Semi-supervised Learning in Gigantic Image Collections

Rob Fergus, Yair Weiss and Antonio Torralba

Rui Su & Jingheng Chen
"Haggis"
• What if we have 10,000,000 images?????
Outline
Outline

- Background
- Introduction
- Algorithm
- Experiments
- Conclusion
Background
Background

- There’re gigantic quantities of images.
Background

- There’re gigantic quantities of images.
- Effective searching and labeling.
Background

- There’re gigantic quantities of images.
- Effective searching and labeling.
- Diversity of label information:
Background

- There’re gigantic quantities of images.
- Effective searching and labeling.
- Diversity of label information:
  - clean label
Background

- There’re gigantic quantities of images.
- Effective searching and labeling.
- Diversity of label information:
  - clean label
  - noisy label
Background

- There’re gigantic quantities of images.
- Effective searching and labeling.
- Diversity of label information:
  - clean label
  - noisy label
  - no label
Background

- There’re gigantic quantities of images.
- Effective searching and labeling.
- Diversity of label information:
  - clean label
  - noisy label
  - no label
- How to handle this spectrum of label information?
Background

• There’re gigantic quantities of images.

• Effective searching and labeling.

• Diversity of label information:
  • clean label
  • noisy label
  • no label

• How to handle this spectrum of label information?
  • semi-supervised learning (SSL)
Semi-supervised learning
Semi-supervised learning

- Lots of *unlabeled* data and few *labeled* data.
Semi-supervised learning

- Lots of \textit{unlabeled} data and few \textit{labeled} data.
- Model the \textit{density} of the data and measure \textit{proximities}.
Semi-supervised learning

- Lots of **unlabeled** data and few **labeled** data.
- Model the **density** of the data and measure **proximities**
- Popular approaches: **graph Laplacian**.
Semi-supervised learning

- Lots of **unlabeled** data and few **labeled** data.
- Model the **density** of the data and measure **proximities**
- Popular approaches: **graph Laplacian**.
  - manipulation of an $n \times n$ Laplacian matrix.
Semi-supervised learning

- Lots of **unlabeled** data and few **labeled** data.
- Model the **density** of the data and measure **proximities**

- Popular approaches: **graph Laplacian**.
  - manipulation of an $n \times n$ Laplacian matrix.
  - $O(n^3)$ complexity $\Rightarrow$ impractical for large datasets
Semi-supervised learning

• Lots of **unlabeled** data and few **labeled** data.
• Model the **density** of the data and measure **proximities**

• Popular approaches: **graph Laplacian**.
  • manipulation of an \( n \times n \) Laplacian matrix.
  • \( O(n^3) \) complexity \( \Rightarrow \) impractical for large datasets

• **Efficient** semi-supervised learning.
Efficient semi-supervised learning
Efficient semi-supervised learning

- Calculate the Laplacian for a \textit{smaller graph}
Efficient semi-supervised learning

- Calculate the Laplacian for a *smaller graph*
- Different construction methods
Efficient semi-supervised learning

- Calculate the Laplacian for a *smaller graph*
- Different construction methods
- Graph Laplacian *change dramatically* with different backbone construction methods.
Efficient semi-supervised learning

- Calculate the Laplacian for a smaller graph
- Different construction methods
- Graph Laplacian change dramatically with different backbone construction methods.
- Take the limit as the number of points goes to infinity.
Algorithm
Algorithm

• Graph setting
• From eigenvectors to eigenfunctions
Graph setting
Graph setting

• Labeled dataset: \((X_l, Y_l) = \{(x_1, y_1), \ldots, (x_l, y_l)\}\)

• Unlabeled dataset: \(X_u = \{x_{l+1}, \ldots, x_n\}\)
Graph setting

- Labeled dataset: \((X_l, Y_l) = \{(x_1, y_1), \ldots, (x_l, y_l)\}\)

- Unlabeled dataset: \(X_u = \{x_{l+1}, \ldots, x_n\}\)
Graph Laplacian
Graph Laplacian

