類似画像検索手法 Spectral Hashing の実現

Implementation of Spectral Hashing technique for retrieving similar images .

島根大学大学院　総合理工学研究科
数理・情報システム学専攻　田中研究室所属

S119314　小室　翼

1. 序論

近年インターネット上での画像の量は爆発的に増加しており、類似画像を検索する機械学習における訓練サンプルとして画像の巨大データセットを用意することが可能となり、より精度を上げることが出来るようになった。しかし、画像の巨大データセットに対して類似画像を検索することは非常に計算時間がかかることとなった。

そこでメモリ中に数百万枚の画像を格納することが出来、類似画像を素早く見つける様な特徴を持つ手法として、画像をバイナリコード化し類似画像は類似バイナリコードを持つ semantic hashing[1]や、機械学習手法を用いてより検索性能を良くした Boosting SSC[2]、RBMs[3]などが考案された。

今回再現をした手法 Spectral Hashing[4]は、上で挙げた手法より優れた性能であるとしており、実際に実装をしてみてその結果を評価・考察してみた。

2. Spectral Hashing とは

Spectral Hashing は教師なし学習手法の 1 つであり、複数枚の同じサイズの画像データをバイナリデータに非可逆変換し、変換したデータ間のハミング距離を計算することで類似画像を検索することが出来る。

2.1. Spectral Hashing の特徴・欠点

Spectral Hashing は画像データをバイナリコードにするにあたって、
(1)バイナリコード化するにあたってできる限り少ないビット数で表現ができる
(2)類似したバイナリコード間に対して類似した画像をマッピングする
という特徴を持つことを前提条件としている。また、画像という大きなバイトデータをバイナリデータに変換することから、変換したバイナリデータをメモリに大量に格納できるため、比較的高速で類似した画像を検索することができる。

2.2. Spectral Hashing の原理

特徴(1)について満たすコードというのは k ビットのバイナリデータに変換する際に、2^k 通りあるバイナリデータから等確率で1つのバイナリデータに変換されることである。つまり、0 と 1 がそれぞれ 50%の確率で選ばれて、かつ各ビットが独立していることが求められる。

特徴(2)については、画像はユークリッド距離が類似性と相互に関連するよう R^d に埋め込まれると仮定し、affinity matrix W_{n*n} を用いる。

この上記 2 点の特徴を満たすすべてのバイナリコードにおいて、類似ポイント間の平均ハミング距離が最小となるものを求める。

N 個のデータポイントに対して、

- $\{x_i\}$は i 番目のデータポイントの値(ベクトル)
- $\{y_i\}$は i 番目のデータポイントに対する変換後バイナリコードで長さ k のベクトル
- $W(i,j) = \exp(-||x_i-x_j||^2/\epsilon^2)$　(ϵ は正規分布の標準偏差のようなもの)

以上の表記を用いて、類似近傍間の平均ハミング距離は $\Sigma_{ij}W_{ij}||y_i-y_j||^2$ と表される。

ここで、ビットが独立であるという仮定を緩和して、非相関であるビットを求めるならば、以下のラプラシアン固有マップ法[5,6]を用いた問題となる。

問題 1: $\sum_{ij} W_{ij}\|y_i - y_j\|^2$

条件:

$y_i \in \{-1,1\}^k$

$\sum_i y_i = 0$

$\frac{1}{n}\sum y_i y_i^T = I$

条件 1: $y_i \in \{-1,1\}^k$ は長さ k のバイナリベクトル y_i の各要素を定義

条件 2: $\sum_i y_i = 0$ は 50%の確率で-1 と 1 が出るよう定義

条件 3: $\frac{1}{n}\sum y_i y_i^T = I$ はビットが非相関であること定義

ただし、この問題は平衡グラフ分割問題に相当し、NP 困難である。
先ほどの問題に対して、n×k 行列 Y(j 行が y_j^T)と n×n 対角行列 D
($D(i,i) = \sum_j W(i,j)$)を用いることで、以下のように書き換えることが出来る。

問題 2:最小となる $trace(Y^T(D-W)Y)$ を求める。

条件:

$Y(i,j) \in \{-1,1\}$

$Y^T 1 = 0$

$Y^T Y = I$

これはただ問題 1 とその条件を書き換えただけであり、NP 困難のままである。しかし、ここで条件 Y(i,j)∈{-1,1}を取り除くと、この問題の解は最小固有値を持つ D-W の k 次元固有ベクトルとなるような簡単な問題となる。
制限緩和した問題 2 の解が最小固有値をもつ D-W の k 次元固有ベクトルということは、バイナリコードを得るためにその固有値を 0 を閾値として 2 値化することを提案する。ただし、現状この方法は訓練サンプルにおける画像のコード表現をどのように計算するかということに留まっているため、訓練サンプル外においても成り立つよう out-of-sample extension をしなければならない。
この問題における out-of-sample extension は Nystrom method[7,8]を用いて解かれるスペクトル法における out-of-sample extension 問題である。しかし、新しいデータポイントに対する Nystrom extension 計算のコストは、データセットのサイズに対して線形であるため、何百万もの画像に対して計算を行うことは実用的ではない。そこで、効果的な out-of-sample extension を実現するために、データポイント $x_i \in R^d$ がとある確率分布 p(x)からのサンプルだと仮定する。

すると、問題 1 の方程式 $\sum_{ij} W_{ij}\|y_i - y_j\|^2$ は期待値で置き換えられた標本の期待値とみなせ、以下の問題と置き換えることが出来る。

問題 3:最小となる $\int \|y(x_1) - y(x_2)\|^2 W(x_1, x_2) p(x_1) p(x_2) dx_1 x_2$ を求める。

条件:

$y(x) \in \{-1,1\}^k$

$$\int y(x)\,p(x)\,dx = 0$$
$$\int y(x)\,y(x)^T\,p(x)\,dx = I$$
(ここで $W(x_1,x_2)$は $W(x_1,x_2)=\exp(-\|x_1-x_2\|^2/\epsilon^2)$ である)

この問題における、$y(x)\in\{-1,1\}^k$ 制限を緩和すると、解法が多様体上で定義される重み付き Laplace-Beltrami 作用素の固有関数であるスペクトル問題となる。
具体的には、
$$\frac{g(x)}{p(x)} = D(x)f(x)p(x) - \int_s W(s,x)f(s)p(s)ds \quad (\text{ここで } D(x)=\int_s W(x,s))$$
によって表される g に対して、$g = L_p f$ となるように関数 f をマッピングする演算子として重み付きラプラシアン L_p を定義し、最小固有値で $L_p f = \lambda f$ を満たす関数を求める問題である。
(ただし、固有値が 0 となる単純な解 f(x)=1 は除く)
ここで、参考論文[9,10,11]で議論されている正しい正規化において、$p(x)$から n 個のサンプルによって定義される離散ラプラシアンの固有ベクトルは $n\to\infty$としたとき、L_p の固有関数に収束する。
このとき、$p(x)$が分割可能な分布である多次元一様分布 $Pr(x)=\Pi_i u_i(x_i)$ (u_i は$[a_i,b_i]$を範囲とする独立分布)であると仮定する。
$p(x)$が分割可能かつデータポイント間の類似度が $\exp(-\|x_i-x_j\|^2/\epsilon^2)$ と定義されたならば、連続重み付きラプラシアン L_p の固有関数は外積形式を持つ[12]。
これは、もし Φ_i が固有値 λ_i の R1 上で定義される重み付きラプラシアンの固有関数ならば、$\Phi_i(x_1)\Phi_j(x_2)\cdots\Phi_d(x_d)$は d 次元問題の $\lambda_i\lambda_j\cdots\lambda_d$ を固有値とする固有関数であり、特に$[a,b]$区間の一様分布の場合において、一次元ラプラシアン L_p の固有関数は数学的によく研究されたものであり、金属板の基準振動モードと一致する。
すなわち、固有関数 $\Phi_k(x)$と固有値 λ_k は以下の式によって求められる。
$$\Phi_k(x) = \sin\left(\frac{\pi}{2} + \frac{i\pi}{(b-a)}x\right)$$
$$\lambda_k = 1 - e^{\frac{-\epsilon^2}{2}\left|\frac{i\pi}{b-a}\right|^2}$$
この固有値は分割と一致し、どの k ビットが使われるべきかを決定する。
そして、k 個の最小固有値 $\lambda_1,\lambda_2,\ldots,\lambda_k$ に対応する固有関数 $\Phi_1(x),\Phi_2(x),\ldots\Phi_k(x)$を計算し、0 を閾値で 2 値化することでバイナリコードを得ることにつながる。
ここで、固有値はアスペクト比と空間周波数 i に依存し、固有値はパラメータϵによって変わってしまうが、定数であるため固有値の並び方には影響を与えない。
また、固有関数の計算においてはパラメータϵを使わないことから、パラメータϵは最小固有値に対応する固有関数を求める際には無視しても構わない。
しかし固有関数のうち、単一次元の式 $\Phi_k(x_1)$か $\Phi_k(x_2)$であるような固有関数と、式 $\Phi_k(x_1)\Phi_l(x_2)$であるような外積の固有関数を識別しなければならない。
それは例えば、最小固有値 $y(x)=\mathrm{sign}(\Phi_k(x))$で L_p の k 個の固有関数を閾値で 2 値化すると仮定した時、もし固有関数のいづれかが外積の固有関数であるならば、そのビットはコードの他のビットの決定的関数となる。
なぜならば、$\mathrm{sign}(\Phi_1(x_1)\Phi_2(x_2)) = \mathrm{sign}(\Phi_1(x_1))\mathrm{sign}(\Phi_2(x_2))$であり、各ビットは独立であるという前提を緩和することになるからである。

