  - *How much x_1 and x_2 are close to each other*
  - *…*

Knowledge Distillation

- To transfer a model's knowledge to another
  - Given a trained (teacher) model M_1
  - Given a target (student) model M_2
  - Feed a feature vector x_i to produce $\hat{y}_i = M_1(x_i)$
  - Train M_2 using $\hat{y}_i$ as labels instead of true y_i

- Why does it work?
  - $\hat{y}$ contains richer information than one-hot y
  - $\hat{y}$ represents the knowledge of M_1 to be transferred

Knowledge without Data

- What happens when there are no data?
- Knowledge cannot be distilled
  - We cannot feed feature vectors to M_1
  - We cannot generate predictions of M_1
- We have no ideas about M_1's knowledge

- The solution is knowledge extraction!

Estimating Data

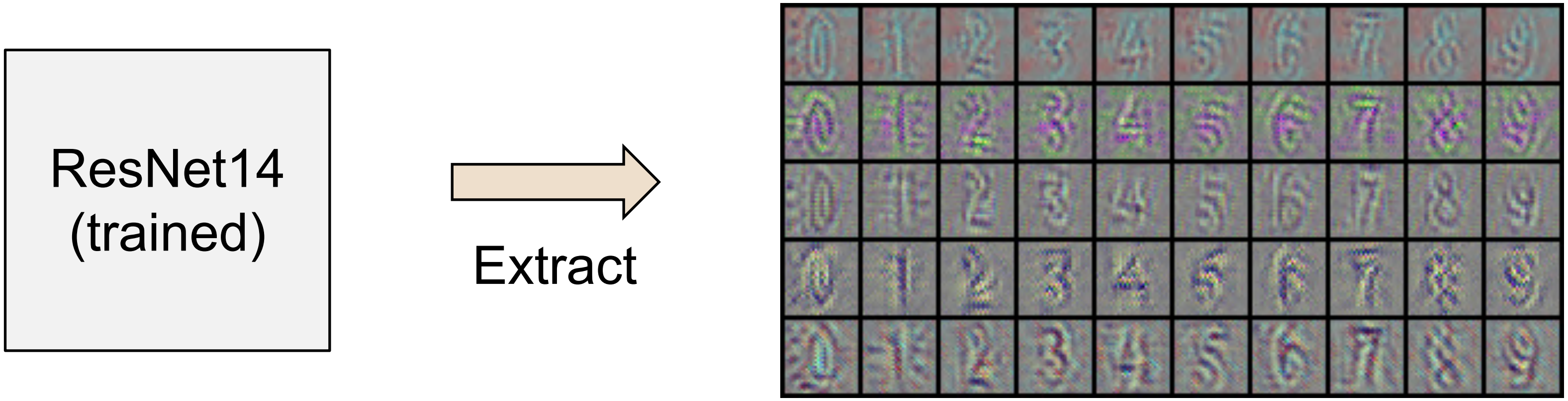
- **Given**
  - A trained model M which maps x to y
- **Estimate**
  - The unknown distribution $p(x)$ of data points
- **Such that**
  - $p(x)$ is useful for distilling M's knowledge

Knowledge Extraction

- **Given**
  - A trained model M which maps x to y
- **Generate**
  - A set $\mathcal{D} = \{(x_i, y_i) | \, i = 1, 2, \dots\}$ of artificial data
- **Such that**
  - Every y_i is a (one-hot or soft) label vector
  - Every x_i has a high conditional probability $p(x_i | y_i)$
  - $\mathcal{D}$ is useful for distilling M's knowledge

Overview

- What does exacted knowledge look like?
- We are given a pre-trained ResNet14
  - Trained for the SVHN dataset of street digit images
- Our model generates the following images:



ResNet14 (trained)

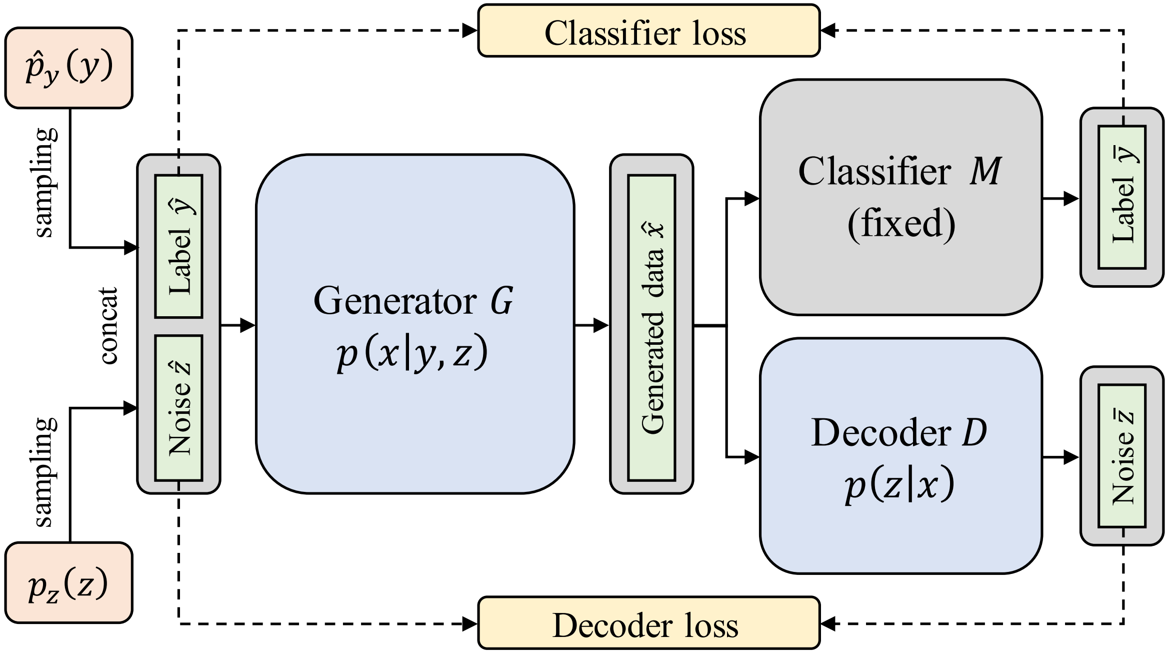
Extract

Outline

- Introduction
- **Proposed Approach**
- Experimental Settings
- Experimental Results
- Conclusion