- Graph: $G = (V, E)$
Graph Laplacian

• Graph: $G = (V, E)$

• Vertices $V$: data points $\{x_1, \ldots, x_n\}$
Graph Laplacian

• **Graph:** $G = (V, E)$

• **Vertices** $V$: data points $\{x_1 \ldots x_n\}$

• **Edges** $E$: represented by weight matrix
Graph Laplacian

- Graph: \( G = (V, E) \)
- Vertices \( V \): data points \( \{x_1, \ldots, x_n\} \)
- Edges \( E \): represented by weight matrix
- Weight Matrix \( W \): \( n \times n \) matrix
Graph Laplacian
Graph Laplacian
Graph Laplacian

- “boundary” points: labeled
- “interior” points: unlabeled
- Weighted edges
Graph Laplacian

- Weight:

- What does it mean?

- Nearby points in Euclidean space are assigned large edge weight
Graph Laplacian

• Weight: \( w_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\varepsilon^2}\right) \)

• What does it mean?

• Nearby points in Euclidean space are assigned large edge weight
Graph Laplacian

- Weight: \( w_{ij} = \exp(-\|x_i - x_j\|^2 / 2\varepsilon^2) \)
Graph Laplacian

- Weight: \( w_{ij} = \exp(-\|x_i - x_j\|^2 / 2\varepsilon^2) \)

- What does it mean?
Graph Laplacian

- Weight: \( w_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\varepsilon^2}\right) \)

- What does it mean?

- Nearby points in Euclidean space are assigned large edge weight
Combinatorial graph Laplacian
Combinatorial graph Laplacian

- Diagonal Matrix $D$:
Combinatorial graph Laplacian

- Diagonal Matrix $D$:
  \[ D_{ii} = \sum_j W_{ij} \]
Combinatorial graph Laplacian

- Diagonal Matrix $D$: $D_{ii} = \sum_j W_{ij}$
Combinatorial graph Laplacian

- Diagonal Matrix $D$ : $D_{ii} = \sum_j W_{ij}$
Combinatorial graph Laplacian

- Diagonal Matrix $D$ : $D_{ii} = \sum_j W_{ij}$
Combinatorial graph Laplacian

- Diagonal Matrix $D$:
  \[ D_{ii} = \sum_j W_{ij} \]
Combinatorial graph Laplacian

• Diagonal Matrix $D$ : $D_{ii} = \sum_j W_{ij}$

• Combinatorial graph Laplacian:
Combinatorial graph Laplacian

- Diagonal Matrix $D$: $D_{ii} = \sum_j W_{ij}$

- Combinatorial graph Laplacian:
  - $L = D - W$
Combinatorial graph Laplacian

• Diagonal Matrix $\mathbf{D}$:  
  \[ D_{ii} = \sum_j W_{ij} \]

• Combinatorial graph Laplacian:
  
  \[ \mathbf{L} = \mathbf{D} - \mathbf{W} \]

• Also called unnormalized Laplacian
Smoothness
Smoothness

- Graph Laplacian $L$ is used to define a smoothness operator that takes into account the unlabeled data.
Smoothness

- Graph Laplacian $L$ is used to define a smoothness operator that takes into account the unlabeled data

$$f^T L f = \frac{1}{2} \sum_{i,j} W_{ij} (f(i) - f(j))^2$$
Smoothness

- Graph Laplacian $L$ is used to define a smoothness operator that takes into account the unlabeled data

$$f^T L f = \frac{1}{2} \sum_{i,j} W_{ij} (f(i) - f(j))^2$$
Smoothness

- Graph Laplacian $L$ is used to define a smoothness operator that takes into account the unlabeled data

$$f^T L f = \frac{1}{2} \sum_{i,j} W_{ij} (f(i) - f(j))^2$$

- What is Smoothness?
Smoothness

- Graph Laplacian $L$ is used to define a smoothness operator that takes into account the unlabeled data

$$f^T L f = \frac{1}{2} \sum_{i,j} W_{ij} (f(i) - f(j))^2$$

- What is Smoothness?