以上の方法によって導き出される Spectral Hashing のコード化は特に、軸に沿った PCA と、その各軸が独立であると仮定した際の多次元矩形分布からデータへの単純なフィッティングが広範囲分布に対してよく働くとしている。

2.3. Spectral Hashing の計算の流れ

2.3.1. Spectral Hashing 全体の大まかな流れについて説明。

① Spectral Hashing を行うために、訓練サンプル(訓練データ)の数を設定する。

② Spectral Hashing で変換後のビット数と訓練サンプルをどの程度分割して計算するかを設定する。これは訓練サンプル数が多すぎる際メモリに格納しきれなくなるため。

③ ②で設定した変換後ビット数と分割数を引数として trainSH 関数を実行することで、Spectral Hashing に必要な様々なパラメータを含む構造体 SHparam を求め、それを返り値として取得する。

④ビット変換したいデータと SHparam を引数として compressSH 関数を実行し、データをビット変換した結果と、ビット変換をする際に用いた 0 を閾値として 2 値化される前の結果を返り値として取得する。

⑤類似画像を検索したい画像をビット変換したものと、類似画像が検索されるデータセットをビット変換したデータ全てを引数として hammingDist 関数を実行し、引数に用いた 2 つのデータのハミング距離を返り値として取得する。

⑥ ⑤で求めたハミング距離と表示画像枚数、どの長さのハミング距離まで検索を行うか、を引数として hamdisplayImage 関数を実行し、類似画像を表示させる。返り値は類似画像の index 番号。
図 2.3.1 はこの大きな流れのフローチャートである。

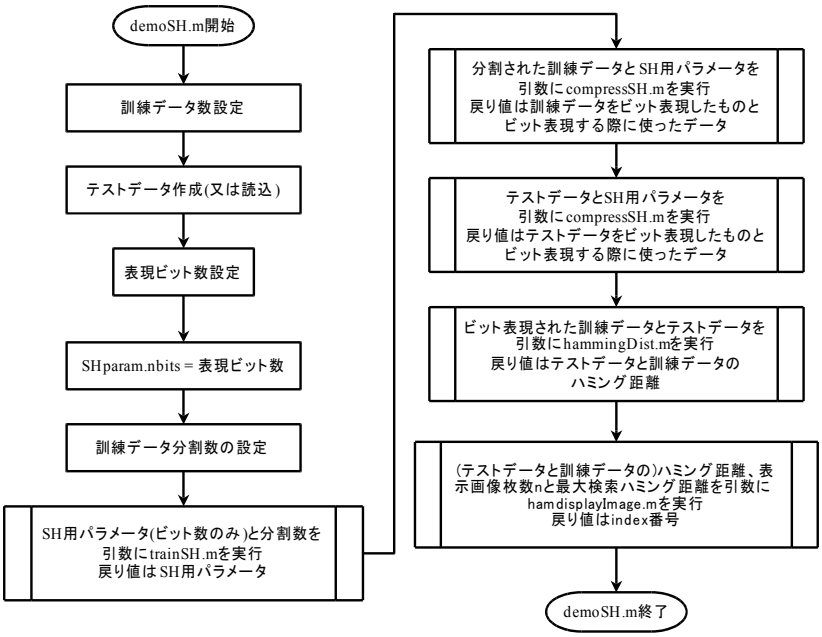


図 2.3.1:Spectral Hashing の大まかなフローチャート

2.3.2. trainSH 関数

訓練サンプルに対して主成分分析(SHparam.pc)を行い、主成分分析の各方向における最大(SHparam.mx)・最小(SHparam.mn)・範囲を計算し、空間周波数 i(モード)を求める。

次に、k 個の最小固有値に対応するモードの場所を求めるのために、固有値の計算式 $\lambda_k = 1 - e^{\frac{-\epsilon^2}{2}\left|\frac{i\pi}{b-a}\right|^2}$ から最小固有値を計算しなければならないが、実際は最小固有値の値ではなく、最小固有値のある位置が重要であるため、固有値の大小に関係する部分 $\left|\frac{i\pi}{b-a}\right|^2$ のみを計算し、最小固有値の位置に対応するモード(SHparam.modes)を求めることで Spectral Hashing 用パラメータ SHparam(構造体)の導出を行う。

図 2.3.2 はそのフローチャートである。

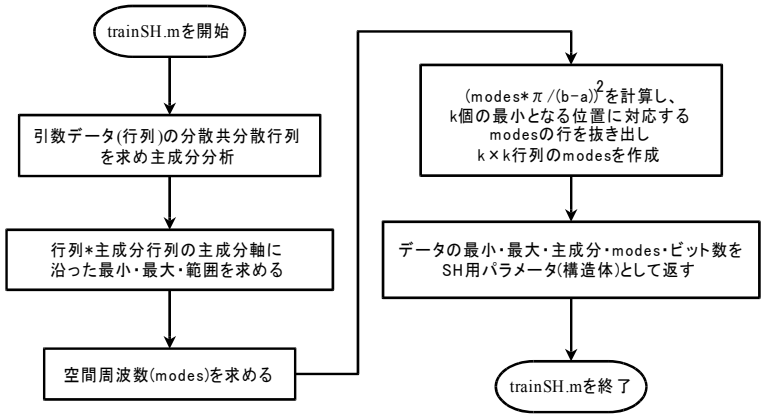


図 2.3.2:trainSH のフローチャート

2.3.3. compressSH 関数

ビット変換を行うデータ[X×N 行列(X はデータ数,N はバイト数)]を引数 1、trainSH 関数で求めた Spectral Hashing 用パラメータ SHparam(構造体)を引数 2 として実行される。

ビット変換を行うデータと trainSH 関数で求められた SHparam.pc をかけることで、ビット変換を行うデータの次元削減(PCA の軸に合わせる)をする。

次元削減されたデータの各行について SHparam.mn を引いたものを x とおくと、k bit バイナリデータに変換するための行列 B(X×k 行列)は

$$B(m,n)=sign(\prod_{j=1}^{k}(\sin(\frac{\pi}{2}+\frac{SHparam.modes(m,j)\times\pi}{(SHparam.mx_n-SHparam.mn_n)}x(m,j))))$$