Overall Structure

- KegNet (Knowledge Extraction with Generative Networks)
  - Consists of three types of neural networks
  - *Generator*, *classifier*, and *decoder* networks



Motivation (1)

- We introduce a latent variable $z \in \mathbb{R}^d$
- Our objective is to generate a dataset $\mathcal{D}$

$$\mathcal{D} = \left\{ \mathrm{argmax}_{\hat{x}}\, p(\hat{x} | \hat{y}, \hat{z}) \,\middle|\, \hat{y} \sim \hat{p}_y(y) \text{ and } \hat{z} \sim p_z(z) \right\}$$

  - $\hat{p}_y$ and p_z are proposed distributions for $\hat{y}$ and z

Motivation (2)

- We approximate the argmax function as

$$\mathrm{argmax}_{\hat{x}}\, p(\hat{x}|\hat{y},\hat{z}) \approx \mathrm{argmax}_{\hat{x}}(\log p(\hat{y}|\hat{x}) + \log p(\hat{z}|\hat{x}))$$

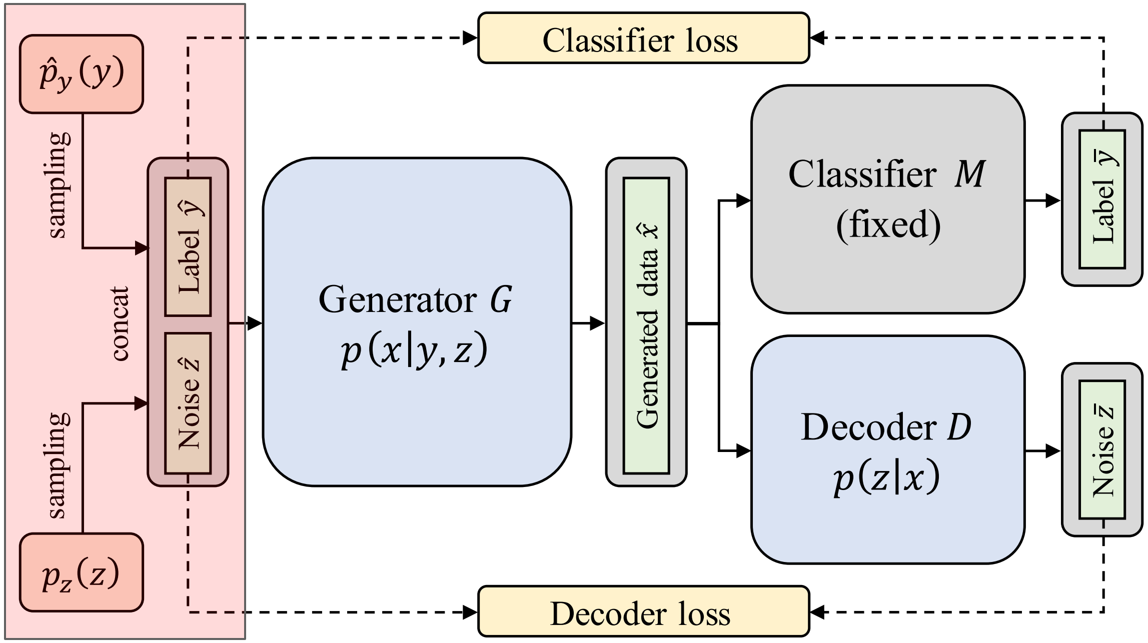
- Then, our model can be optimized as
  - Sampling $\hat{y}$ and $\hat{z}$ from $\hat{p}_y$ and p_z, resp.
  - Generating $\hat{x}$ from sampled $\hat{y}$ and $\hat{z}$
  - Reconstructing $\hat{y}$ from $\hat{x}$ (max. $p(\hat{y}|\hat{x})$)
  - Reconstructing $\hat{z}$ from $\hat{x}$ (max. $p(\hat{z}|\hat{x})$)

Training Process in Detail

- Sample $\hat{y}$ and $\hat{z}$ from simple distributions
- Convert variables by deep neural networks
  - Generator (to learn): $(\hat{y}, \hat{z}) \rightarrow \hat{x}$
  - Decoder (to learn): $\hat{x} \rightarrow \bar{z}$
  - Classifier (given and fixed): $\hat{x} \rightarrow \bar{y}$
- Train all networks my minimizing two losses
  - Classifier loss: the distance $\hat{y} \leftrightarrow \bar{y}$
  - Decoder loss: the distance $\hat{z} \leftrightarrow \bar{z}$