- Distribution of data *changes slowly in one category and greatly between two separated categories*
Smoothness

- Graph Laplacian $L$ is used to define a smoothness operator that takes into account the unlabeled data

$$f^T L f = \frac{1}{2} \sum_{i,j} W_{ij} (f(i) - f(j))^2$$

- What is Smoothness?

- Distribution of data **changes slowly in one category and greatly between two separated categories**

- Also called: quadratic energy function
Minimization
Minimization

• Motivation: Minimize the smoothness (energy) and the training loss
Minimization

- Motivation: Minimize the smoothness (energy) and the training loss

\[ J(f) = f^T Lf + \sum_{i=1}^{l} \lambda (f(i) - y_i)^2 = f^T Lf + (f - y)^T \Lambda (f - y) \]
Minimization

• **Motivation:** Minimize the smoothness (energy) and the training loss

\[ J(f) = f^T Lf + \sum_{i=1}^{l} \lambda (f(i) - y_i)^2 = f^T Lf + (f - y)^T \Lambda (f - y) \]
Minimization

- Motivation: Minimize the smoothness (energy) and the training loss

\[ J(f) = f^T Lf + \sum_{i=1}^{l} \lambda(f(i) - y_i)^2 = f^T Lf + (f - y)^T \Lambda(f - y) \]
Minimization

- Motivation: Minimize the smoothness (energy) and the training loss

\[ J(f) = f^T Lf + \sum_{i=1}^{l} \lambda (f(i) - y_i)^2 = f^T Lf + (f - y)^T \Lambda (f - y) \]
Minimization

• **Motivation:** Minimize the smoothness (energy) and the training loss

\[
J(f) = f^T Lf + \sum_{i=1}^{l} \lambda (f(i) - y_i)^2 = f^T Lf + (f - y)^T \Lambda (f - y)
\]
Minimization

- Motivation: Minimize the smoothness (energy) and the training loss

\[
J(f) = f^T L f + \sum_{i=1}^{l} \lambda (f(i) - y_i)^2 = f^T L f + (f - y)^T \Lambda (f - y)
\]
Minimization

- Motivation: Minimize the smoothness (energy) and the training loss

\[ J(f) = f^T Lf + \sum_{i=1}^{l} \lambda(f(i) - y_i)^2 = f^T Lf + (f - y)^T \Lambda(f - y) \]
Minimization

- Motivation: Minimize the smoothness (energy) and the training loss

\[ J(f) = f^T Lf + \sum_{i=1}^{l} \lambda(f(i) - y_i)^2 = f^T Lf + (f - y)^T \Lambda (f - y) \]
Minimization

• Motivation: Minimize the smoothness (energy) and the training loss

\[ J(f) = f^T Lf + \sum_{i=1}^{l} \lambda (f(i) - y_i)^2 = f^T Lf + (f - y)^T \Lambda (f - y) \]
Minimization

• Motivation: Minimize the smoothness (energy) and the training loss

\[ J(f) = f^T Lf + \sum_{i=1}^{l} \lambda(f(i) - y_i)^2 = f^T Lf + (f - y)^T \Lambda(f - y) \]
• Motivation: Minimize the smoothness (energy) and the training loss

\[ J(f) = f^T Lf + \sum_{i=1}^{l} \lambda (f(i) - y_i)^2 = f^T Lf + (f - y)^T \Lambda (f - y) \]

• Minimizer:
Minimization

• Motivation: Minimize the smoothness (energy) and the training loss

\[ J(f) = f^T Lf + \sum_{i=1}^{l} \lambda(f(i) - y_i)^2 = f^T Lf + (f - y)^T \Lambda(f - y) \]

• Minimizer: \((L + \Lambda)f = \Lambda y\)
Minimization

• Motivation: Minimize the smoothness (energy) and the training loss

\[ J(f) = f^T Lf + \sum_{i=1}^{l} \lambda(f(i) - y_i)^2 = f^T Lf + (f - y)^T \Lambda(f - y) \]

• Minimizer: \((L + \Lambda)f = \Lambda y\)

• Still has problem
Eigenvectors
Eigenvectors

- Problem: It requires solving an $n \times n$ system of linear equations.
Eigenvectors

• Problem: It requires solving an $n \times n$ system of linear equations.