と表され、B の行列要素の-1 を 0,1 を 1 と置いていくことで k ビットバイナリデータとなる。

返り値は、k ビットバイナリデータを 8bit 毎に 1 バイトに置き換えてまとめたものと、行列要素を置き換える前の B。

図 2.3.3 はそのフローチャートである。

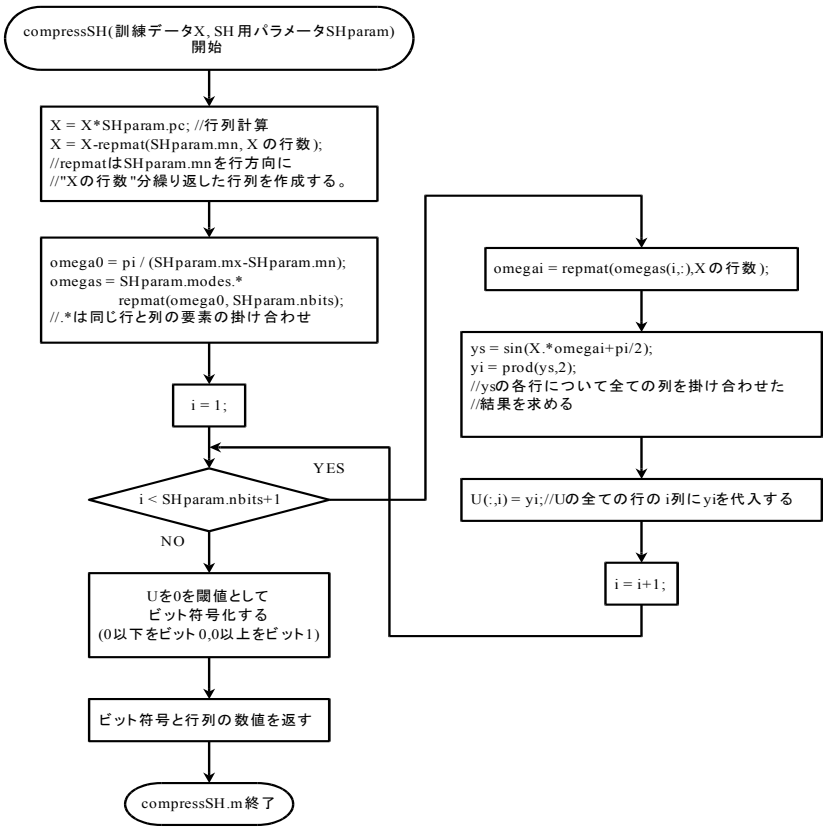


図 2.3.3:compressSH 関数のフローチャート

2.3.4. HammingDist 関数

類似画像検索元のバイナリコードを引数 1,類似画像を検索するデータセットのバイナリコードを引数 2 とおく。

引数 1 と引数 2 の XOR を 8bit 毎にとり、その結果の数字からハミング距離を計算し、足し合わせていくことで、ハミング距離を求める。

返り値は引数 1 と引数 2 のハミング距離。
図 2.3.4 はそのフローチャートである。

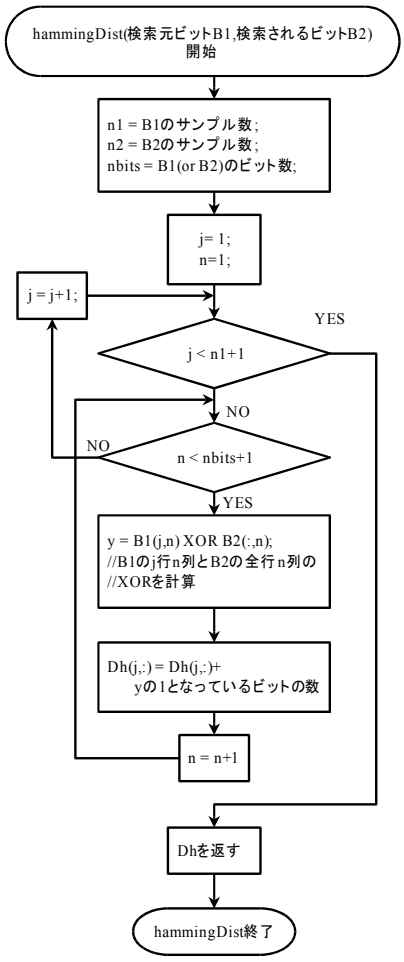


図 2.3.4:hammingDist 関数の
フローチャート

2.3.5. HamdisplayImage 関数

ハミング距離を引数 1、類似画像を検索する枚数を引数 2、どこまでのハミング距離を計算したら検索終了するか(最大検索ハミング距離)を引数 3 とおく。

ハミング距離 0 でバイナリデータを探して、対応する画像を表示して、類似画像検索枚数に達していないならば、ハミング距離 1 であるバイナリデータを探していき、類似画像検索枚数に達するか、現在検索中のハミング距離が最大検索ハミング距離以上になったら、類似画像検索を終了する。

返り値は index 番号。

図 2.3.5 はそのフローチャートである。

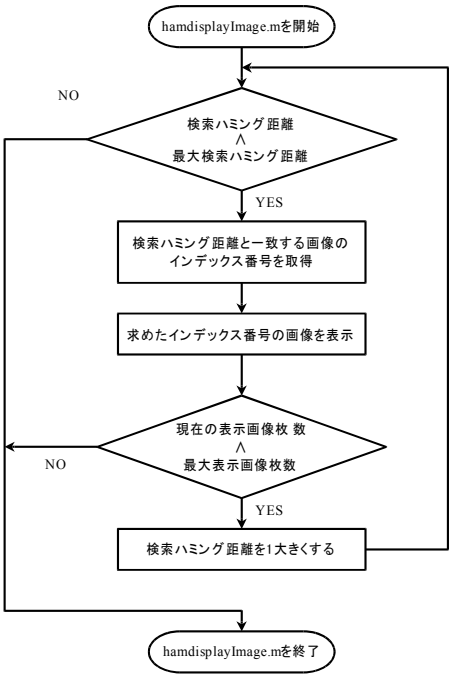


図 2.3.5:hamdisplayImage 関数の
フローチャート

3. 実験

3.1. 実験方法

79,302,017 枚の 32px×32px×RGB の画像データセット tiny_images.bin[13]に対して、16bit,32bit,64bit のバイナリデータに変換し、変換が完了するまでの各処理の時間と類似画像検索時間の計測、及び類似画像検索結果を表示する。

変換が完了するまでの各処理は、

① 主成分分析

② モード計算

③ バイナリデータ変換

の 3 つであり、それぞれ 3 回実行しその平均時間を計測する。ただし、主成分分析における分散共分散の計算時間は bit 数にかかわらず同じであるため、主成分分析における分散共分散の計算と固有ベクトルの導出は分けて計測する。

また類似画像検索は、

① 類似画像検索元をバイナリデータに変換したものと、類似画像を検索したいデータセットをバイナリデータに変換したもののハミング距離を計算する。

② ハミング距離が小さいバイナリデータを 30 個見つけ、表示する。

という、2 つの処理があり、どちらも類似画像検索毎に 3 回実行し、その都度平均時間を計測する。

なお検索するバイナリデータの個数を 30 個としたのは類似画像を検索するという目的のため、どの程度類似した画像が出現するかを視覚的にわかりやすくするためである。

類似画像検索を行う画像は、tiny_images データセット中から画像を 3 枚選び、また単純な画素移動をした画像に対して類似画像検索を行い、画素移動が行われていても、画素移動前の画像の類似画像検索結果と同様な結果が出るかどうかも調査する。
今回の実験では単純な画素移動のパターンとして、元画像に対して 1 ピクセルずらす,90 度右回転,180 度回転の 3 パターンを行った。
16bit,32bit,64bit 変換をしたバイナリデータに対して、元画像 3 枚×4 パターン(画素移動なし+画素移動 3 パターン)の計 12 種類の画像の類似画像検索を行い、検索結果及び計算時間を求めた。
類似画像検索方法は、バイナリデータのハミング距離が近いものが類似画像であるという特徴であるため、ハミング距離が 0 であるものから検索していき、最終的に 30 枚ヒットするまでハミング距離を 1 づつ増やして検索する方法をとった。
表 1 は実験において使用した計算機のスペックである。

表 1:実行環境

OS	CentOS5.4(64bit)
CPU	Celeron G1101 (2 コア 2.26GHz)
メモリ	1GB*2(DDR3-1333)
実行言語	Matlab R14SP3