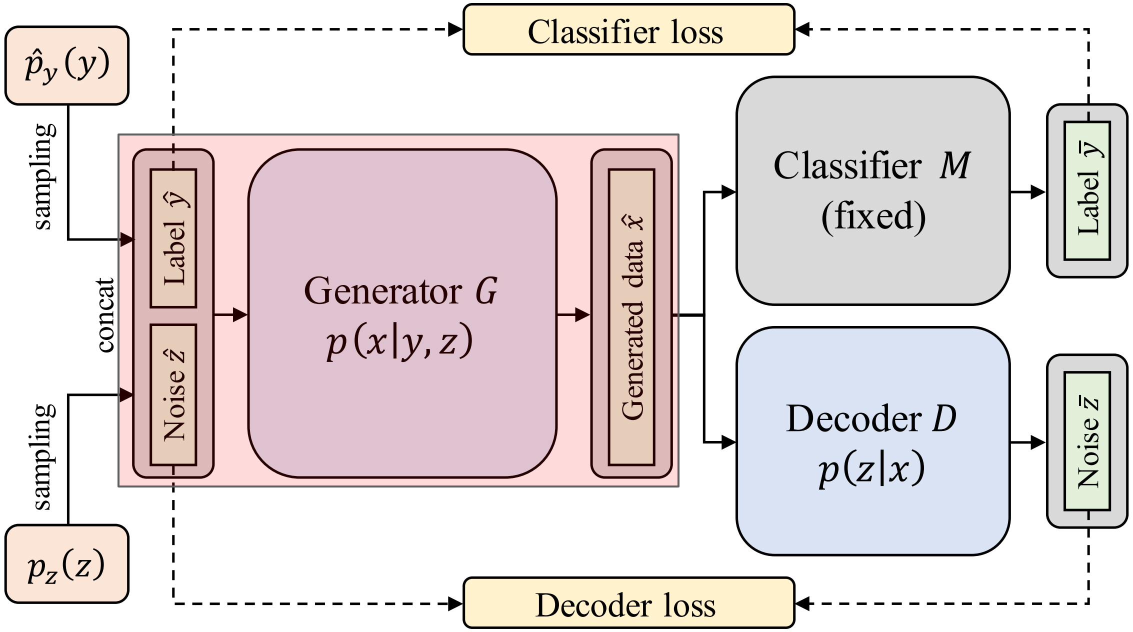
Sampling Variables

- Remember that we have no observable data
- We sample $\hat{y}$ and $\hat{z}$ from distributions $\hat{p}_y$ and p_z
  - Categorical and Gaussian distributions, resp.



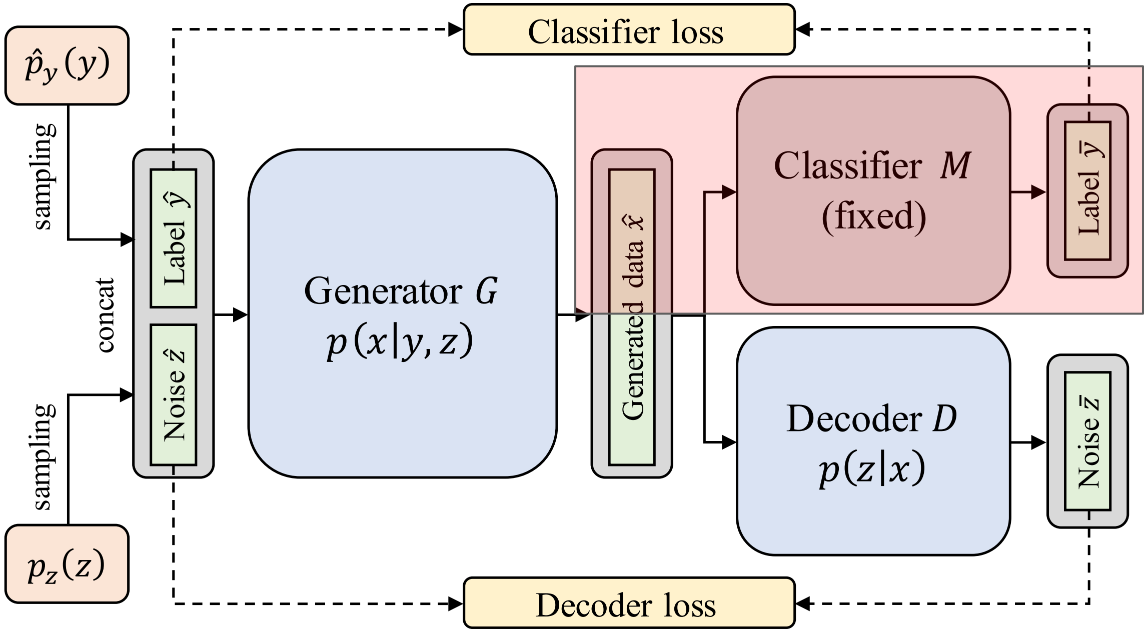
Generator Network

- A generator network generates $\hat{x}$ from $\hat{y}$ and $\hat{z}$
- Its structure is based on DCGAN and ACGAN
  - Transposed convolutional layers and dense layers



