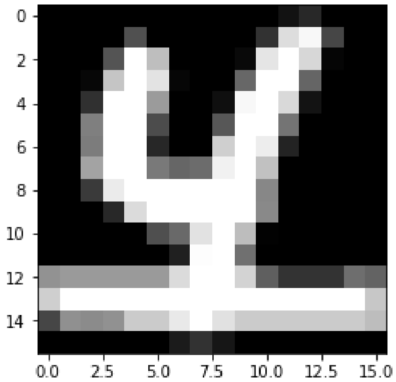
In [2]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt



class KNNClassifier:
    def __init__(self, train_x, train_y):
        self._train_x = train_x
        self._train_y = train_y

    def classify(self, X, k):
        def classify_single(x):
            distances = np.sqrt(np.sum(np.square(self._train_x - x), axis=1))
            nearest_labels = self._train_y[np.argpartition(distances, k)[:k]]
            predicted = np.argmax(np.bincount(nearest_labels))
            return predicted
        if X.ndim == 1:
            return classify_single(X)
        elif X.ndim == 2:
            y_pred = np.empty(len(X), dtype=np.uint8)
            for i in range(len(X)):
                y_pred[i] = classify_single(X[i])
            return y_pred
        else:
            raise ValueError('invalid input shape')


if __name__ == '__main__':
    _training_data = np.array(pd.read_csv('zip.train', sep=' ', header=None),
                              dtype=np.float32)
    _test_data = np.array(pd.read_csv('zip.test', sep=' ', header=None),
                          dtype=np.float32)

    _train_x = _training_data[:, 1:-1]
    _train_y = _training_data[:, 0].astype(np.uint8)

    _test_x = _test_data[:, 1:]
    _test_y = _test_data[:, 0].astype(np.uint8)