3.2. 実験結果

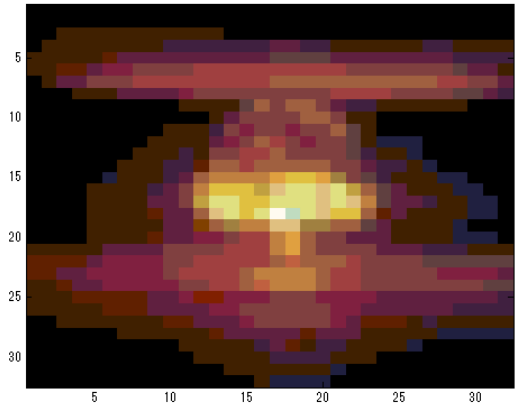
表 2 は 16bit,32bit,64bit へのバイナリデータ変換が完了するまでの各処理時間を計測した結果である。

表 2:バイナリデータ変換が完了するまでの各処理の処理時間

	16bit	32bit	64bit
分散共分散計算[sec]	315234.06832[sec] (約 87.5 時間)		
固有ベクトル導出[sec]	1.200169[sec]	1.331223[sec]	2.346969[sec]
モード計算[sec]	11702.617835[sec]	13010.214674[sec]	15774.360375[sec]
バイナリデータ変換[sec]	13632.217984[sec]	16737.920986[sec]	27210.748993[sec]

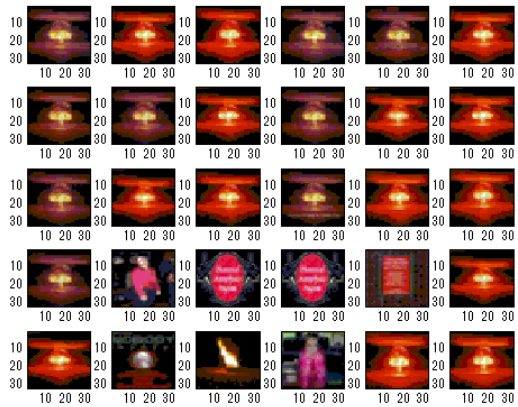
次に、類似画像検索をしたい適当な 3 枚の画像(tiny_images.bin から抽出)と、その各画像の画素移動 3 パターン、計 12 種類の画像について

16bit,32bit,64bit 変換の類似画像 30 枚、及びハミング距離計算時間と類似画像検索時間の測定をした。
以下の図 3.1.1~表 3.4.4 がその結果である。

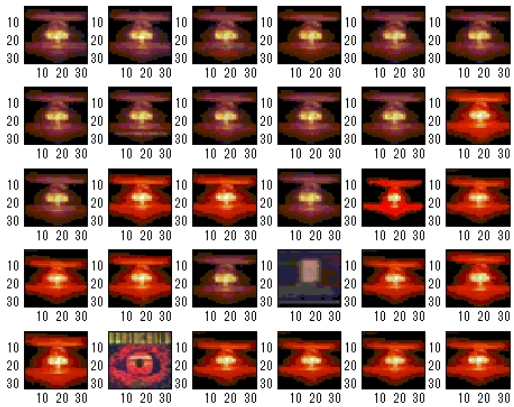


元画像
(画像 1)

16bit
ハミング距離計算時間:4.940990[sec]
類似画像検索時間:1.5886410[sec]



32bit
ハミング距離計算時間:9.6274770[sec]
類似画像検索時間:2.6286940[sec]



64bit
ハミング距離計算時間:19.162180[sec]
類似画像検索時間:5.7745230[sec]

図 3.1.1:画像 1 に対する 16bit,32bit,64bit 変換の類似画像と

ハミング距離計算時間・類似画像検索時間

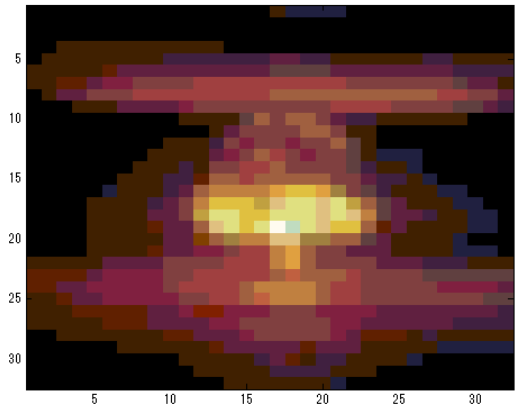
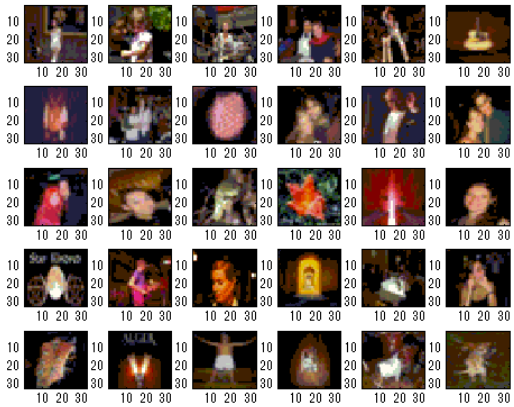
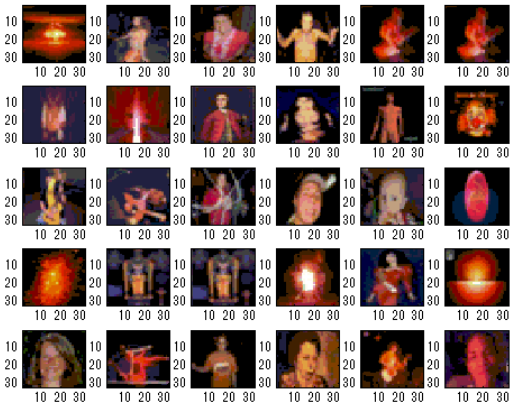
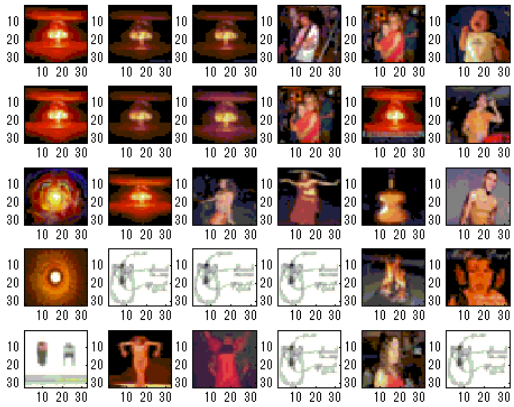
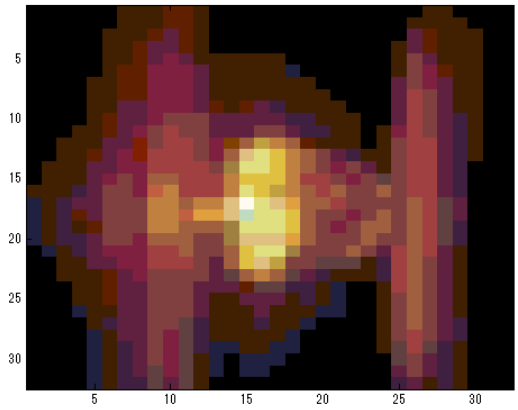
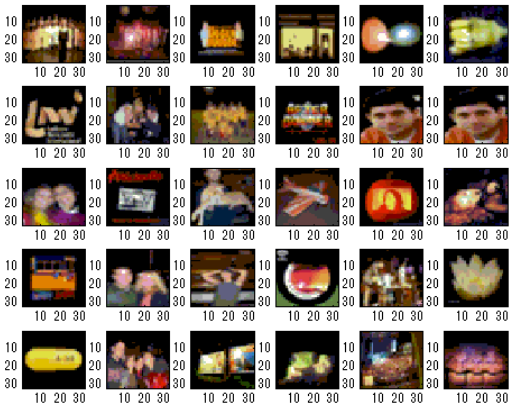
<table>
<tr>
<td> 
元画像
(画像 1 を 1 画素移動したもの)</td>
<td> 
16bit
ハミング距離計算時間:5.0739550[sec]
類似画像検索時間:1.58250[sec]</td>
</tr>
<tr>
<td> 
32bit
ハミング距離計算時間:9.528850[sec]
類似画像検索時間:2.5705670[sec]</td>
<td> 
64bit
ハミング距離計算時間:18.9325870[sec]
類似画像検索時間:7.5772220[sec]</td>
</tr>
</table>