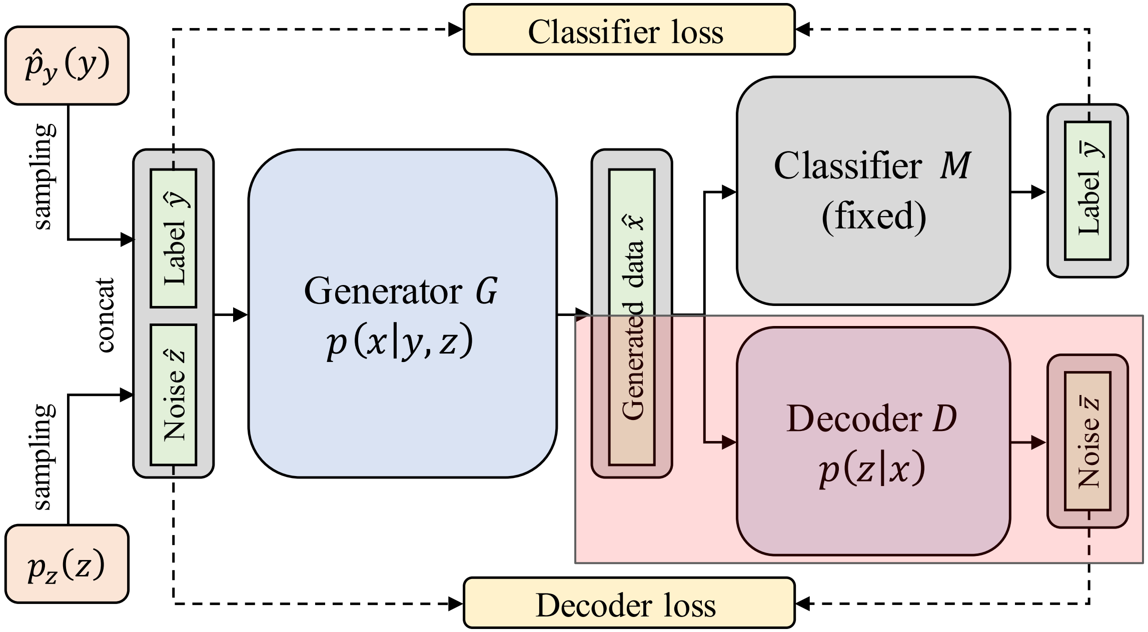
Classifier Network

- The given network works here as evidence
- It reconstructs given $\hat{y}$ based on its knowledge
  - This part is fixed (although it passes back-props)



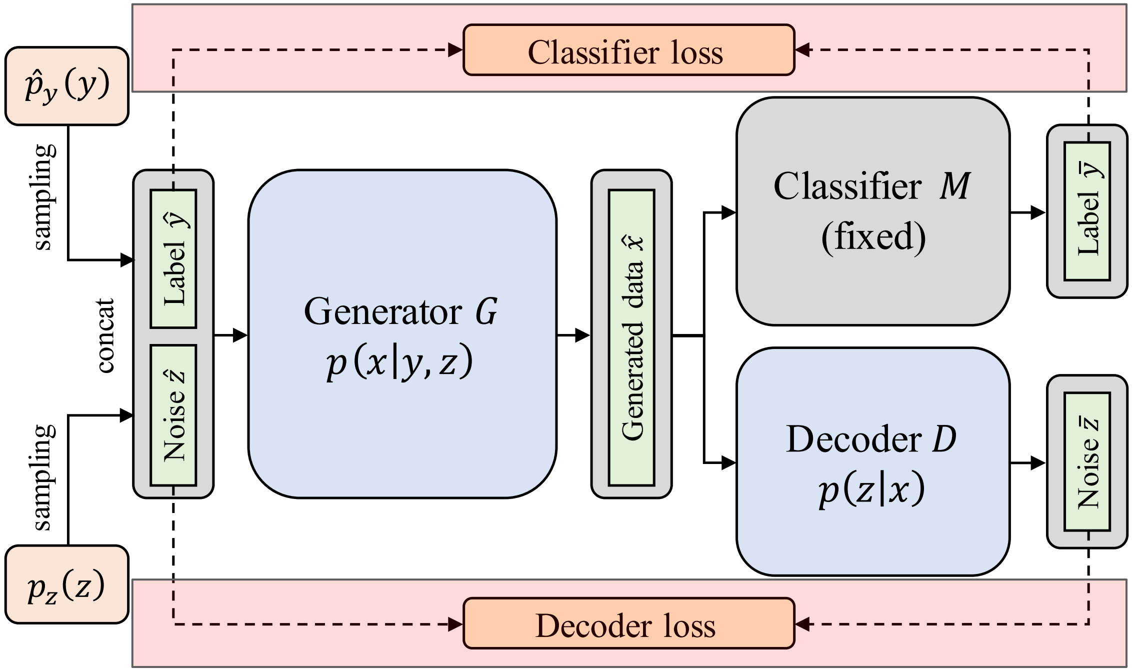
Decoder Network

- A decoder network extracts given $\hat{z}$ from $\hat{x}$
- Its structure is a simple multilayer perceptron
  - It solves the regression problem which is difficult



Reconstruction Losses

- Two reconstruction losses: $\hat{y} \leftrightarrow \bar{y}$ and $\hat{z} \leftrightarrow \bar{z}$
  - Loss for y: cross entropy between probabilities
  - Loss for z: Euclidean distance between vectors



Data Diversity

- One problem exists in the current structure
  - *The generated data have insufficient diversity!*
- Diversity of data is important to our problem
  - The model should distill its knowledge to others
  - The dataset should cover a large data space
  - It will activate many combinations of neurons

Diversity Loss

- In each batch $\mathcal{B}$, we calculate a new loss

$$l_{\text{div}}(\mathcal{B}) = \exp\left(-\sum_{(\hat{z}_1,\hat{x}_1)}\sum_{(\hat{z}_2,\hat{x}_2)}\|\hat{z}_1-\hat{z}_2\|\cdot d(\hat{x}_1,\hat{x}_2)\right)$$

  - $d(\cdot)$ is a distance function between two x's
- Includes distances between all pairs of x's
  - But, it is multiplied by $\|\hat{z}_1-\hat{z}_2\|$
  - When z's are distant, then x's should be distant too

Overall Loss Function

- The overall loss function is given as follows:

$$l(\mathcal{B}) = \sum_{(\hat{y},\hat{z})} \left(l_{\text{cls}}(\hat{y}, \hat{z}) + \alpha l_{\text{dec}}(\hat{y}, \hat{z}) \right) + \beta l_{\text{div}}(\mathcal{B})$$