• $n$ can be very large
Eigenvectors

• Problem: It requires solving an $n \times n$ system of linear equations.

• $n$ can be very large

• Solution: Dimension can be reduced by working with a small number of eigenvectors of Laplacian
Eigenvectors

• Problem: It requires solving an $n \times n$ system of linear equations.

• $n$ can be very large

• Solution: Dimension can be reduced by working with a small number of eigenvectors of Laplacian

• Eigenvectors: just as what you did in PCA
Eigenvectors

- Problem: It requires solving an $n \times n$ system of linear equations.
- $n$ can be very large
- Solution: Dimension can be reduced by working with a small number of eigenvectors of Laplacian
- Eigenvectors: just as what you did in PCA
Eigenvectors

- Problem: It requires solving an $n \times n$ system of linear equations.
- $n$ can be very large
- Solution: Dimension can be reduced by working with a small number of eigenvectors of Laplacian
- Eigenvectors: just as what you did in PCA

$$L\phi_i = \sigma_i D\phi_i$$
Eigenvector
Eigenvector

- Smoothness of eigenvector:
Eigenvector

• Smoothness of eigenvector: $\Phi_i^T L \Phi_i = \sigma_i$
Eigenvector

• Smoothness of eigenvector: \( \Phi_i^T L \Phi_i = \sigma_i \)

• Eigenvalue : smaller means smoother
Eigenvector

- Smoothness of eigenvector: $\Phi_i^T L \Phi_i = \sigma_i$
- Eigenvalue $\sigma_i$: smaller means smoother
Eigenvector

- Smoothness of eigenvector: $\Phi_i^T L \Phi_i = \sigma_i$
- Eigenvalue $\sigma_i$: smaller means smoother
- Require $f$ to be the form:
Eigenvector

- Smoothness of eigenvector: \( \Phi_i^T L \Phi_i = \sigma_i \)
- Eigenvalue \( \sigma_i \): smaller means smoother
- Require f to be the form: \( f = U\alpha \)
Eigenvector

- Smoothness of eigenvector: \( \Phi_i^T L \Phi_i = \sigma_i \)
- Eigenvalue \( \sigma_i \): smaller means smoother
- Require \( f \) to be the form: \( f = U\alpha \)
- \( U \) is a \( n \times k \) matrix whose columns are the \( k \) eigenvectors with smallest eigenvalue
Eigenvector

- Smoothness of eigenvector: $\Phi_i^T L \Phi_i = \sigma_i$
- Eigenvalue $\sigma_i$: smaller means smoother
- Require $f$ to be the form: $f = U \alpha$
- $U$ is a $n \times k$ matrix whose columns are the $k$ eigenvectors with smallest eigenvalue
- Recall the minimizer?
Eigenvector

- Smoothness of eigenvector: $\Phi_i^T L \Phi_i = \sigma_i$
- Eigenvalue $\sigma_i$: smaller means smoother
- Require $f$ to be the form: $f = U \alpha$
- $U$ is a $n \times k$ matrix whose columns are the $k$ eigenvectors with smallest eigenvalue
- Recall the minimizer?
Eigenvector

- Smoothness of eigenvector: $\Phi_i^T L \Phi_i = \sigma_i$
- Eigenvalue $\sigma_i$: smaller means smoother
- Require $f$ to be the form: $f = U\alpha$
- $U$ is a $n \times k$ matrix whose columns are the $k$ eigenvectors with smallest eigenvalue
- Recall the minimizer?
- $(L + U^T \Lambda U)\alpha = U^T \Lambda y$
From eigenvectors to eigenfunctions
From eigenvectors to eigenfunctions