図 3.1.2:画像 1 を 1 画素移動したものに対する 16bit,32bit,64bit 変換の類似画像と

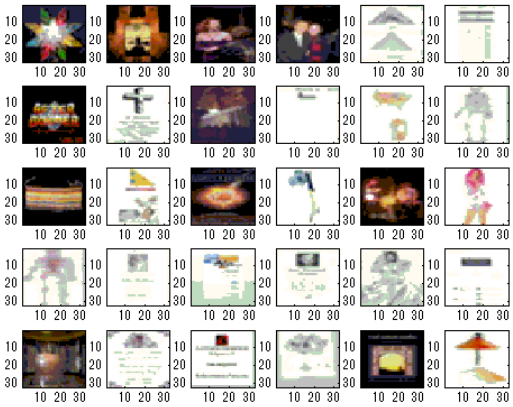
ハミング距離計算時間・類似画像検索時間



元画像
(画像 1 を右に 90 度回転したもの)



16bit
ハミング距離計算時間:5.0774230[sec]
類似画像検索時間:1.5875120[sec]



32bit
ハミング距離計算時間:9.5364180[sec]
類似画像検索時間:2.617890[sec]

64bit
ハミング距離計算時間:18.9150[sec]
類似画像検索時間:6.5244670[sec]

図 3.1.3:画像 1 を右に 90 度回転したものに対する 16bit,32bit,64bit 変換の類似画像と

ハミング距離計算時間・類似画像検索時間

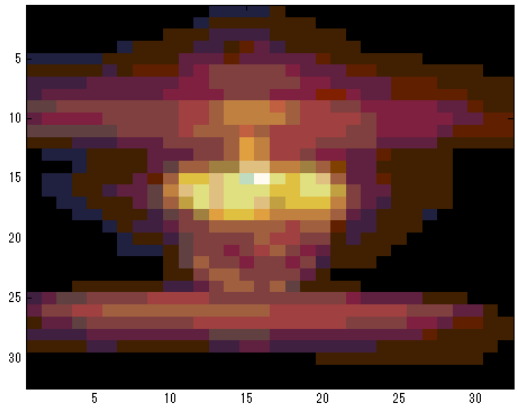
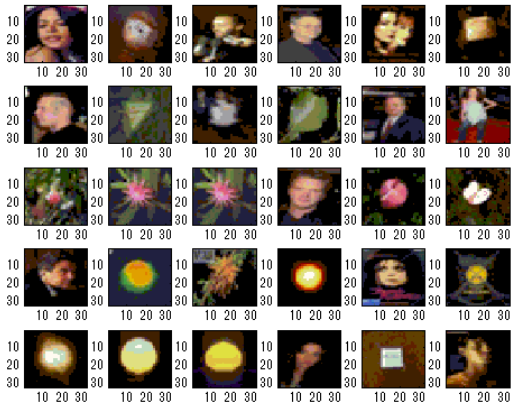
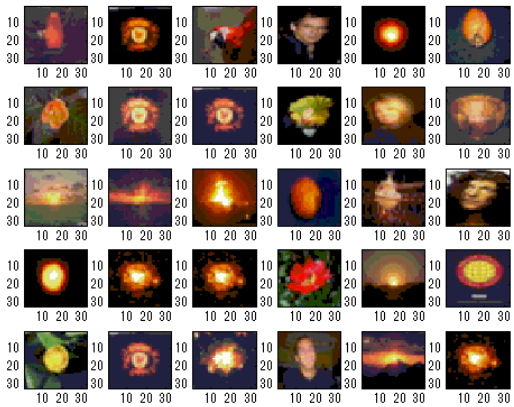
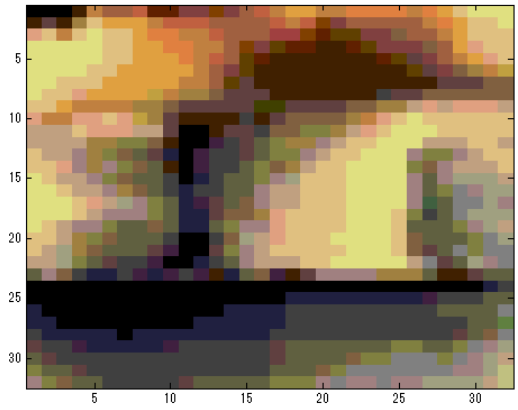
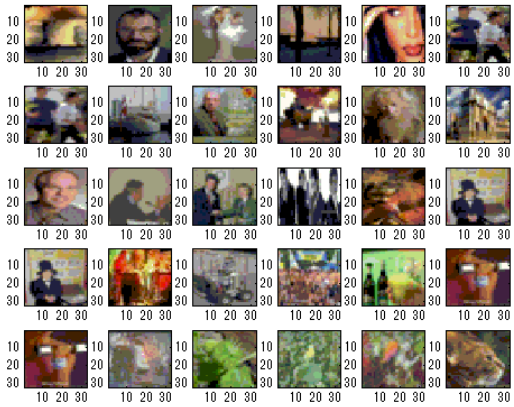
<table>
<tr>
<td> 
元画像
(画像 1 を 180 度回転したもの)</td>
<td> 
16bit
ハミング距離計算時間:5.0736180[sec]
類似画像検索時間:1.5409510[sec]</td>
</tr>
<tr>
<td> 
32bit
ハミング距離計算時間:9.5328490[sec]
類似画像検索時間:2.532910[sec]</td>
<td> 
64bit
ハミング距離計算時間:18.9176710[sec]
類似画像検索時間:6.3784830[sec]</td>
</tr>
</table>

図 3.1.4:画像 1 を 180 度回転したものに対する 16bit,32bit,64bit 変換の類似画像と

ハミング距離計算時間・類似画像検索時間

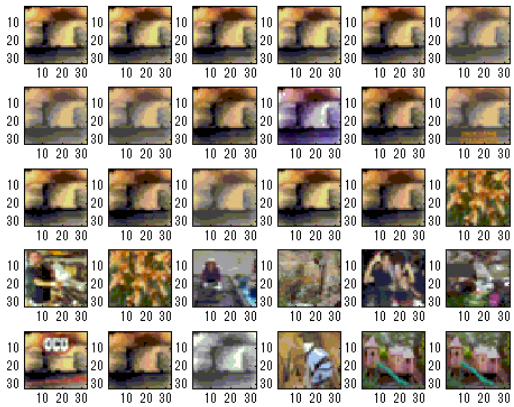


元画像
(画像 2)



16bit
ハミング距離計算時間:5.0717690[sec]
類似画像検索時間:1.5940540[sec]

32bit
ハミング距離計算時間:9.9321640[sec]
類似画像検索時間:1.5444440[sec]



64bit
ハミング距離計算時間:18.9135460[sec]
類似画像検索時間:5.7445080[sec]