  - ❑ l_{cls} denotes the classification loss
  - ❑ l_{dec} denotes the decoder loss
  - ❑ α and β are hyperparameters adjusting the balance

Outline

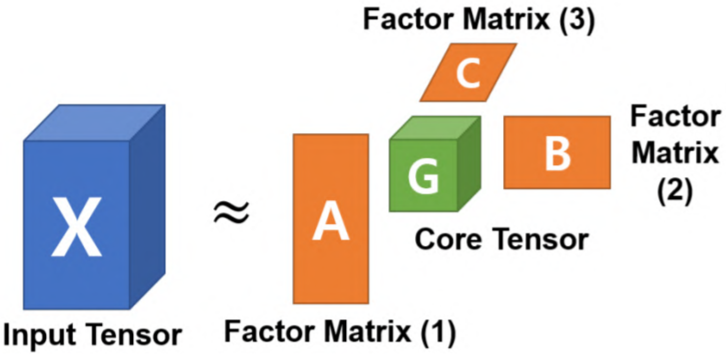
- Introduction
- Proposed Approach
- **Experimental Settings**
- Experimental Results
- Conclusion

Evaluation

- We apply our model to model compression
  - The problem of reducing the size of a network
- **Given** a trained model M
- **Return** a compressed model S
  - S has fewer parameters than M has
  - S shows comparable accuracy to that of M

Tucker Decomposition

- Use Tucker decomposition for compression
  - Factorizes a large tensor into low-rank tensors
  - Has been applied to compress CNNs or RNNs
- Compression by Tucker
  - Initialize a new network with decomposed weights
  - Fine-tune the new network with training data

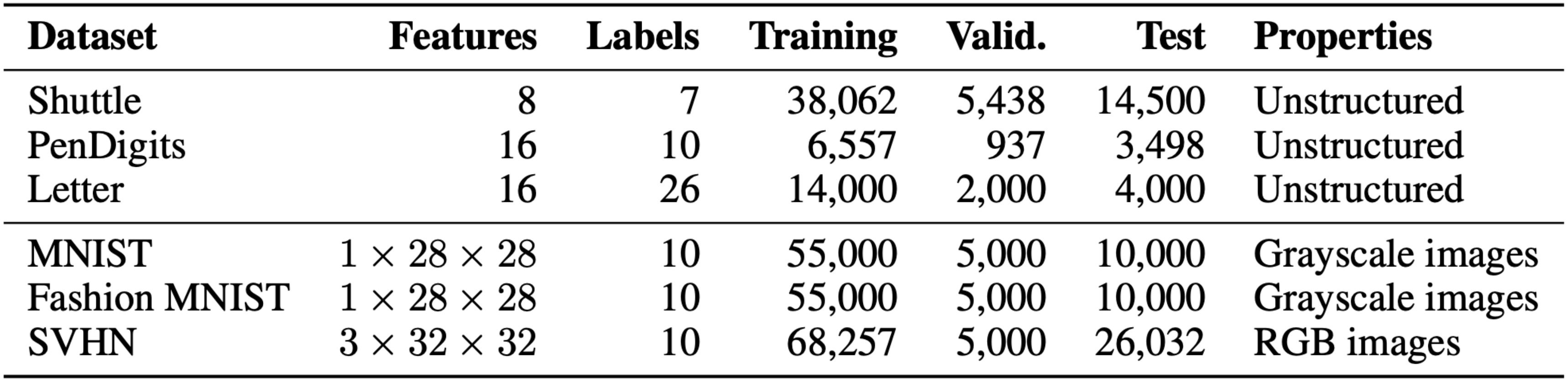


Baseline Approaches

- In our case, we modify the fine-tuning step
  - Because we have no training data available
- We propose three baseline approaches
  - Tucker (T) does not fine-tuning at all
  - T+Uniform estimates p_x as the uniform dist.
  - T+Normal estimates p_x as the normal dist.
- KegNet uses artificial data in fine-tuning
  - 5 generators are trained to produce data

Datasets

- We use two kinds of datasets in experiments
  - Unstructured datasets from the UCI repo.
  - Famous Image datasets for classification



Dataset	**Features**	**Labels**	**Training**	**Valid.**	**Test**	**Properties**
Shuttle	8	7	38,062	5,438	14,500	Unstructured
PenDigits	16	10	6,557	937	3,498	Unstructured
Letter	16	26	14,000	2,000	4,000	Unstructured
MNIST	$1 \times 28 \times 28$	10	55,000	5,000	10,000	Grayscale images
Fashion MNIST	$1 \times 28 \times 28$	10	55,000	5,000	10,000	Grayscale images
SVHN	$3 \times 32 \times 32$	10	68,257	5,000	26,032	RGB images

Target Classifiers

- We use classifiers according to the datasets
  - These classifiers are our targets of compression
- Unstructured datasets
  - Multilayer perceptrons of 10 layers
  - 128 units, ELU activations and dropouts
- Image datasets
  - LeNet5 for MNIST
  - ResNet14 for Fashion MNIST and SVHN