- Problem: Hard to find the eigenvectors
From eigenvectors to eigenfunctions

- Problem: Hard to find the eigenvectors
- Involves diagonalizing a $n \times n$ matrix
From eigenvectors to eigenfunctions

- Problem: Hard to find the eigenvectors
- Involves diagonalizing a n×n matrix
- New idea?
From eigenvectors to eigenfunctions

• Problem: Hard to find the eigenvectors
• Involves diagonalizing a $n \times n$ matrix
• New idea?
• Yes! Sampling is great!
From eigenvectors to eigenfunctions

• Problem: Hard to find the eigenvectors
• Involves diagonalizing a $n \times n$ matrix
• New idea?
• Yes! Sampling is great!
• (i) Assuming the data are *samples* from a distribution $p(x)$
From eigenvectors to eigenfunctions

- Problem: Hard to find the eigenvectors
- Involves diagonalizing a $n \times n$ matrix
- New idea?
- Yes! Sampling is great!
- (i) Assuming the data are samples from a distribution $p(x)$
- (ii) Analyzing the eigenfunctions of the smoothness operator defined by $p(x)$
Eigenfunction
Eigenfunction

- Redefine the weighted smoothness operator
Eigenfunction

• Redefine the weighted smoothness operator
• Previous:
Eigenfunction

- Redefine the weighted smoothness operator
- Previous: \( f^T Lf = \frac{1}{2} \sum_{i,j} W_{ij} (f(i) - f(j))^2 \)
Eigenfunction

• Redefine the weighted smoothness operator

  Previous:  \[ f^T Lf = \frac{1}{2} \sum_{i,j} W_{ij} (f(i) - f(j))^2 \]

  Redefined:
Eigenfunction

- Redefine the weighted smoothness operator

Previous: \[ f^T Lf = \frac{1}{2} \sum_{i,j} W_{ij} (f(i) - f(j))^2 \]

Redefined:

\[ L_p(F) = \frac{1}{2} \int (F(x_1) - F(x_2))^2 W(x_1, x_2) p(x_1) p(x_2) dx_1 x_2 \]
Eigenfunction

- Redefine the weighted smoothness operator

\[ f^T L f = \frac{1}{2} \sum_{i,j} W_{ij} (f(i) - f(j))^2 \]

- Previous:

- Redefined:

\[ L_p(F) = \frac{1}{2} \int (F(x_1) - F(x_2))^2 W(x_1, x_2) p(x_1)p(x_2) dx_1 x_2 \]
Eigenfunction

- Redefine the weighted smoothness operator
- Previous: \[ f^T L f = \frac{1}{2} \sum_{i,j} W_{ij} (f(i) - f(j))^2 \]
- Redefined:
  \[ L_p (F) = \frac{1}{2} \int (F(x_1) - F(x_2))^2 W(x_1,x_2) p(x_1)p(x_2) dx_1x_2 \]
- The value of the eigenfunction is eigenvalue
Eigenfunction

- Redefine the weighted smoothness operator

Previous: \( f^T L f = \frac{1}{2} \sum_{i,j} W_{ij} (f(i) - f(j))^2 \)

Redefined:

\[
L_p(F) = \frac{1}{2} \int (F(x_1) - F(x_2))^2 W(x_1, x_2) p(x_1) p(x_2) \, dx_1 x_2
\]

- The value of the eigenfunction is eigenvalue

- Simplify:
Eigenfunction

• Redefine the weighted smoothness operator

• Previous: \[ f^T Lf = \frac{1}{2} \sum_{i,j} W_{ij} (f(i) - f(j))^2 \]

• Redefined:

\[ L_p(F) = \frac{1}{2} \int (F(x_1) - F(x_2))^2 W(x_1, x_2) p(x_1)p(x_2) dx_1x_2 \]

• The value of the eigenfunction is eigenvalue

• Simplify: \[ L_p(\phi(k)) = \sigma_k \]
Eigenfunction

- Redefine the weighted smoothness operator

Previous:\[ f^T L f = \frac{1}{2} \sum_{i,j} W_{ij} (f(i) - f(j))^2 \]

