hw7

May 7, 2015

1 Homework 7

- Name: Austin Chen
- SID: 23826762
- Repro: Open up hw7.ipynb in IPython Notebook.

1.1 1. K-means Clustering

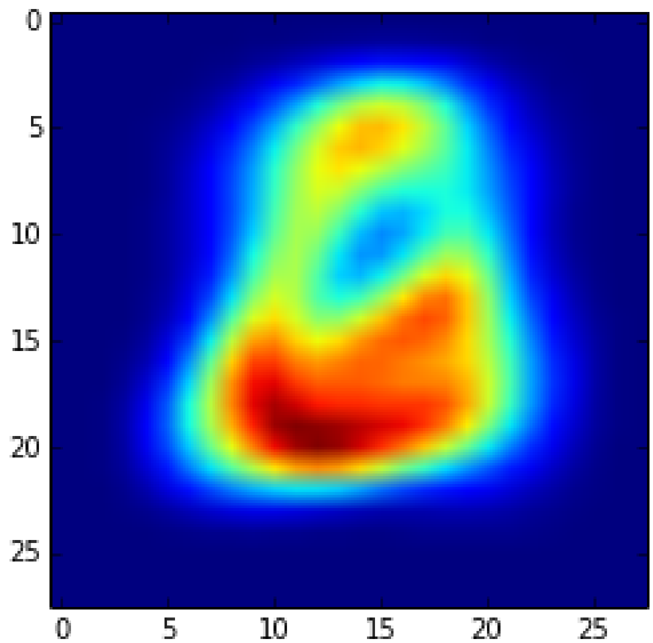
The final k-means loss does indeed vary across differ runs, depending on the random initialization

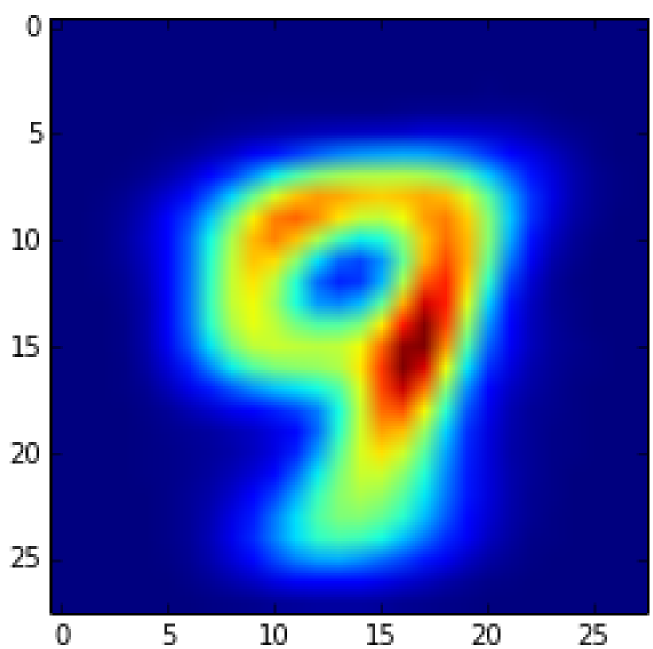
In [2]:
```
import numpy as np
import scipy.io
import math
import random
from sklearn import preprocessing
from __future__ import division
import matplotlib.pyplot as plt
%matplotlib inline

# Load the data
mat = scipy.io.loadmat('mnist_data/images.mat')
images = mat['images']
images = [images[:, :, i].astype(float) for i in range(60000)]
```

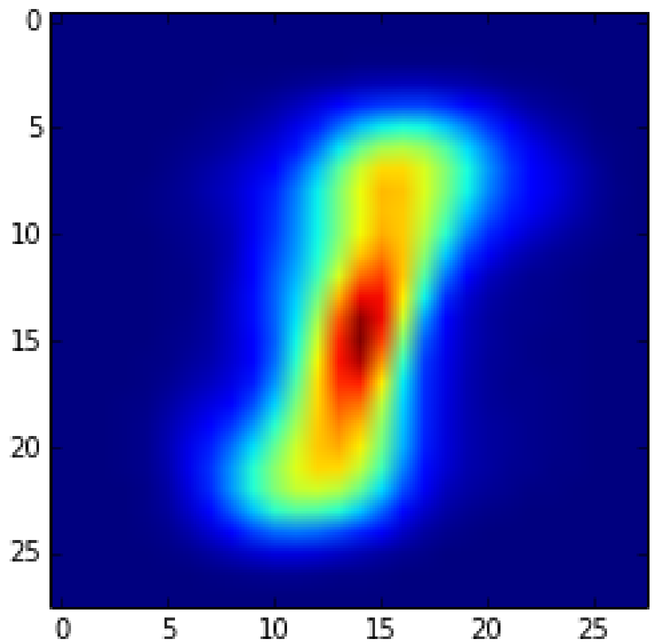
In [2]:
```
for k in [5, 10, 20]:
    # Randomly initialize centers
    # centers = [np.random.rand(28, 28) * 256 for _ in range(k)]
    centers = np.random.permutation(images)[0:k]

    prevCounts, counts = np.ones(k) * 100, np.zeros(k)
    while(True):
        # Cluster images to nearest centers
        clusters = [[] for _ in range(k)]
        for image in images:
            i = min(range(k), key=lambda i: np.linalg.norm(image - centers[i]))
            clusters[i].append(image)

        # Recompute means
        for i in range(k):
            if len(clusters[i]) > 0:
                centers[i] = sum(clusters[i]) / len(clusters[i])

        # Calculate loss
```

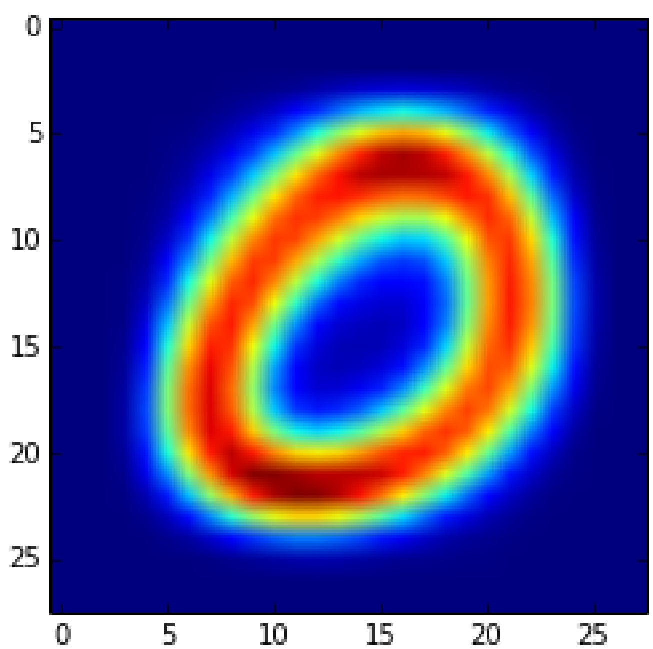
```
        J = sum(np.linalg.norm(image - centers[i])**2 for i in range(k) for image in clusters[i
        print(J)

        # Check for convergence
        counts = np.array([len(cluster) for cluster in clusters])
        print(counts)
        if np.linalg.norm(prevCounts - counts) < 100:
            break
        else:
            prevCounts = counts

    # Show the centers
    for i in range(k):
        plt.imshow(centers[i])
        plt.show()
```