Outline

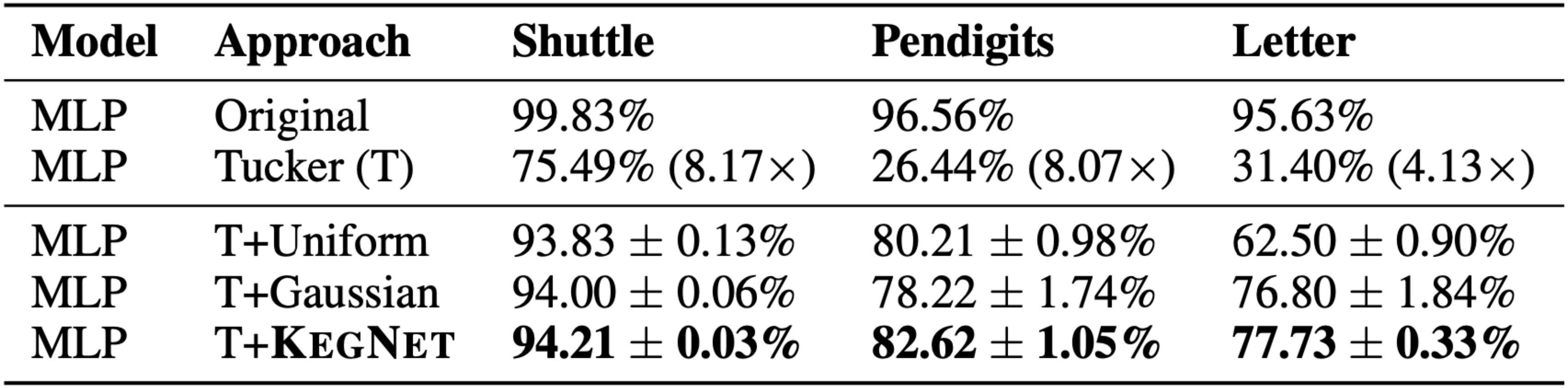
- Introduction
- Proposed Approach
- Experimental Settings
- **Experimental Results**
- Conclusion

Summary

- Three ways of experiments are done
- Quantitative results
  - *Done for the unstructured & image datasets*
  - Compare accuracy and compression ratios
- Qualitative results
  - *Done for the image datasets*
  - Visualize generated data changing $\hat{y}$ and $\hat{z}$

Quantitative Results (1)

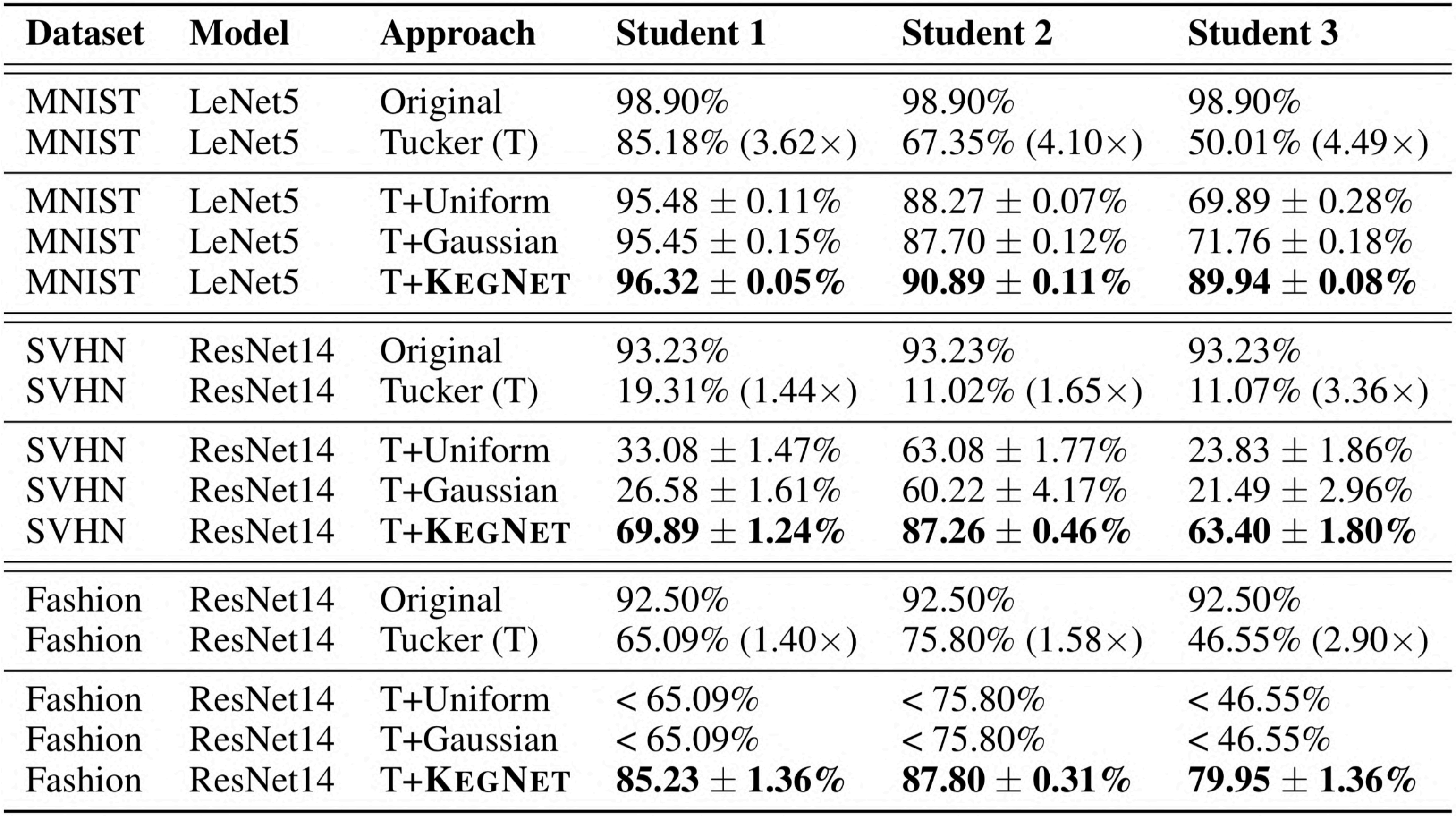
- KegNet outperforms the baselines consistently
- The compression ratios are between 4× and 8×
- T+Gaussian works relatively well
  - Because the features are already standardized
  - Even a Gaussian covers most of the feature space