    for k in range(1, 16):
        _n_test_samples = 2007
        _knn = KNNClassifier(_train_x, _train_y)
        _y_pred = _knn.classify(_test_x[:_n_test_samples], k)
        _accuracy = np.sum(np.equal(_y_pred, _test_y[:_n_test_samples])) / len(_y_pred)
        _true_neg = _test_x[:_n_test_samples][np.where(np.not_equal(_y_pred, _test_y[:_n_test_sampl
es]))]
        _conf_m = pd.crosstab(pd.Series(_test_y[:_n_test_samples], name='Actual'),
                              pd.Series(_y_pred, name='Predicted'))
        plt.imshow(_true_neg[np.random.randint(0,_true_neg.shape[0])].reshape(16,16), cmap='gray')
        print('k', k)
        print('Accuracy', _accuracy)
        print(_conf_m)
        plt.show()
```

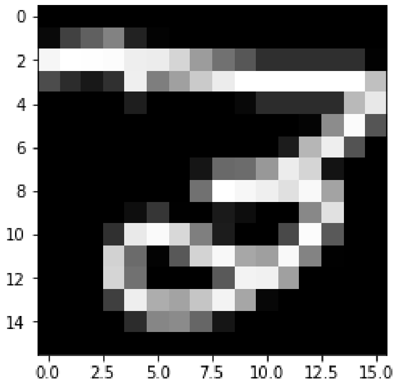
```
k 1
Accuracy 0.9436970602889886
Predicted    0    1    2    3    4    5    6    7    8    9
Actual
0          355    0    2    0    0    0    0    1    0    1
1            0  255    0    0    6    0    2    1    0    0
2            6    1  183    2    1    0    0    2    3    0
3            3    0    2  154    0    5    0    0    0    2
4            0    3    1    0  182    1    2    2    1    8
5            2    1    2    4    0  145    2    0    3    1
6            0    0    1    0    2    3  164    0    0    0
```

6	0	0	1	0	2	3	164	0	0	0
7	0	1	1	1	4	0	0	139	0	1
8	5	0	1	6	1	1	0	1	148	3
9	0	0	1	0	2	0	0	4	1	169



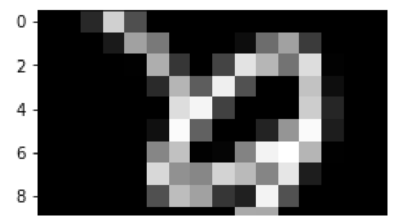
k 2
Accuracy 0.9412057797708022

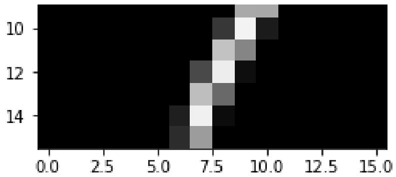
Predicted	0	1	2	3	4	5	6	7	8	9
Actual										
0	355	0	3	0	0	0	0	0	0	1
1	0	259	0	0	3	0	1	1	0	0
2	10	1	181	1	2	0	0	2	1	0
3	3	0	2	156	0	4	0	0	0	1
4	0	3	4	0	185	1	2	3	0	2
5	4	1	2	8	0	143	0	0	1	1
6	4	0	1	0	2	2	161	0	0	0
7	0	2	1	1	4	0	0	139	0	0
8	6	0	3	5	1	1	1	1	146	2
9	1	1	1	0	5	1	0	4	0	164



k 3
Accuracy 0.9446935724962631

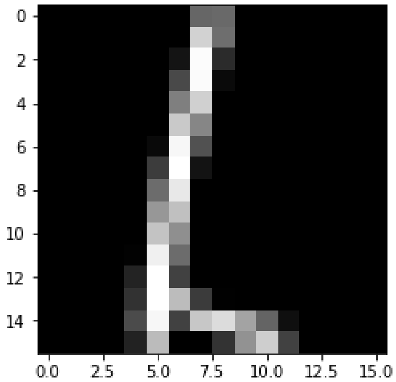
Predicted	0	1	2	3	4	5	6	7	8	9
Actual										