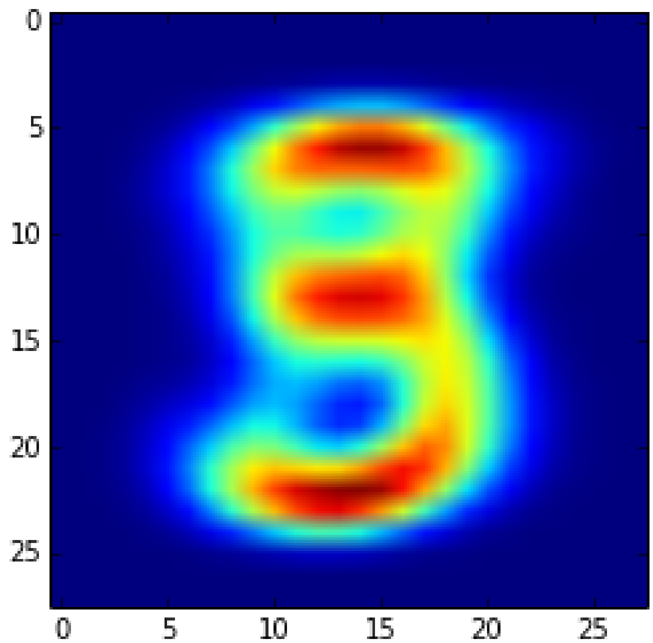
```
188401674407.0
[ 8309 14895 15794 14736  6266]
181304793180.0
[ 8880 14804 15125 12989  8202]
179629937635.0
[ 9494 15253 15013 11204  9036]
178830744981.0
[ 9970 15642 14982 10057  9349]
178335832912.0
[10198 15971 15046  9190  9595]
177971614122.0
[10233 16275 15190  8467  9835]
177645850188.0
[10277 16519 15356  7734 10114]
177340845398.0
[10364 16674 15515  7040 10407]
177096837756.0
[10501 16745 15602  6454 10698]
176933141432.0
[10620 16745 15609  6005 11021]
176866572143.0
[10724 16733 15533  5760 11250]
176841972700.0
[10805 16696 15468  5642 11389]
176829877856.0
[10855 16662 15403  5587 11493]
176824875852.0
[10872 16647 15348  5571 11562]
```


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25

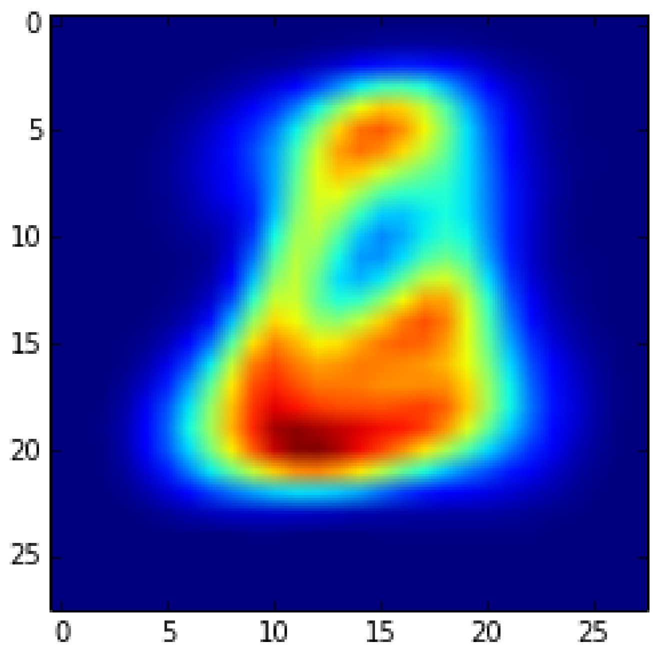

0
5
10
15
20
25
0
5
10
15
20
25

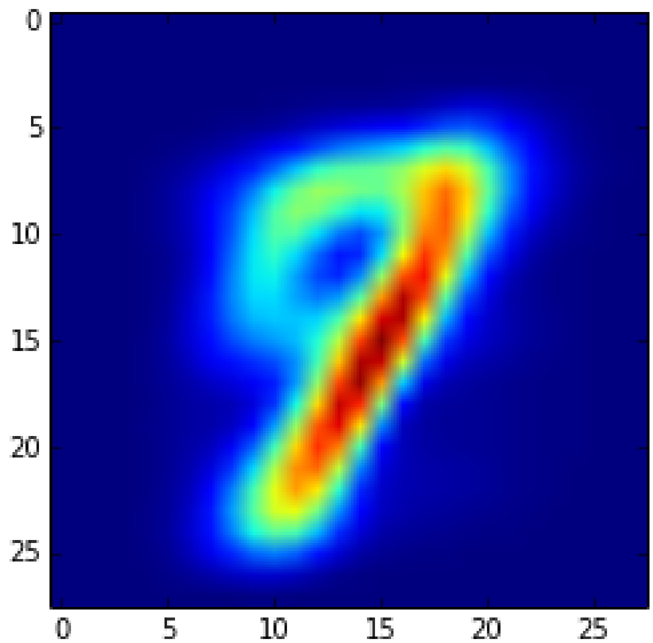

0
5
10
15
20
25
0
5
10
15
20
25

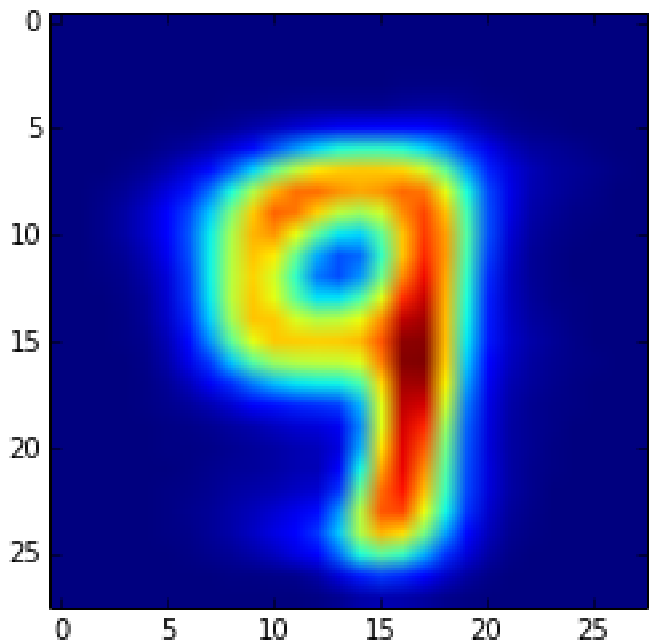


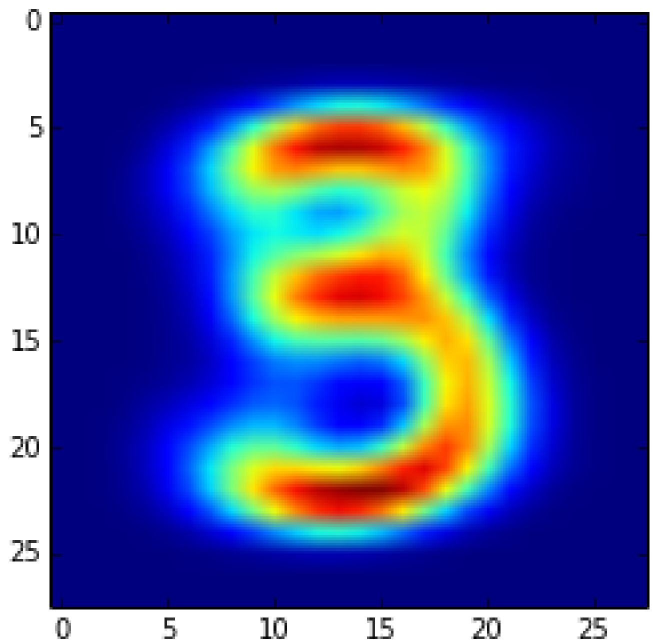

```
178353166722.0
[ 5503  9169  5023  7681 15206  5312  2208  2400  4684  2814]
171086950722.0
[ 5462  7310  6405  7065 14124  5199  3338  3080  4613  3404]
168925576815.0
[ 6086  7248  6846  6945 12821  4966  3584  2958  5250  3296]
167459088863.0
[ 6631  7320  6861  7026 11738  4955  3686  2849  5792  3142]
166516912095.0
[ 7082  7176  6634  7272 11045  5081  3756  2834  6105  3015]
165908922013.0
[ 7479  6836  6383  7480 10704  5373  3856  2851  6084  2954]
165462549964.0
[ 7670  6463  6235  7595 10601  5640  3967  2888  6003  2938]
165151106537.0
[ 7734  6147  6174  7685 10517  5838  4059  2919  5995  2932]
164932998298.0
[ 7769  5933  6241  7737 10446  5949  4127  2974  5922  2902]
164757307562.0
[ 7830  5808  6403  7747 10372  6018  4159  3004  5756  2903]
164612704005.0
[ 7912  5758  6590  7747 10246  6064  4171  3026  5584  2902]
164489399132.0
[ 7977  5727  6755  7758 10108  6100  4182  3041  5441  2911]
164386610929.0
[8049 5694 6836 7750 9985 6141 4199 3053 5367 2926]
164295853723.0
[8124 5647 6919 7730 9887 6187 4210 3047 5281 2968]
164224018375.0
[8175 5594 6987 7716 9805 6209 4216 3057 5256 2985]
```