図 3.2.1:画像 2 に対する 16bit,32bit,64bit 変換の類似画像と

ハミング距離計算時間・類似画像検索時間

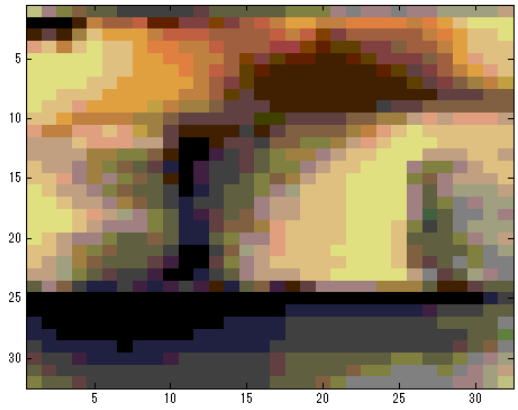
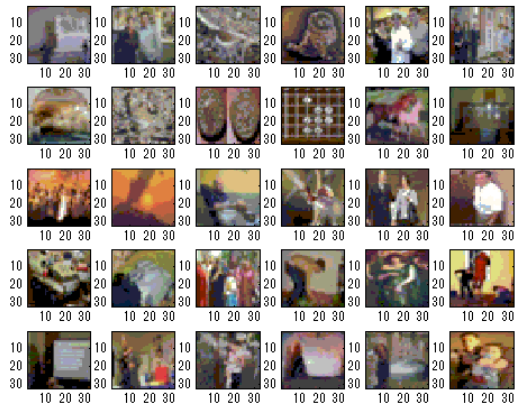
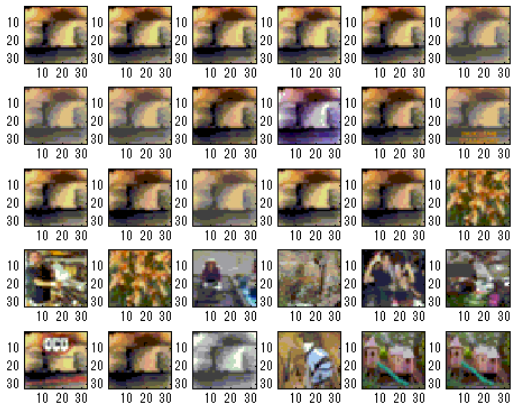
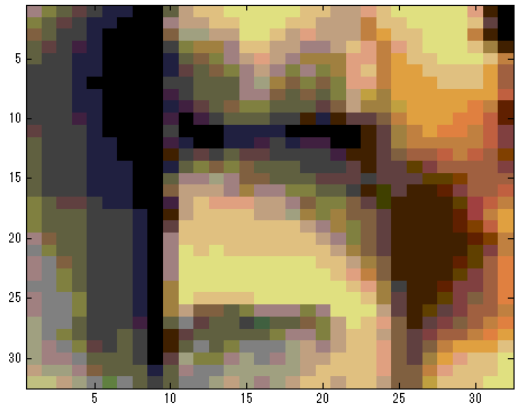
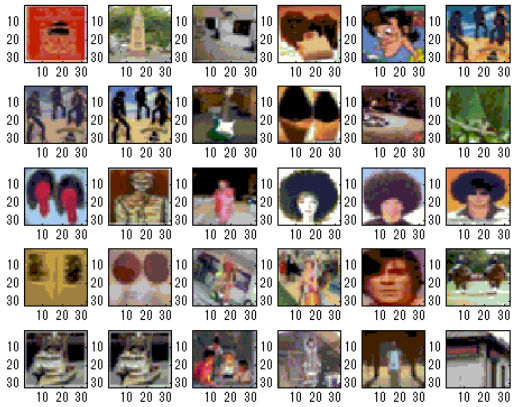
<table>
<tr>
<td> 
元画像
(画像 2 を 1 画素移動したもの)</td>
<td> 
16bit
ハミング距離計算時間:4.8548960[sec]
類似画像検索時間:1.5223630[sec]</td>
</tr>
<tr>
<td> 
32bit
ハミング距離計算時間:9.9711640[sec]
類似画像検索時間:3.7958830[sec]</td>
<td> 
64bit
ハミング距離計算時間:18.9059020[sec]
類似画像検索時間:6.6087850[sec]</td>
</tr>
</table>

図 3.2.2:画像 2 を 1 画素移動したものに対する 16bit,32bit,64bit 変換の類似画像と

ハミング距離計算時間・類似画像検索時間

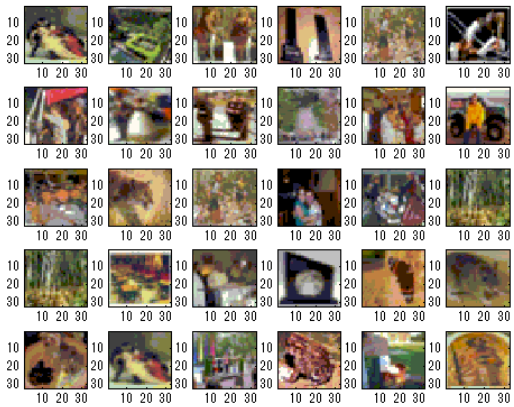


元画像
(画像 2 を右に 90 度回転したもの)



16bit
ハミング距離計算時間:4.8772300[sec]
類似画像検索時間:1.5878120[sec]

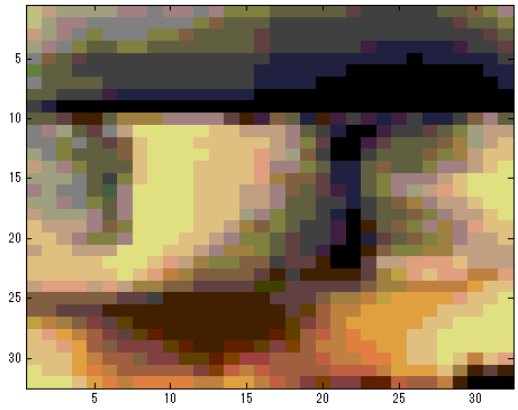
32bit
ハミング距離計算時間:9.9713550[sec]
類似画像検索時間:2.5923790[sec]



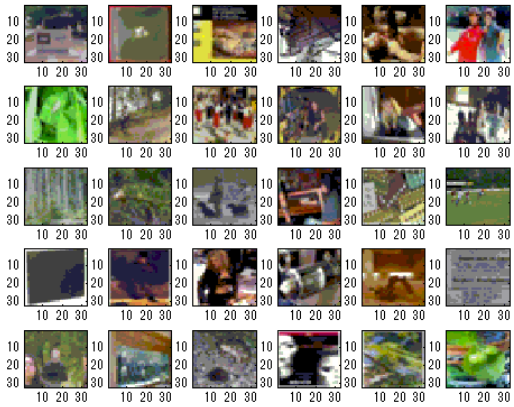
64bit
ハミング距離計算時間:18.8970620[sec]
類似画像検索時間:7.5142740[sec]

図 3.2.3:画像 2 を右に 90 度回転したものに対する 16bit,32bit,64bit 変換の類似画像と

ハミング距離計算時間・類似画像検索時間



元画像
(画像 2 を 180 度回転したもの)



16bit
ハミング距離計算時間:4.8526240[sec]
類似画像検索時間:1.5173490[sec]

32bit
ハミング距離計算時間:9.9231630[sec]
類似画像検索時間:2.6407770[sec]

64bit
ハミング距離計算時間:18.9016360[sec]
類似画像検索時間:8.5028520[sec]