Model	Approach	Shuttle	Pendigits	Letter
MLP	Original	99.83%	96.56%	95.63%
MLP	Tucker (T)	75.49% (8.17×)	26.44% (8.07×)	31.40% (4.13×)
MLP	T+Uniform	93.83 ± 0.13%	80.21 ± 0.98%	62.50 ± 0.90%
MLP	T+Gaussian	94.00 ± 0.06%	78.22 ± 1.74%	76.80 ± 1.84%
MLP	T+KEGNET	**94.21 ± 0.03%**	**82.62 ± 1.05%**	**77.73 ± 0.33%**

Quantitative Results (2)

- Results are much better in the image datasets



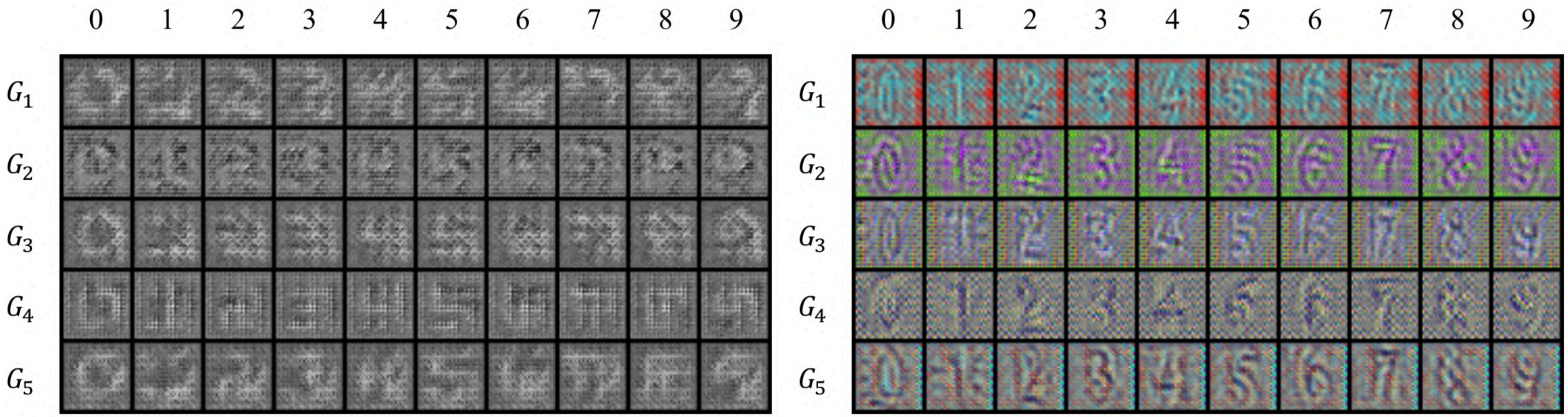
Dataset	Model	Approach	Student 1	Student 2	Student 3
MNIST	LeNet5	Original	98.90%	98.90%	98.90%
MNIST	LeNet5	Tucker (T)	85.18% (3.62×)	67.35% (4.10×)	50.01% (4.49×)
MNIST	LeNet5	T+Uniform	95.48 ± 0.11%	88.27 ± 0.07%	69.89 ± 0.28%
MNIST	LeNet5	T+Gaussian	95.45 ± 0.15%	87.70 ± 0.12%	71.76 ± 0.18%
MNIST	LeNet5	T+**KEGNET**	**96.32 ± 0.05%**	**90.89 ± 0.11%**	**89.94 ± 0.08%**
SVHN	ResNet14	Original	93.23%	93.23%	93.23%
SVHN	ResNet14	Tucker (T)	19.31% (1.44×)	11.02% (1.65×)	11.07% (3.36×)
SVHN	ResNet14	T+Uniform	33.08 ± 1.47%	63.08 ± 1.77%	23.83 ± 1.86%
SVHN	ResNet14	T+Gaussian	26.58 ± 1.61%	60.22 ± 4.17%	21.49 ± 2.96%
SVHN	ResNet14	T+**KEGNET**	**69.89 ± 1.24%**	**87.26 ± 0.46%**	**63.40 ± 1.80%**
Fashion	ResNet14	Original	92.50%	92.50%	92.50%
Fashion	ResNet14	Tucker (T)	65.09% (1.40×)	75.80% (1.58×)	46.55% (2.90×)
Fashion	ResNet14	T+Uniform	< 65.09%	< 75.80%	< 46.55%
Fashion	ResNet14	T+Gaussian	< 65.09%	< 75.80%	< 46.55%
Fashion	ResNet14	T+**KEGNET**	**85.23 ± 1.36%**	**87.80 ± 0.31%**	**79.95 ± 1.36%**

Quantitative Results (3)

- Two main observations from the results
- Large improvements in complicated datasets
  - MNIST < Fashion MNIST < SVHN
  - Competitors even can decrease the accuracy
  - Because the manifolds are difficult to capture
- Large improvements in high compression rates
  - Because they require better samples

Qualitative Results (1)

- Generated images contain recognizable digits
- SVHN looks more clear than MNIST
  - Because the manifold of SVHN is more predictable
  - The digits of MNIST are more diverse (handwritten)