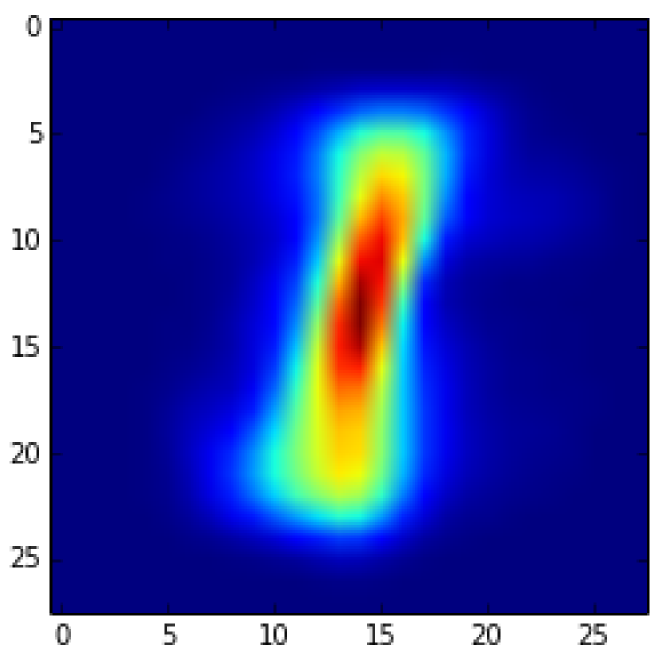
```
164163161464.0
[8226 5535 7041 7713 9732 6198 4234 3052 5267 3002]
164120064778.0
[8254 5573 7080 7691 9643 6181 4248 3051 5257 3022]
164086379019.0
[8288 5588 7092 7656 9575 6130 4257 3071 5317 3026]
164059836544.0
[8313 5614 7108 7622 9510 6088 4253 3099 5368 3025]
164038907404.0
[8322 5650 7114 7572 9458 6055 4263 3113 5427 3026]
164020843443.0
[8332 5677 7114 7541 9421 5989 4277 3126 5492 3031]
164004709596.0
[8328 5661 7112 7504 9394 5966 4289 3140 5567 3039]
```