Redefined:\[ L_p (F) = \frac{1}{2} \int (F(x_1) - F(x_2))^2 W(x_1, x_2) p(x_1)p(x_2) dx_1 x_2 \]

- The value of the eigenfunction is eigenvalue

- Simplify:\[ L_p (\phi(k)) = \sigma_k \]

- Aim : minimize the smoothness or eigenvalue
Eigenfunction
Eigenfunction

- Tremendous advantage: Focus on density function instead of dealing with large data sets
Eigenfunction

- Tremendous advantage: Focus on density function instead of dealing with large data sets
- Example:
Eigenfunction

• Tremendous advantage: Focus on density function instead of dealing with large data sets

• Example:

• Problem of 80 million images
Eigenfunction

- Tremendous advantage: Focus on density function instead of dealing with large data sets

- Example:

- Problem of 80 million images

- Diagonalizing 80 million by 80 million matrix?
Eigenfunction

- Tremendous advantage: Focus on density function instead of dealing with large data sets
- Example:
- Problem of 80 million images
- Diagonalizing 80 million by 80 million matrix?
- Sampled from 32 dimensional Gaussian
Eigenfunction

- Tremendous advantage: Focus on density function instead of dealing with large data sets
- Example:
- Problem of 80 million images
- Diagonalizing 80 million by 80 million matrix?
- Sampled from 32 dimensional Gaussian
- Simply estimate a $32 \times 32$ covariance matrix!

Amazing!
Eigenfunction
Eigenfunction

- Improvement: p(x) can has a product form
Eigenfunction

- Improvement: \( p(x) \) can has a product form
- Rotate the data: \( S = Rx \)
Eigenfunction

- Improvement: $p(x)$ can have a product form
- Rotate the data: $S = Rx$
- Product form:
Eigenfunction

- Improvement: $p(x)$ can have a product form
- Rotate the data: $S = Rx$
- Product form: $p(s) = p(s_1)p(s_2)\ldots p(s_d)$
Eigenfunction

- Improvement: $p(x)$ can has a product form
- Rotate the data: $S = Rx$
- Product form: $p(s) = p(s_1)p(s_2)......p(s_d)$
- Allows us to calculate the eigenfunctions of $L_p$ using only marginal distribution
Eigenfunction

- Improvement: $p(x)$ can have a product form
- Rotate the data: $S = Rx$
- Product form: $p(s) = p(s_1)p(s_2)\ldots p(s_d)$
- Allows us to calculate the eigenfunctions of $L_p$ using only marginal distribution $p(s_i)$. 
Eigenfunction

• Improvement: $p(x)$ can have a product form

• Rotate the data: $S = Rx$

• Product form: $p(s) = p(s_1)p(s_2)\ldots p(s_d)$

• Allows us to calculate the eigenfunctions of $L_p$ using only marginal distribution $p(s_i)$.

• Constrains: $s = Rx$ are as independent as possible
Eigenfunction
Eigenfunction

• Assume the semi-supervised solution is a \textit{linear combination} of only the single-coordinate eigenfunction
Eigenfunction

• Assume the semi-supervised solution is a *linear combination* of only the single-coordinate eigenfunction

• We now have k functions whose value is given at a set of discrete points for each coordinate
Eigenfunction

• Assume the semi-supervised solution is a linear combination of only the single-coordinate eigenfunction

• We now have k functions $\Phi_k(x)$ whose value is given at a set of discrete points for each coordinate
• Assume the semi-supervised solution is a *linear combination* of only the single-coordinate eigenfunction

• We now have $k$ functions $\Phi_k(x)$ whose value is given at a set of discrete points for each coordinate

• Use linear interpolation in 1D to interpolate
Eigenfunction

• Assume the semi-supervised solution is a **linear combination** of only the single-coordinate eigenfunction

• We now have $k$ functions $\Phi_k(x)$ whose value is given at a set of discrete points for each coordinate