図 3.2.4:画像 2 を 180 度回転したものに対する 16bit,32bit,64bit 変換の類似画像と

ハミング距離計算時間・類似画像検索時間

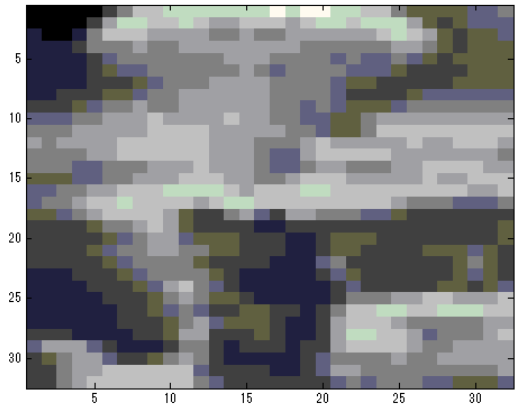
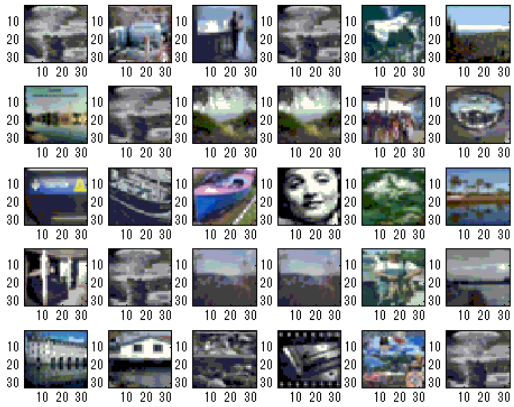
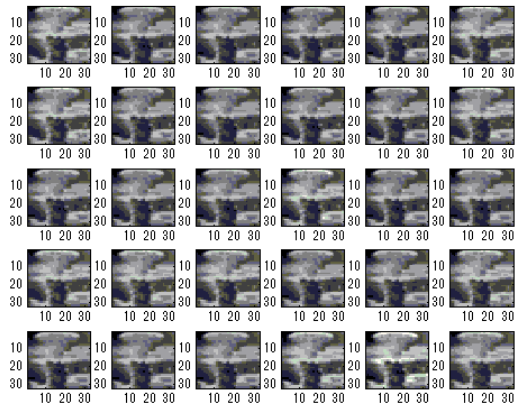
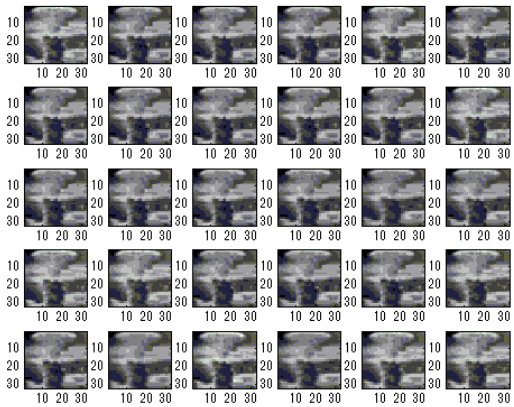
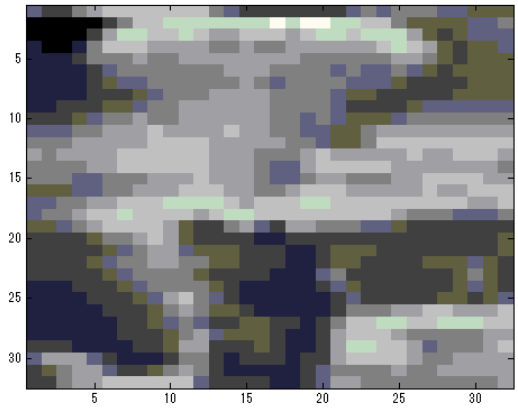
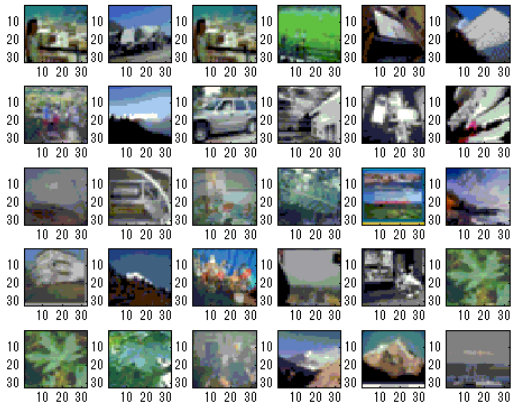
<table>
<tr>
<td> 
元画像
(画像 3)</td>
<td> 
16bit
ハミング距離計算時間:5.0034810[sec]
類似画像検索時間:1.5193450[sec]</td>
</tr>
<tr>
<td> 
32bit
ハミング距離計算時間:9.9321640[sec]
類似画像検索時間:1.5444440[sec]</td>
<td> 
64bit
ハミング距離計算時間:19.8573070[sec]
類似画像検索時間:2.7387870[sec]</td>
</tr>
</table>

図 3.3.1:画像 3 に対する 16bit,32bit,64bit 変換の類似画像と

ハミング距離計算時間・類似画像検索時間



元画像
(画像 3 を 1 画素移動したもの)



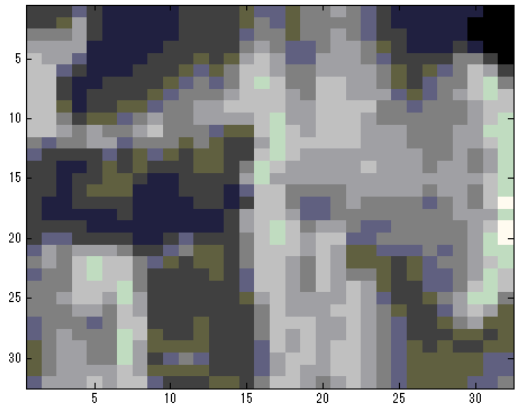
16bit
ハミング距離計算時間:4.8666680[sec]
類似画像検索時間:1.5227890[sec]

32bit
ハミング距離計算時間:9.9711640[sec]
類似画像検索時間:3.7958830[sec]

64bit
ハミング距離計算時間:19.7378990[sec]
類似画像検索時間:7.8979910[sec]

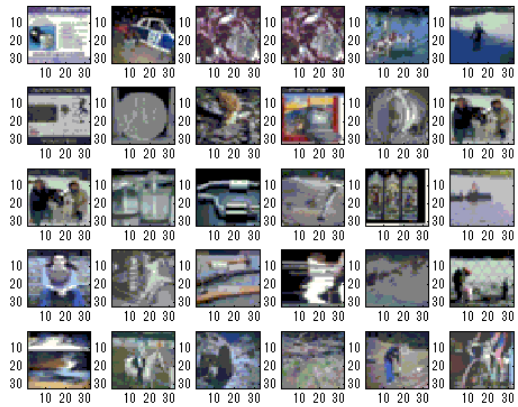
図 3.3.2:画像 3 を 1 画素移動したものに対する 16bit,32bit,64bit 変換の類似画像と

ハミング距離計算時間・類似画像検索時間

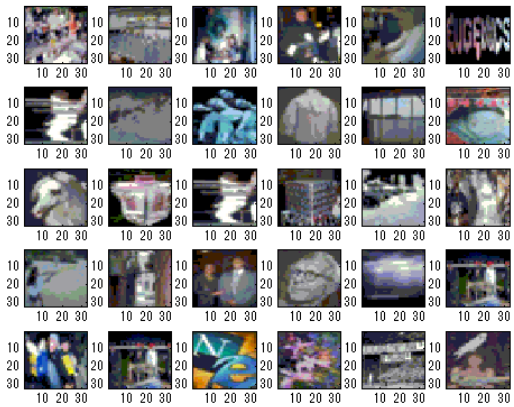


元画像
(画像 3 を右に 90 度回転したもの)

16bit
ハミング距離計算時間:4.8436060[sec]
類似画像検索時間:1.5129140[sec]



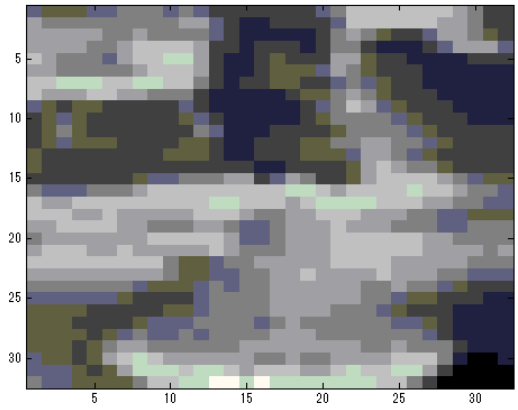
32bit
ハミング距離計算時間:9.9713550[sec]
類似画像検索時間:2.5923790[sec]



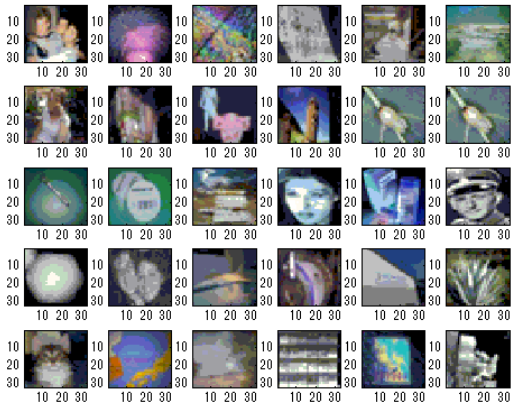
64bit
ハミング距離計算時間:19.7303690[sec]
類似画像検索時間:8.9288860[sec]