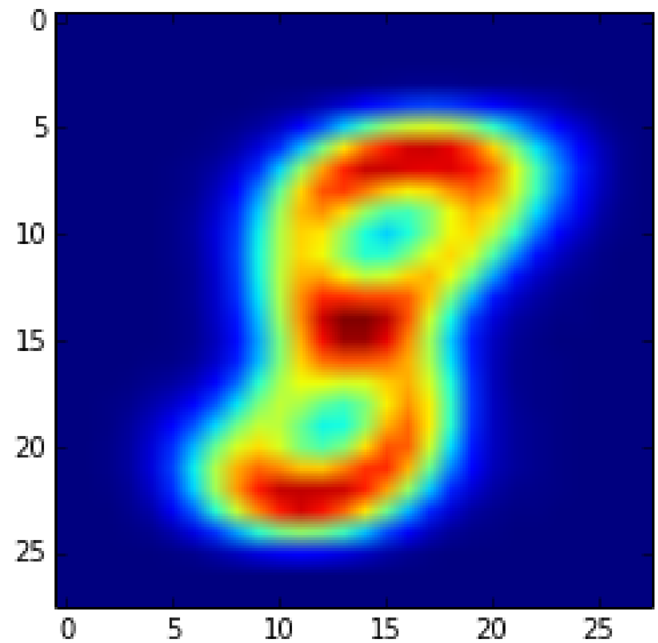


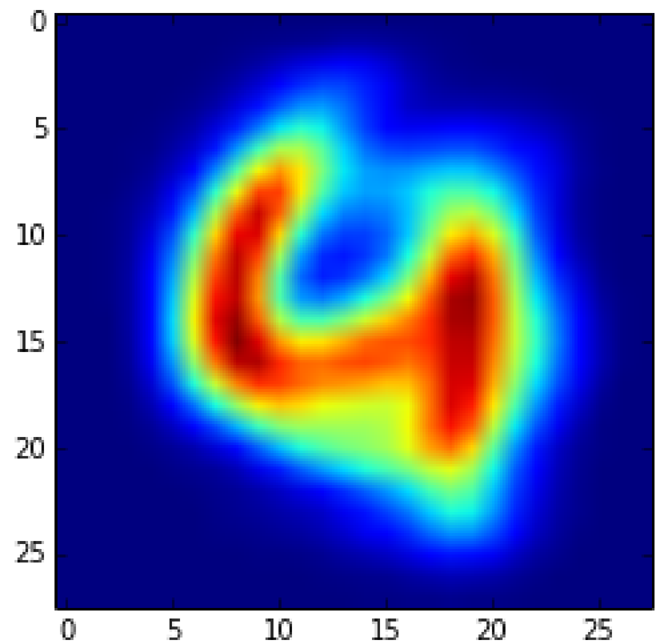

0
5
10
15
20
25
0 5 10 15 20 25

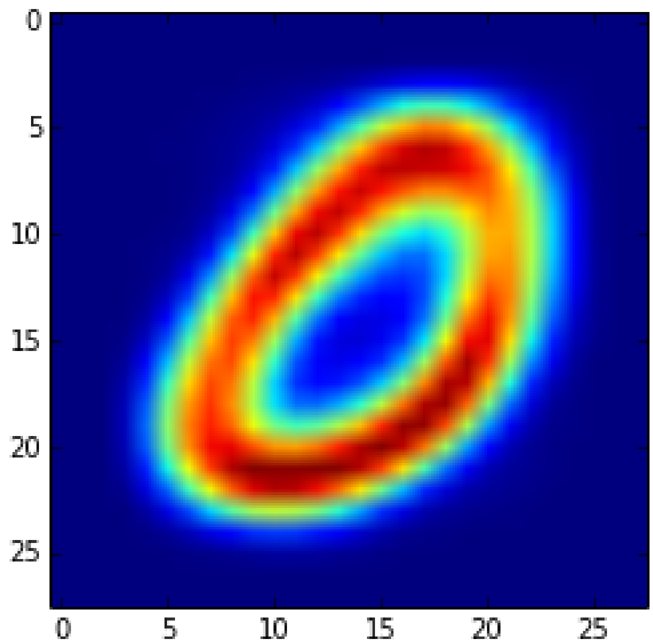

0
5
10
15
20
25
0 5 10 15 20 25

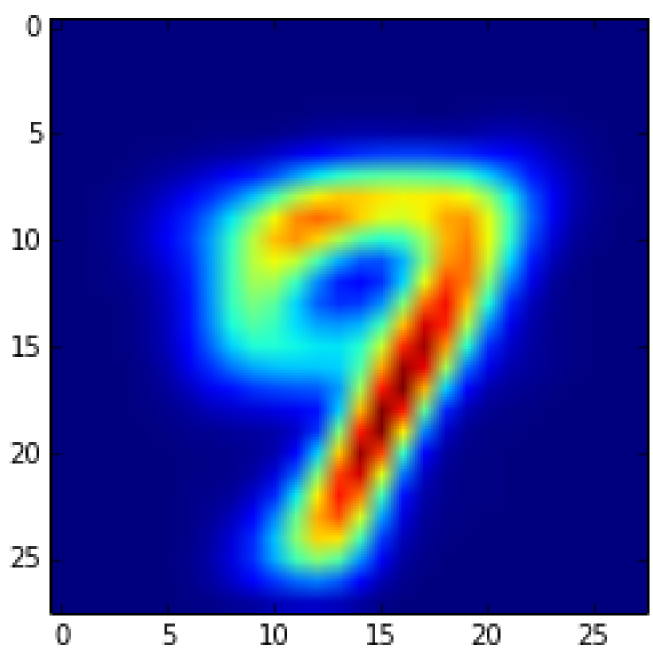

0
5
10
15
20
25
0
5
10
15
20
25

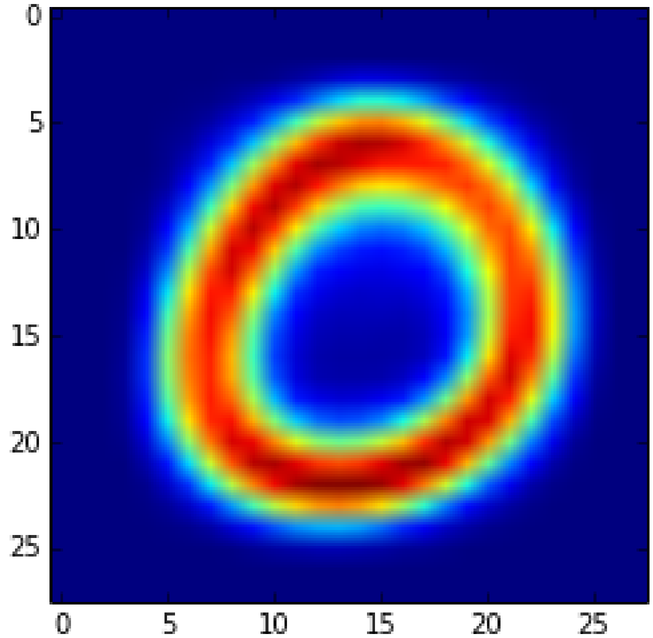

0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25

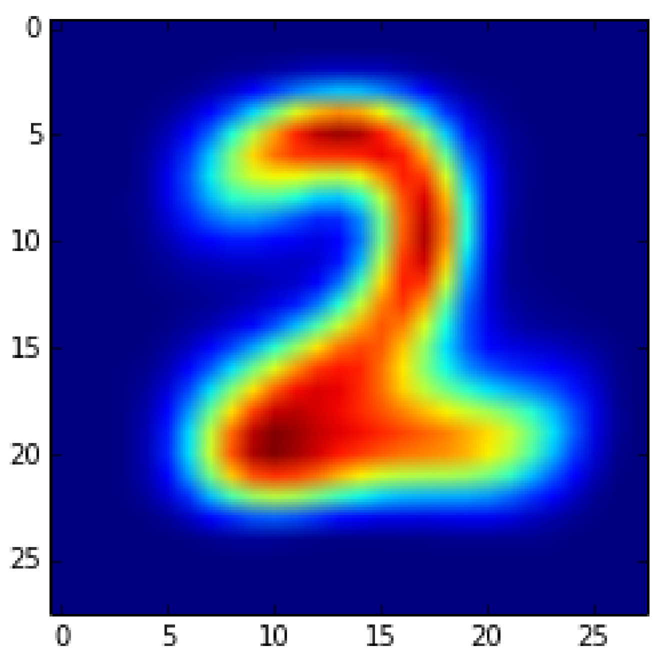

0
5
10
15
20
25
0
5
10
15
20
25

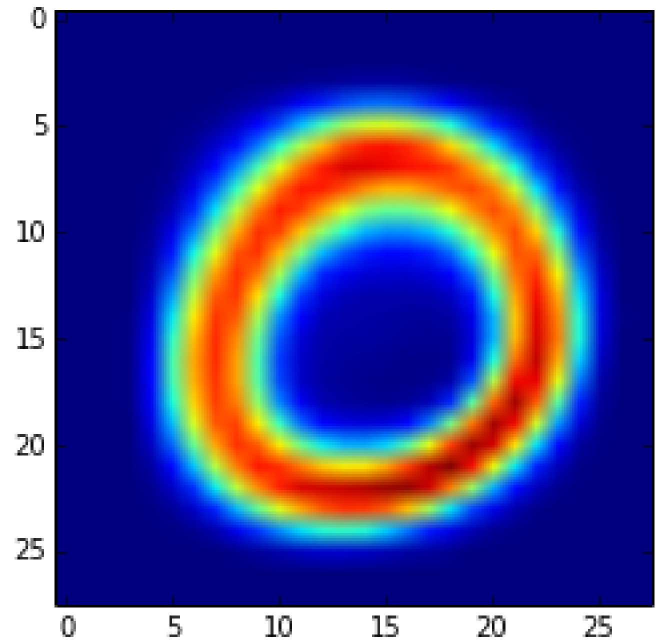

0
5
10
15
20
25
0
5
10
15
20
25

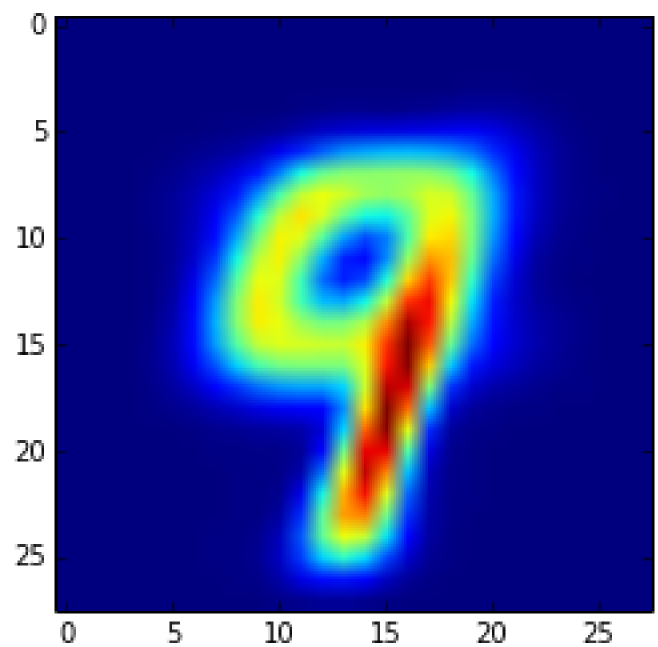


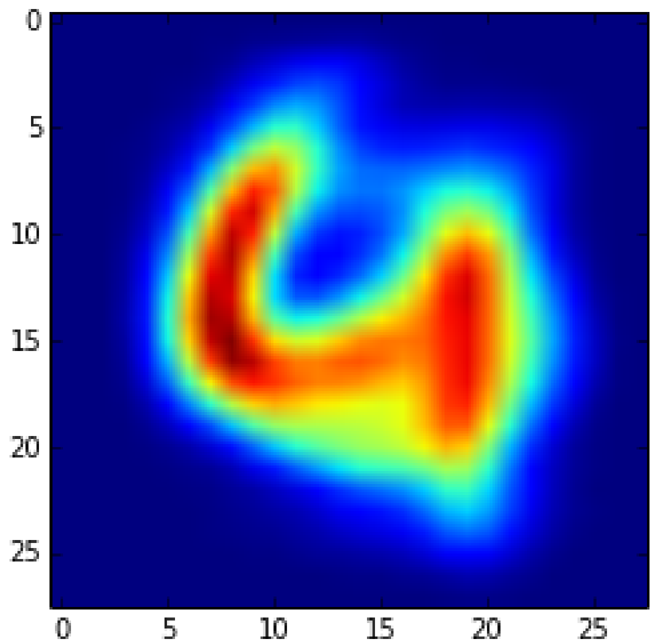

```
166161561525.0
[ 682 1585 3711 1057 1750 2453 5625 3862 1560  771 3823 3733 2710 1644  598
 1381 7555 6997 6561 1942]
156382870389.0
[1807 2050 4031 2077 2225 2871 4766 3634 1789 1509 3964 3203 2960 2964 1517
 1516 5394 5299 4668 1756]
153789352575.0
[2300 1989 4043 2391 2408 2827 4509 3278 1986 1946 3981 3023 3171 3367 1878
 1737 4647 4624 4171 1724]
152514567511.0
[2632 1916 4208 2444 2625 2829 4280 3048 2134 2184 3822 3041 3321 3474 1980
 1991 4274 4322 3794 1681]
151687185593.0
[2914 1872 4339 2467 2790 2891 4062 2912 2228 2306 3598 3127 3395 3409 2041
 2255 4013 4218 3511 1652]
151223213857.0
[3041 1826 4435 2534 2908 3028 3900 2822 2292 2344 3568 3210 3416 3294 2070
 2355 3804 4187 3293 1673]
150932187186.0
[3080 1794 4450 2635 3001 3175 3768 2827 2339 2359 3613 3250 3415 3165 2102
 2394 3617 4166 3168 1682]
150722523546.0