• Use linear interpolation in 1D to interpolate $\Phi(x)$
Toy example
Toy example
Toy example
Toy example
Toy example
Experiments
Experiments

- Comparison with other algorithm
- Subset of CIFAR
- Tiny Images illustration
Comparison with Nystrom
Comparison with Nystrom

- **Case 1**: the landmarks do not adequately summarize the density
Comparison with Nystrom

- Case 1: the landmarks do not adequately summarize the density

![Data points and landmarks](image-url)
Comparison with Nystrom

- **Case 1:** The landmarks do not adequately summarize the density
Comparison with Nystrom

- **Case 1**: the landmarks do not adequately summarize the density
Comparison with Nystrom

- **Case 1:** the landmarks do not adequately summarize the density
Comparison with Nystrom

- **Case 1**: the landmarks do not adequately summarize the density
Comparison with Nystrom

- **Case I:** the landmarks do not adequately summarize the density

- **Case II:** the density is far from a product form
Comparison with Nystrom

- **Case I:** the landmarks do not adequately summarize the density

- **Case II:** the density is far from a product form
Comparison with Nystrom

- **Case I:** the landmarks do not adequately summarize the density
- **Case II:** the density is far from a product form
Comparison with Nystrom

- **Case I:** the landmarks do not adequately summarize the density

- **Case II:** the density is far from a product form
Comparison with Nystrom

- **Case I:** the landmarks do not adequately summarize the density

- **Case II:** the density is far from a product form
CIFAR
CIFAR

• A subset of classes of the Tiny Images dataset
CIFAR

- A subset of classes of the Tiny Images dataset
- Given a keyword and image: positive or negative
CIFAR

- A subset of classes of the Tiny Images dataset
- Given a keyword and image: positive or negative
- Illustrated experiment:
CIFAR

• A subset of classes of the Tiny Images dataset
• Given a keyword and image: positive or negative
• Illustrated experiment:
  • 63,000 images
CIFAR

- A subset of classes of the Tiny Images dataset
- Given a keyword and image: positive or negative

Illustrated experiment:
- 63,000 images
- 126 classes (at least 200 positive and 300 negative labels)
CIFAR

- A subset of classes of the Tiny Images dataset
- Given a keyword and image: positive or negative

Illustrated experiment:
- 63,000 images
- 126 classes (at least 200 positive and 300 negative labels)
- Random subset of $C$ classes
CIFAR

- A subset of classes of the Tiny Images dataset
- Given a keyword and image: positive or negative
- Illustrated experiment:
  - 63,000 images
  - 126 classes (at least 200 positive and 300 negative labels)
  - random subset of C classes
  - for each class c:
CIFAR

- A subset of classes of the Tiny Images dataset
- Given a keyword and image: positive or negative

Illustrated experiment:
- 63,000 images
- 126 classes (at least 200 positive and 300 negative labels)
- random subset of C classes
- for each class c:
  - test set: 100 positive and 200 negative examples
CIFAR

- A subset of classes of the Tiny Images dataset
- Given a keyword and image: positive or negative
- Illustrated experiment:
  - 63,000 images
  - 126 classes (at least 200 positive and 300 negative labels)
  - random subset of C classes
  - for each class c:
    - test set: 100 positive and 200 negative examples
    - training set: t positive/negative pairs
CIFAR

- Eigenfunction approach:
  - $k = 256$ eigenfunctions
  - 64D space
  - $\varepsilon = 0.2$
  - $\lambda = 50$
CIFAR

- Eigenfunction approach:
  - $k = 256$ eigenfunctions
  - 64D space
  - $\varepsilon = 0.2$
  - $\lambda = 50$

- Propagation for each class $c$ in turn:
CIFAR

- Eigenfunction approach:
  - $k = 256$ eigenfunctions
  - 64D space
  - $\varepsilon = 0.2$
  - $\lambda = 50$

- Propagation for each class $c$ in turn:
  - assign higher probability to the genuine positive images
CIFAR