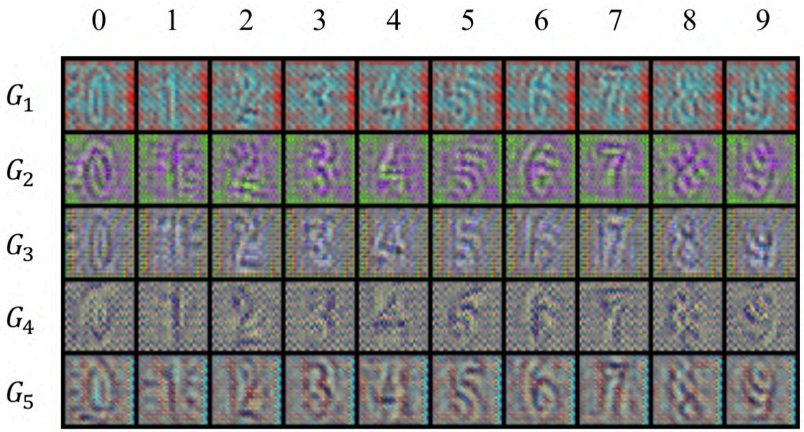


(a) MNIST ($z = \mathbf{0}$).

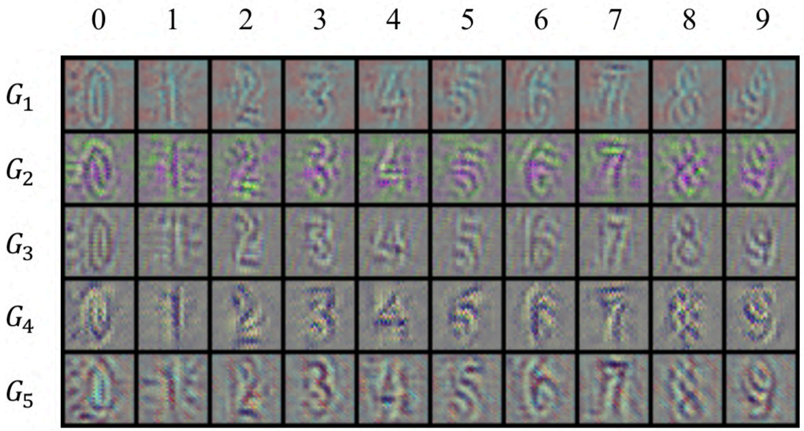
(b) SVHN ($z = \mathbf{0}$).

Qualitative Results (2)

- The variable z gives randomness to images
  - The images seem noisy when $z = 0$
  - The images seem organized when averaged by z
- The 5 generators have different properties



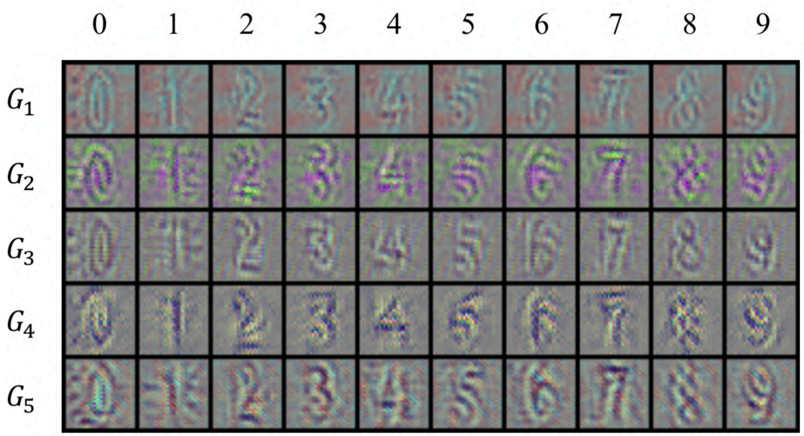
(b) SVHN ($z = \mathbf{0}$).



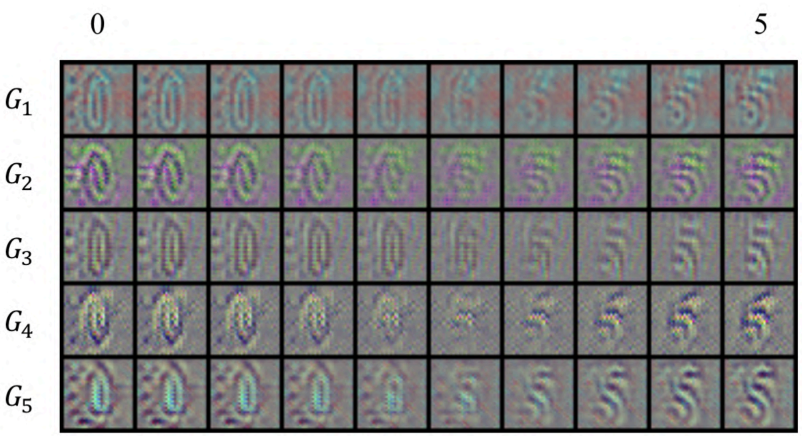
(c) SVHN (averaged by z).

Qualitative Results (3)

- Our generator can take soft distributions of $\hat{y}$
  - We change $\hat{y}$ from 0 to 5 to see the differences
  - The amount of evidence changes slowly
  - An image becomes like 5 from a certain point



(c) SVHN (averaged by z).



(d) Latent space walking from 0 to 5 in SVHN.

Outline

- Introduction
- Proposed Approach
- Experimental Settings
- Experimental Results
- **Conclusion**

Conclusion

- We propose KegNet for data-free distillation
  - *Knowledge extraction with generative networks*
  - It enables knowledge distillation even without data
- KegNet consists of three deep neural networks
  - Classifier network which is given and fixed
  - Generator network for generating artificial data
  - Decoder network for capturing latent variables
- KegNet outperforms all baselines significantly
  - Experiments on unstructured and image datasets

Thank you !

https://github.com/snudatalab/KegNet