[3080 1777 4440 2735 3088 3310 3727 2843 2357 2361 3660 3286 3393 3059 2132
 2412 3462 4137 3054 1687]
150570344237.0
[3066 1769 4422 2825 3166 3440 3706 2858 2377 2378 3661 3306 3370 2982 2152
 2430 3338 4109 2956 1689]
150448338431.0
[3025 1752 4434 2885 3205 3555 3689 2869 2383 2404 3667 3319 3338 2961 2146
 2446 3243 4098 2887 1694]
```

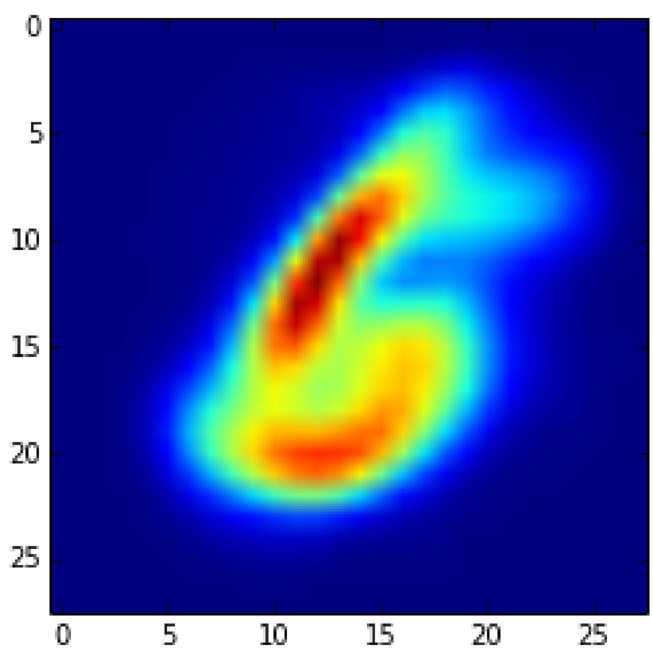
```
150338504057.0
[2945 1720 4447 2911 3262 3656 3689 2885 2391 2430 3650 3339 3317 2929 2137
 2468 3171 4111 2828 1714]
150223273809.0
[2858 1694 4448 2951 3324 3732 3696 2892 2407 2468 3643 3325 3288 2941 2116
 2447 3128 4129 2778 1735]
150092058910.0
[2769 1680 4440 2993 3421 3788 3725 2894 2428 2499 3630 3331 3259 2947 2087
 2399 3097 4151 2717 1745]
149948233696.0
[2695 1666 4456 3040 3546 3814 3748 2881 2438 2527 3631 3340 3247 2975 2077
 2302 3076 4150 2646 1745]
149818902264.0
[2606 1661 4458 3048 3665 3820 3775 2877 2455 2549 3625 3340 3244 3017 2060
 2254 3062 4163 2572 1749]
149719739878.0
[2536 1669 4461 3072 3757 3810 3801 2875 2466 2584 3621 3363 3226 3057 2027
 2226 3059 4143 2498 1749]
149653870028.0
[2515 1682 4464 3076 3813 3786 3806 2875 2478 2620 3604 3366 3195 3088 2008
 2229 3065 4119 2467 1744]
149612996813.0
[2478 1695 4456 3064 3863 3782 3818 2871 2488 2653 3575 3379 3191 3125 1992
 2255 3066 4088 2420 1741]
149584992444.0
[2452 1709 4449 3046 3896 3785 3839 2871 2501 2687 3552 3380 3174 3161 1976
 2275 3053 4075 2390 1729]
```