- Eigenfunction approach:
  - $k = 256$ eigenfunctions
  - 64D space
  - $\varepsilon = 0.2$
  - $\lambda = 50$

- Propagation for each class $c$ in turn:
  - assign higher probability to the \textit{genuine positive images}
  - treat training examples other than $c$ as \textit{additional negative examples}
CIFAR

- Eigenfunction approach:
  - $k = 256$ eigenfunctions
  - $64D$ space
  - $\varepsilon = 0.2$
  - $\lambda = 50$

- Propagation for each class $c$ in turn:
  - assign higher probability to the **genuine positive images**
  - treat training examples other than $c$ as **additional negative examples**
  - **re-rank** the 300 test images
CIFAR
CIFAR

- Evaluation:
CIFAR

- Evaluation:
  - **precision** at a low recall rate of 15%
CIFAR

• Evaluation:
  • **precision** at a low recall rate of 15%
  • **chance level**: precision of 33%
CIFAR

• Evaluation:
  • precision at a low recall rate of 15%
  • chance level: precision of 33%
  • average over 10 different runs
CIFAR

• Evaluation:
  • precision at a low recall rate of 15%
  • chance level: precision of 33%
  • average over 10 different runs
  • different random train/test draws
CIFAR

- Evaluation:
  - precision at a low recall rate of 15%
  - chance level: precision of 33%
  - average over 10 different runs
    - different random train/test draws
    - different subsets of classes
CIFAR label set
CIFAR label set

![Graph showing mean precision at 15% recall averaged over 16 classes against the logarithm of the number of positive training examples per class. The graph compares different methods: Eigenfunction, Eigenfunction w/noisy labels, Nystrom, Least-squares, Eigenvector, SVM, NN, and Chance.]
CIFAR label set

- $C = 16$
- $0 \leq t \leq 100$
CIFAR label set

- $C = 16$
- $0 \leq t \leq 100$
- Baseline classifiers:
  - nearest-neighbor
  - RBF kernel SVM

![Graph showing mean precision at 15% recall averaged over 16 classes vs. log2 of number of positive training examples per class.](image)
CIFAR label set

- $C = 16$
- $0 \leq t \leq 100$
- Baseline classifiers:
  - nearest-neighbor
  - RBF kernel SVM
- Noisy labels:
  - keyword for query
  - small weight ($\lambda/10$) for each test example
CIFAR label set
CIFAR label set

(a) Without noisy labels
(b) With noisy labels
(c) Without noisy labels
Tiny Images dataset
Tiny Images dataset

- 79,302,017 images
- 32D space (PCA)
- $k = 64$ eigenfunctions
- 445,954 CIFAR labels
- 386 keywords
- 4 different keywords
- $t = 3$ labeled training pairs
Tiny Images dataset

- 79,302,017 images
- 32D space (PCA)
- $k = 64$ eigenfunctions
- 445,954 CIFAR labels
- 386 keywords
- 4 different keywords
- $t = 3$ labeled training pairs
Tiny Images dataset

- 79,302,017 images
- 32D space (PCA)
- \( k = 64 \) eigenfunctions
- 445,954 CIFAR labels
- 386 keywords
- 4 different keywords
- \( t = 3 \) labeled training pairs
Tiny Images dataset

- 79,302,017 images
- 32D space (PCA)
- $k = 64$ eigenfunctions
- 445,954 CIFAR labels
- 386 keywords
- 4 different keywords
- $t = 3$ labeled training pairs
• Eigenfunction method outperforms others
• Eigenfunction method outperforms others

• Increasing the number of classes improves performance
• Eigenfunction method outperforms others

• Increasing the number of classes improves performance

• Noisy labels aid performance
Conclusion
Conclusion

• Combine graph Laplacian with semi-supervised learning
Conclusion

• Combine graph Laplacian with semi-supervised learning

• Eigenfunctions incorporating density distribution
Conclusion

• Combine graph Laplacian with semi-supervised learning

• Eigenfunctions incorporating density distribution

• Demonstrated on challenging datasets and noisy labels