図 3.3.3:画像 3 を右に 90 度回転したものに対する 16bit,32bit,64bit 変換の類似画像と

ハミング距離計算時間・類似画像検索時間

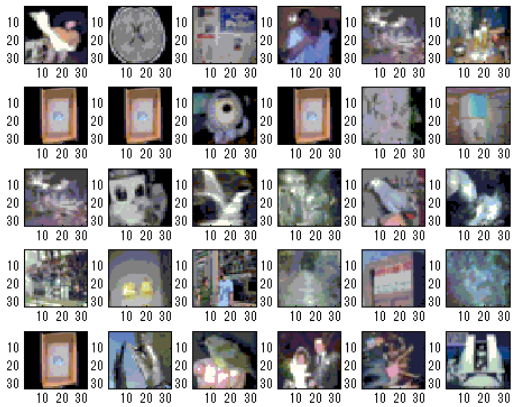


元画像
(画像 3 を 180 度回転したもの)



16bit
ハミング距離計算時間:4.8389230[sec]
類似画像検索時間:1.5166440[sec]

32bit
ハミング距離計算時間:9.9231630[sec]
類似画像検索時間:2.6407770[sec]



64bit
ハミング距離計算時間:19.7439680[sec]
類似画像検索時間:7.731820[sec]

図 3.3.4:画像 3 を 180 度回転したものに対する 16bit,32bit,64bit 変換の類似画像と

ハミング距離計算時間・類似画像検索時間

3.3. 評価・考察

まずは類似画像検索精度について評価・考察をする。
類似画像の検索は変換ビットの数を 16bit,32bit,64bit と上げていくにつれて、検索精度が上昇しているように感じられた。
特に、画素移動をした画像に対する類似画像検索結果に注目をすると、16bit,32bit,64bit とビット表現数が上がるにつれて検索精度が上昇していることが目に見えてわかるようになった。

一方、画素移動をしていない画像に対する類似画像検索結果に注目すると、16bit から 32bit は精度が上昇していることが目に見えてわかるが、32bit から 64bit は 16bit から 32bit にした時ほどの精度上昇はしていないように見える。このことは Spectral Hashing の論文における、32bit から 64bit にしても類似画像検索精度は 16bit から 32bit にした時より大きく上がらなかったという結果と合致した。
16bit から 32bit にした際に性能が大きく上がる原因としては、16bit で表現できるグループの数が 2^{16}(=65536)しかないからであり、ハミング距離 1 につき平均 1,210 枚の画像がヒットすることとなりノイズの数が 30 に対して非常に大きくなるためであると考えられる。
一方 32bit から 64bit にした際に性能がそこまで上がらない原因としては、Spectral Hashing の原理として固有値は分割と一致し、最小固有値はより効果的な分割であるということから、最初の方の最小固有値で大体効果的な分割をしてしまっているからではないかと考えられる。

次に、計算時間について評価・考察する。
Spectral Hashing の実行の際にネックとなるのは、実行時間から見るに分散共分散の計算、モード計算、バイナリデータ変換の 3 つが挙げられる。
しかし、この 3 つは一度計算をしてしまえば後は使い回していく部分、つまり実際に類似画像を検索するための前準備に当たる部分であり、既に計算されているならば、類似画像検索する際には引用するだけで済む所である。けれども、新規に計算する場合は非常に時間がかかるということは変わらないため、今から準備してすぐにでも類似画像を検索したいという場合には有用ではない。
類似画像の検索においては、やはり bit 数が少ないほど速くハミング距離計算や類似画像検索が出来るようになっていた。そこで、ハミング距離計算時間と類似画像検索時間を、画像の種類によらず 16bit,32bit,64bit を基準に平均をとってみると以下の表 4 の様な結果が出た。

表 4:各 bit 数ごとのハミング距離計算時間と類似画像計算時間の平均

	16bit	32bit	64bit
ハミング距離計算時間	4.9479319167[sec]	9.8184405[sec]	19.21792725[sec]
類似画像検索時間	1.5494061667[sec]	2.62475225[sec]	6.8268831667[sec]
合計時間	6.4973380834[sec]	12.44319275[sec]	26.0448104167[sec]

類似画像を検索したい画像を与えられて、画像データセット tiny_images.bin から類似画像を検索する時間としては、64bit 変換でも 26 秒と高速な結果が出たのではないかと思われる。
計算量についてはハミング距離計算時間はそのアルゴリズムからしてビット数に対して線形に計算時間が増えていくことは自明であり、そのとおりの結果が出た。

一方類似画像検索時間は、ハミング距離が d 以下である画像を全部表示するというアルゴリズムを使えば線形に計算時間が増えていくようになると考えられるが、今回用いたアルゴリズムはハミング距離が d 以下かつ 30 枚表示したら終了としている関係上、完璧に線形な計算時間にはならなかった。

4. 今後の課題

今回の実験においては他手法との精度比較や計算速度比較、類似画像検索精度の定量的な評価ができておらず、Spectral Hashing の評価という点では片手落ちな感が否めない。そのため、他手法との比較や類似画像検索精度の定量的評価はデータの信頼性を高めるための課題としてあげられる。

また、32bit 変換から 64bit 変換にした際の類似画像検索精度の向上に対する考察が実際に考察通りであるかどうか 64bit 変換より上の bit 変換をし、精度向上が行われるか確かめることも必要である。

次に、Spectral Hashing の計算において非常に時間がかかった主成分分析やビット変換の計算を分散処理をすることで、計算時間の短縮を図ることも課題として挙げられる。

5. 謝辞

本研究を行うにあたり、御迷惑をおかけしながらも最後まで熱心な御指導をしていただきました田中章司郎教授には心より御礼を申し上げます。また、同じ研究室に所属する皆様には本研究に関して、様々な御協力と御助言を頂きました。この場で厚く御礼申し上げます。

なお、本論文、本研究で作成したデータ、並びに関連する発表資料などの全ての知的財産権を本研究の指導教官である田中章司郎教授に御譲渡いたします。

6. 参考論文

[1] R. R. Salakhutdinov and G. E. Hinton. Semantic hashing. In SIGIR workshop on Information Retrieval and applications of Graphical Models, 2007.

[2] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter sensitive hashing. In ICCV, 2003.

[3] R. R. Salakhutdinov and G. E. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure.

[4] Y.Weiss, A.Torralba and R.Fergus “Spectral Hashing” Advances in Neural Information Processing System, 2008.

[5] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In NIPS, pages 585–591, 2001.

[6] A.Y. Ng, M.I. Jordan, and Y.Weiss. On spectral clustering, analysis and an algorithm. In Advances in Neural Information Processing 14, 2001.

[7] Yoshua Bengio, Olivier Delalleau, Nicolas Le Roux, Jean-Fran¸cois Paiement, Pascal Vincent, and Marie Ouimet. Learning eigenfunctions links spectral embedding and kernel pca. Neural Computation, 16(10):2197–2219, 2004.

[8] Charless Fowlkes, Serge Belongie, Fan R. K. Chung, and Jitendra Malik. Spectral grouping using the nystr¨om method. IEEE Trans. Pattern Anal. Mach. Intell., 26(2):214–225, 2004.

[9] Yoshua Bengio, Olivier Delalleau, Nicolas Le Roux, Jean-Fran¸cois Paiement, Pascal Vincent, and Marie Ouimet. Learning eigenfunctions links spectral embedding and kernel pca. Neural Computation, 16(10):2197–2219, 2004.

[10] R. R. Coifman, S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker.

Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. Proceedings of the National Academy of Sciences, 102(21):7426–7431, May 2005.
[11] M. Belkin and P. Niyogi. Towards a theoretical foundation for laplacian based manifold methods. Journal of Computer and System Sciences, 2007.
[12] Boaz Nadler, Stephane Lafon amd Ronald R. Coifman, and Ioannis G. Kevrekidis. Diffusion maps, spectral clustering and reaction coordinates of dynamical systems. Arxiv, 2008. http://arxiv.org/.
[13] tiny_images.bin http://horatio.cs.nyu.edu/mit/tiny/data/tiny_images.bin