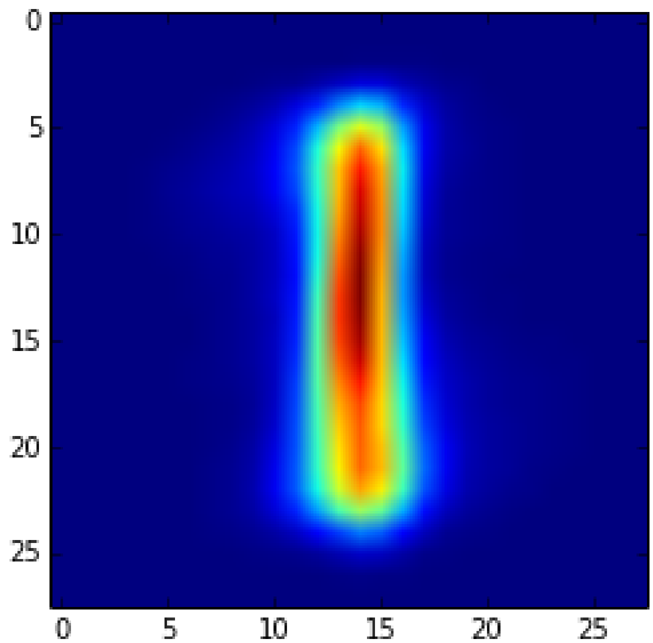


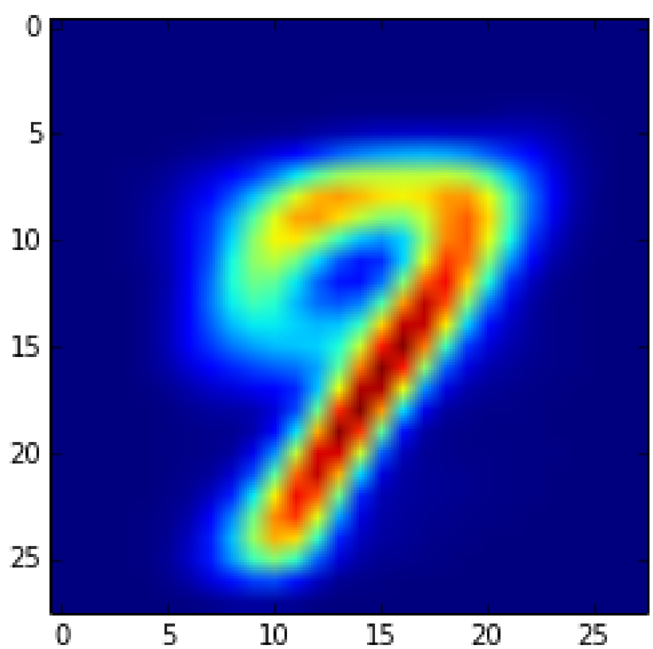

0
5
10
15
20
25
0
5
10
15
20
25

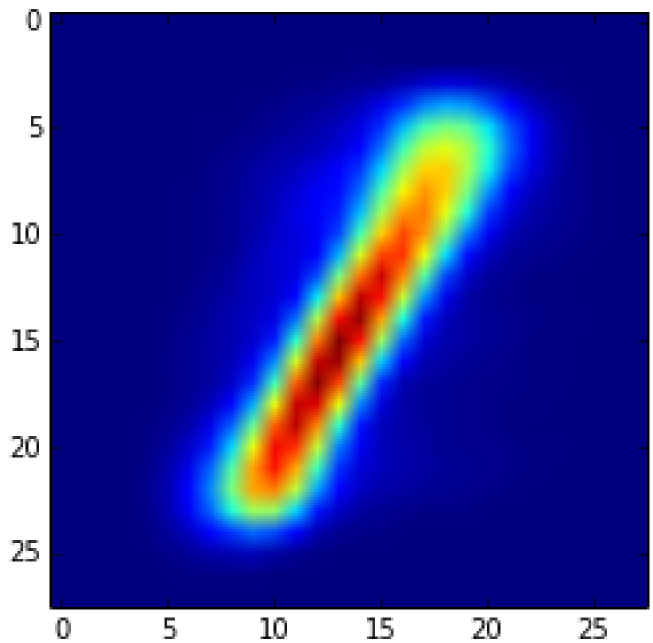

0
5
10
15
20
25
0
5
10
15
20
25

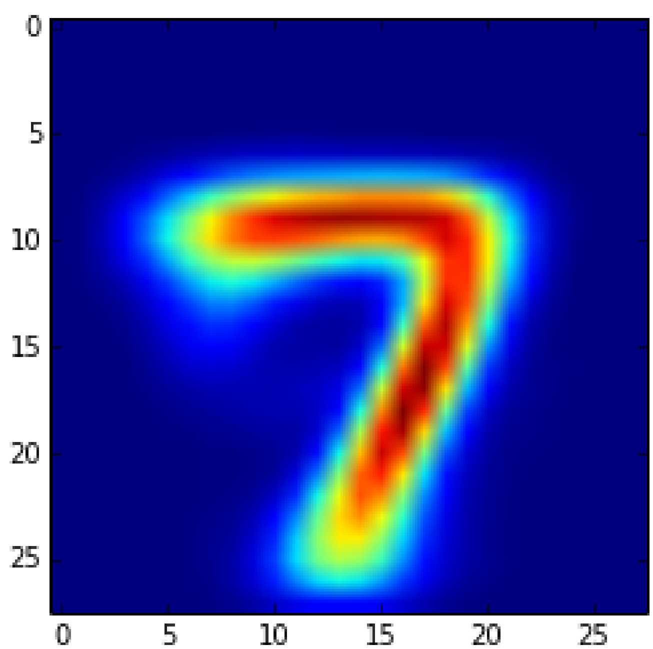

0
5
10
15
20
25
0
5
10
15
20
25

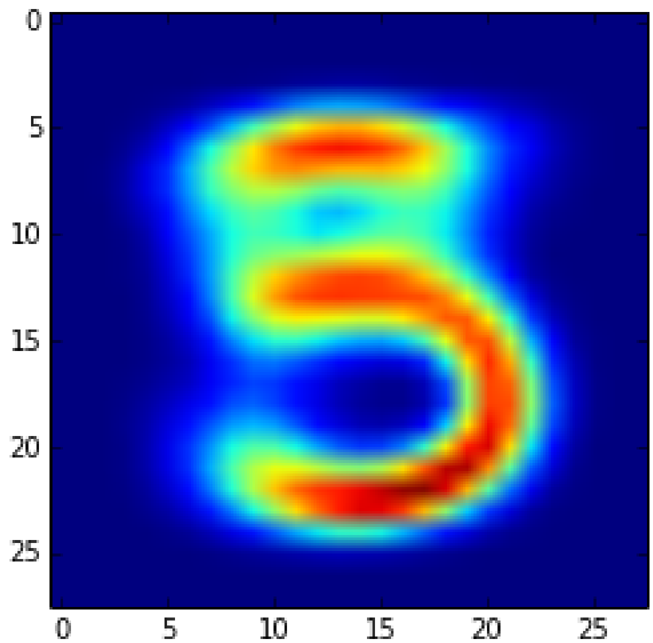

0
5
10
15
20
25
0
5
10
15
20
25

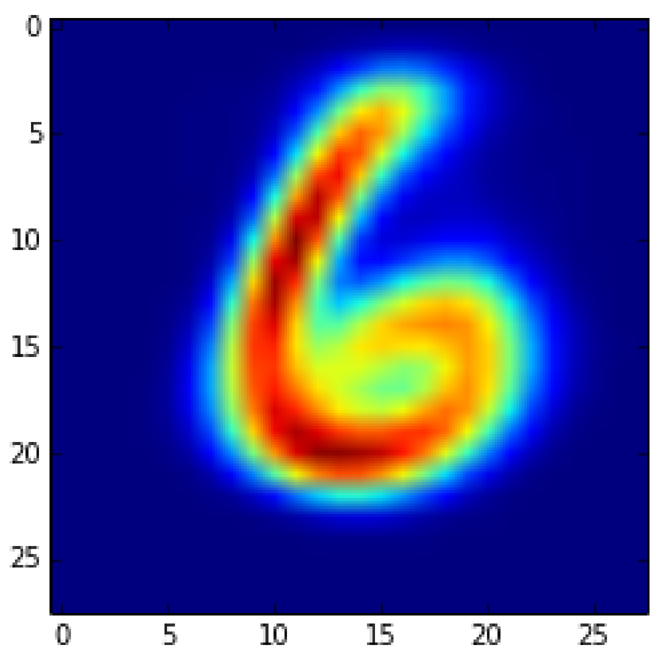

0
5
10
15
20
25
0 5 10 15 20 25

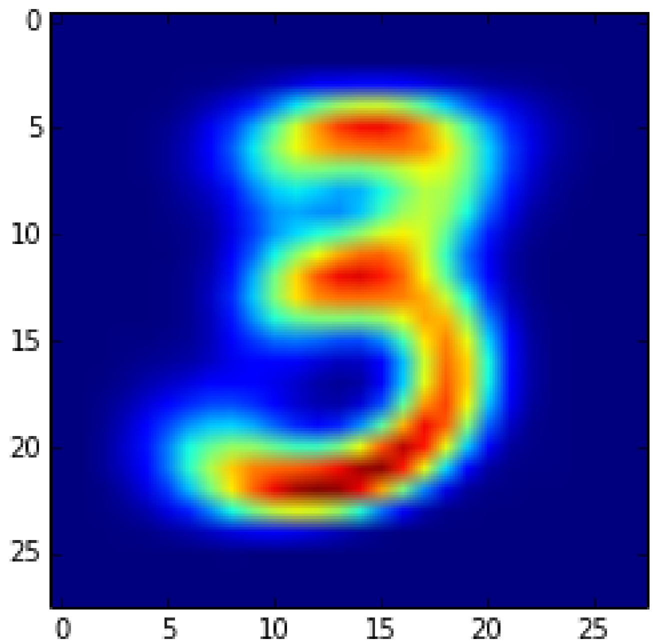

0
5
10
15
20
25
0 5 10 15 20 25

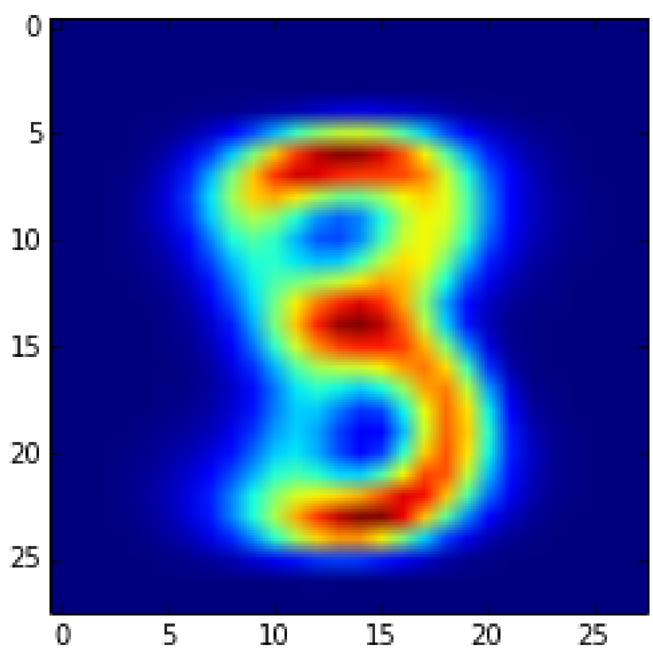

0
5
10
15
20
25
0
5
10
15
20
25

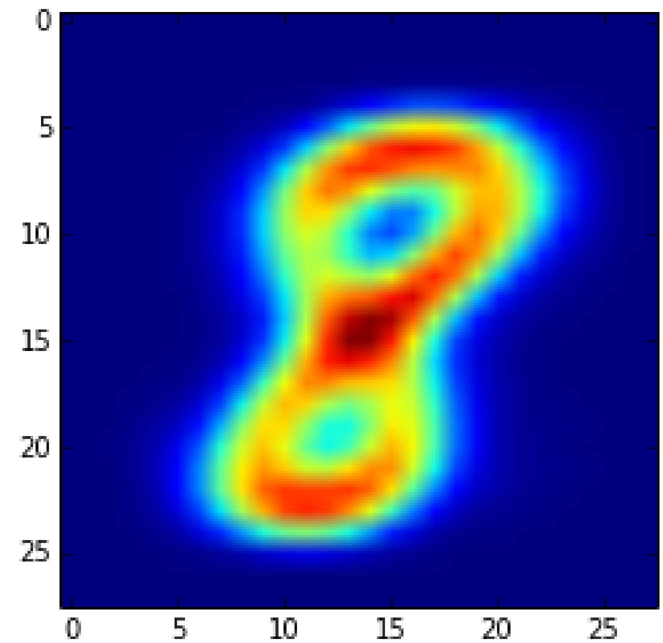

0
5
10
15
20
25
0
5
10
15
20
25

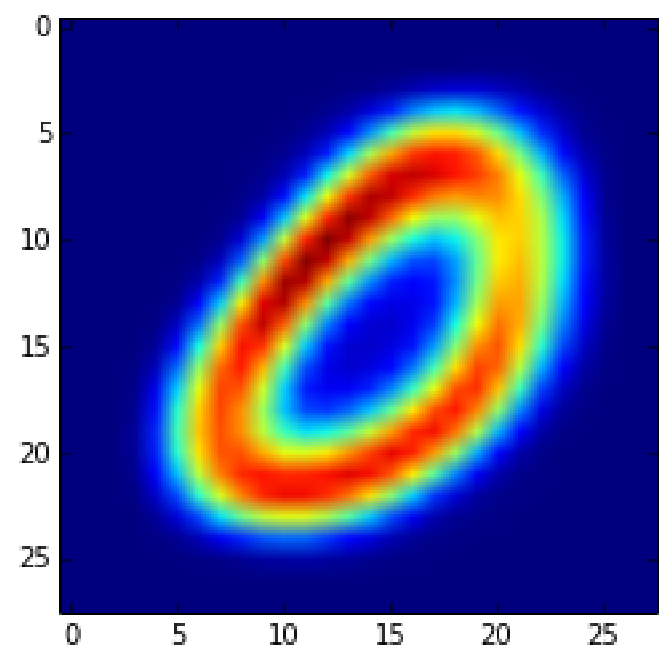

0
5
10
15
20
25
0 5 10 15 20 25

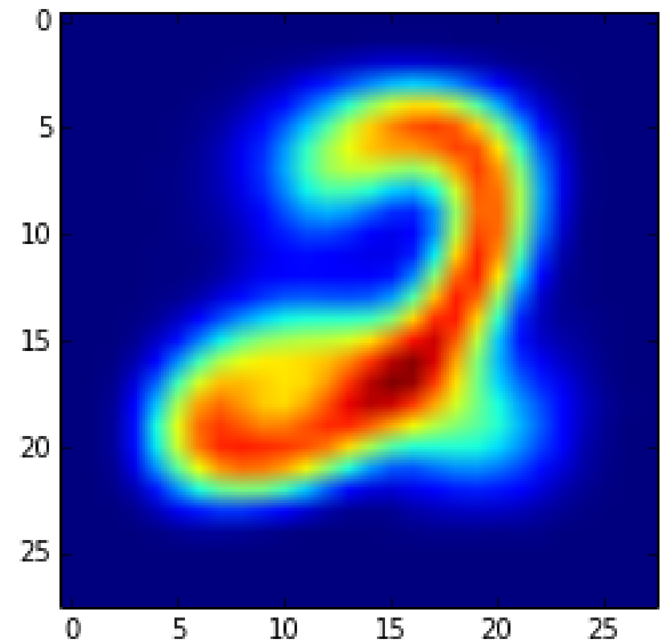

0
5
10
15
20
25
0 5 10 15 20 25

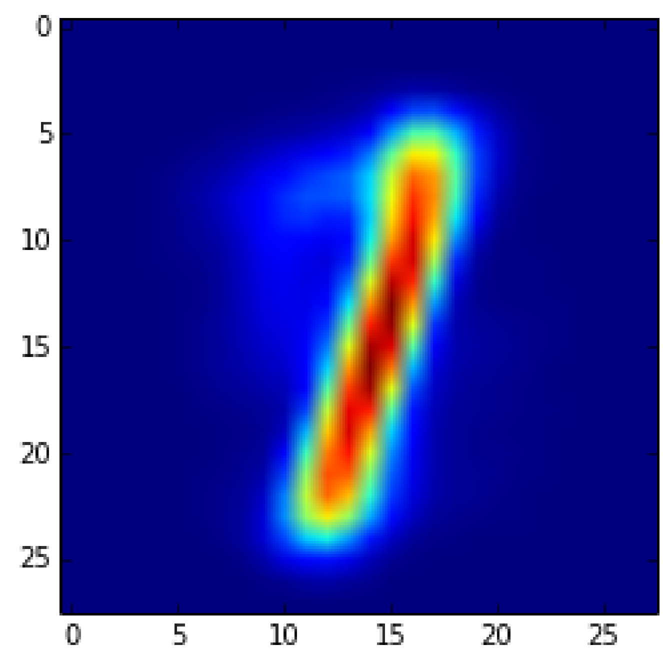

0
5
10
15
20
25
0
5
10
15
20
25

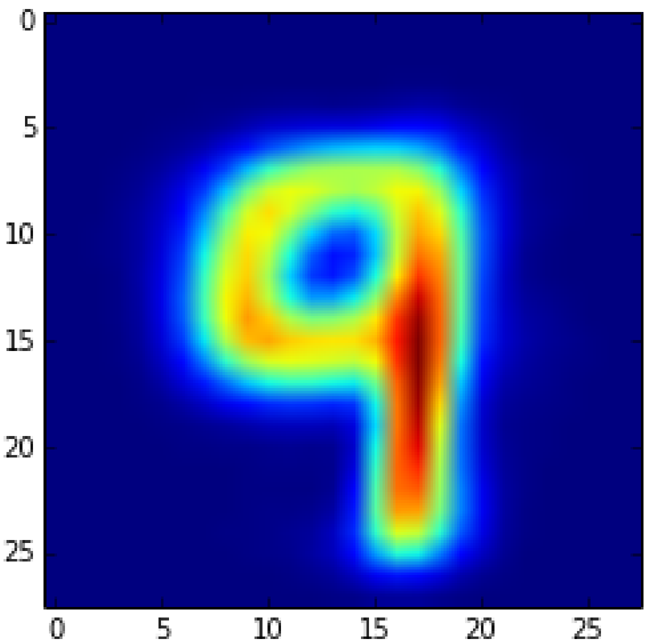

0
5
10
15
20
25
0
5
10
15
20
25

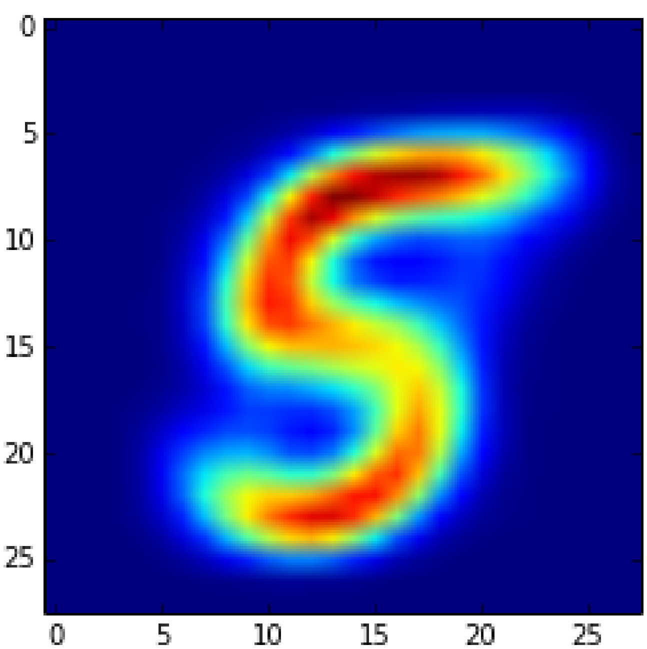

0
5
10
15
20
25
0
5
10
15
20
25

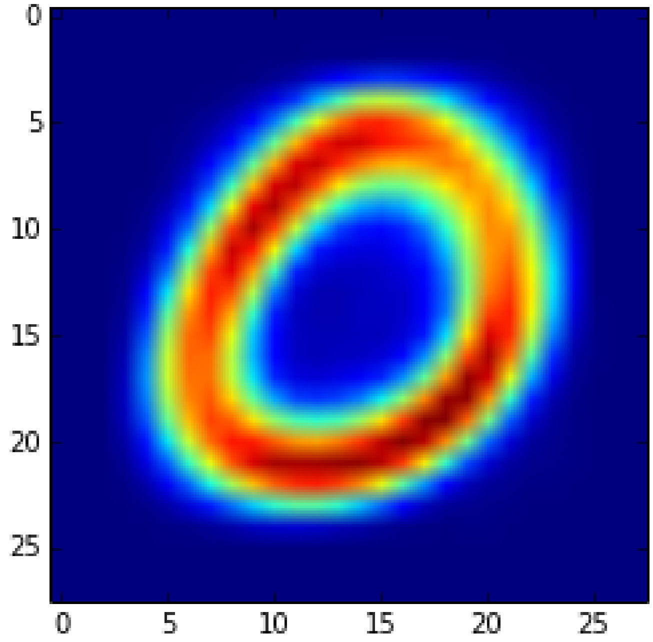

0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25



1.2 2.2 Warm-up

Average rating returns an accuracy of 62.03%. With kNN of k = [10, 100, 1000], we get accuracies of [64.90%, 68.94%, 69.40%].

In [4]:
```
# Load the data
mat = scipy.io.loadmat('joke_data/joke_train.mat')
R = mat['train']
validation = np.loadtxt(open("joke_data/validation.txt","rb"),delimiter=",")

# Zero-index the validation data
validation[:,0] -= np.ones(len(validation))
validation[:,1] -= np.ones(len(validation))
```

In [5]:
```
averages = np.nanmean(R, axis=0)
ratings = [averages[j] for u, j, s in validation]

def score(ratings, validation):
    return sum(v[2] == (1 if r > 0 else 0) for r, v in zip(ratings, validation)) / len(validati

score(ratings, validation)
```

Out[5]: 0.62032520325203255

In [6]:
```
R = np.nan_to_num(R)

def knn(u, R, k):
    distances = np.linalg.norm(R - R[u], axis=1)
    return np.argsort(distances)[1:k+1]
```

```
    for k in [10, 100, 1000]:
        # Memoize k nearest neighbors for each user in the validation set
        neighbors = {u: knn(u, R, k) for u in {v[0] for v in validation}}
        ratings = [np.mean(R[neighbors[u]], axis=0)[j] for u, j, s in validation]

        print(k, score(ratings, validation))
```

```
10 0.649051490515
100 0.689430894309
1000 0.694037940379
```

1.3 2.3 Latent Factor Model

The minimum squared error decreases as d increases, but prediction accuracies stabilize at around d=8, to be about 73%.

```
In [41]: R = np.nan_to_num(R)
         X = R
         # X = R - np.mean(R, axis=0)

         def PCA(X, d):
             U, S, V = np.linalg.svd(X, full_matrices=False)
             users = V.dot(X.T)[:d]
             jokes = U.T.dot(X)[:d]
             return users.T, jokes.T

         for d in [0, 5, 8, 10, 20]:
             users, jokes = PCA(X, d)
             ratings = [users[u].dot(jokes[j]) for u, j, s in validation]
             mse = sum(cell**2 for cell in (users.dot(jokes.T) - mat['train']).flat if not np.isnan(cel
             print("d =", d)
             print("MSE:", mse)
             print("Prediction accuracy:", score(ratings, validation))
             print("-------------")
```

```
d = 0
MSE: 25507785.5073
Prediction accuracy: 0.420054200542
-------------
d = 5
MSE: 7.02857508057e+12
Prediction accuracy: 0.715718157182
-------------
d = 8
MSE: 7.52370927693e+12
Prediction accuracy: 0.726287262873
-------------
d = 10
MSE: 7.81864088295e+12
Prediction accuracy: 0.728184281843
-------------
d = 20
MSE: 9.16652944487e+12
Prediction accuracy: 0.731436314363
-------------
```

In []:
```
l = 10
step = 0.01
# step = 1
Ud, Vd = PCA(X, 5)
Ud = np.random.normal(0, 5, Ud.shape)
Vd = np.random.normal(0, 5, Vd.shape)

R = mat['train']
n, m = R.shape

def loss(Ud, Vd, R, l):
    return sum(cell**2 for cell in (Ud.dot(Vd.T) - R).flat if not np.isnan(cell)) + \
            l * (sum(np.linalg.norm(Ud, axis=1)**2) + sum(np.linalg.norm(Ud, axis=1)))

print("HI")
ratings = [Ud[u].dot(Vd[j]) for u, j, s in validation]
print(score(ratings, validation))
print("HI")

for _ in range(1000):
    i = random.randrange(0, n)
    # print("Ud[i]", Ud[i])
    # print("R[i]", R[i])
    dLdui = sum(2 * (Ud[i].dot(Vd[j]) - R[i][j]) * Vd[j] for j in range(m) if not np.isnan(R[i][
#     print(step * dLdui)

    j = random.randrange(0, m)
    dLdvj = sum(2 * (Ud[i].dot(Vd[j]) - R[i][j]) * Ud[j] for i in range(n) if not np.isnan(R[i][

    Vd[j] -= step * dLdvj
    Ud[i] -= step * dLdui

#     if _ % 20 == 0:
#         print(step * dLdvj)
#         ratings = [Ud[u].dot(Vd[j]) for u, j, s in validation]
#         print(score(ratings, validation))

print("HI")
ratings = [Ud[u].dot(Vd[j]) for u, j, s in validation]
print(score(ratings, validation))
print("HI")
```

In [39]:
```
# Load and zero-index the query data
query = np.loadtxt(open("joke_data/query.txt","rb"),delimiter=",")
query[:,1] -= np.ones(len(query))
query[:,2] -= np.ones(len(query))

# Use Latent Factor Analysis to predict ratings
Ud, Vd = PCA(X, 8)
ratings = [Ud[i].dot(Vd[j]) for id, i, j in query]
result = [1 if rating > 0 else 0 for rating in ratings]

# Write the results to a csv
f = open('kaggle_submission.txt', 'w')
```

```
f.write('Id,Category\n')
for i in range(len(result)):
    f.write("{0},{1}\n".format(i + 1, result[i]))
f.close()
```