0	355	0	3	0	0	0	0	0	0	1
1	0	258	0	0	3	0	2	1	0	0
2	8	0	183	1	1	0	0	2	3	0
3	3	0	2	153	0	6	0	1	0	1
4	0	2	0	0	183	2	2	2	1	8
5	5	0	3	3	0	144	0	0	1	4
6	3	1	1	0	2	0	163	0	0	0
7	0	1	1	1	4	0	0	138	1	1
8	4	0	3	4	0	1	0	1	151	2
9	2	0	0	0	3	0	0	4	0	168





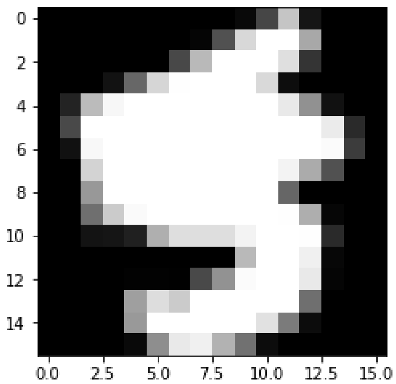
k 4
Accuracy 0.9431988041853513

Predicted / Actual	0	1	2	3	4	5	6	7	8	9
0	355	0	3	0	0	0	0	0	0	1
1	0	258	0	0	4	0	2	0	0	0
2	7	0	183	1	1	0	1	2	3	0
3	3	0	2	155	0	3	0	1	0	2
4	0	3	1	0	184	0	2	2	1	7
5	2	0	2	9	0	143	0	0	0	4
6	4	0	2	0	2	0	162	0	0	0
7	0	2	1	1	3	1	0	137	1	1
8	6	2	1	4	0	2	0	2	148	1
9	1	0	0	0	3	0	0	4	1	168



k 5
Accuracy 0.9446935724962631

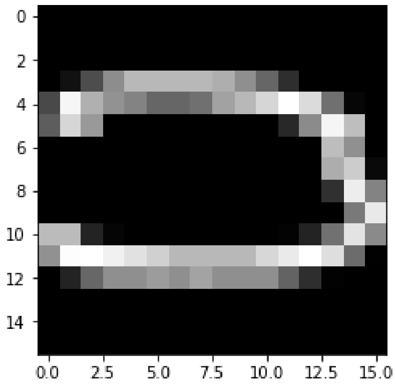
Predicted / Actual	0	1	2	3	4	5	6	7	8	9
0	354	0	3	0	0	0	1	0	0	1
1	0	259	0	0	3	0	2	0	0	0
2	7	0	182	1	1	0	1	2	4	0
3	2	0	2	154	0	5	0	1	0	2
4	0	4	1	0	183	0	2	2	0	8
5	5	0	1	7	0	144	0	0	0	3
6	3	0	2	0	2	0	163	0	0	0
7	0	3	1	0	4	1	0	138	0	0
8	5	0	0	4	0	2	1	1	151	2
9	1	0	0	0	3	1	0	4	0	168



k 6
Accuracy 0.9387144992526159

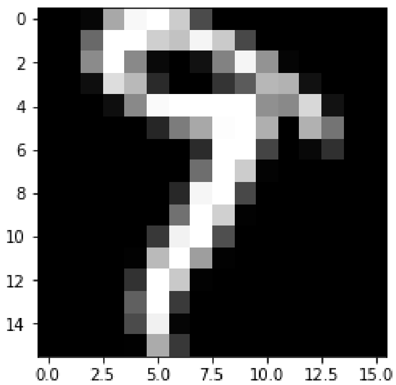
Predicted / Actual	0	1	2	3	4	5	6	7	8	9
0	354	0	3	0	0	0	1	0	0	1

1	0	258	0	0	4	0	2	0	0	0
2	7	0	182	1	1	0	1	2	4	0
3	2	0	2	155	0	4	0	1	0	2
4	0	3	1	0	183	0	2	2	0	9
5	5	0	1	7	0	142	0	0	1	4
6	3	0	3	0	2	0	162	0	0	0
7	0	3	1	1	3	1	0	138	0	0
8	6	3	0	5	0	4	2	2	143	1
9	1	0	0	0	3	1	0	5	0	167



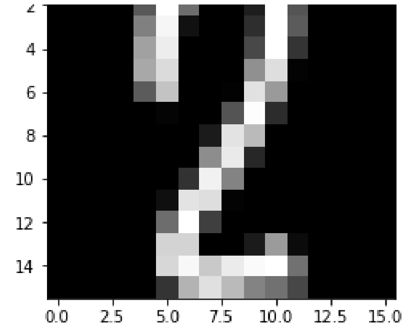
k 7
Accuracy 0.9417040358744395

Predicted Actual	0	1	2	3	4	5	6	7	8	9
0	354	0	2	0	1	0	1	0	0	1
1	0	258	0	0	4	0	2	0	0	0
2	7	0	182	1	1	0	1	2	4	0
3	2	0	2	155	0	4	0	1	0	2
4	0	4	1	0	182	0	2	2	0	9
5	5	0	1	4	0	146	0	0	1	3
6	3	1	2	0	2	0	162	0	0	0
7	0	3	1	0	3	1	0	138	0	1
8	5	3	0	5	0	4	1	1	145	2
9	1	0	0	0	2	1	0	5	0	168



k 8
Accuracy 0.9407075236671649

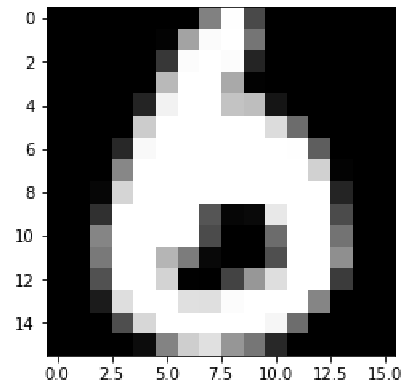
Predicted Actual	0	1	2	3	4	5	6	7	8	9
0	354	0	2	0	1	0	1	0	0	1
1	0	258	0	0	4	0	2	0	0	0
2	7	0	182	1	1	0	1	2	4	0
3	4	0	1	155	0	4	0	1	0	1
4	0	4	2	0	182	0	2	2	0	8
5	5	0	1	4	0	145	0	0	1	4
6	3	0	2	0	2	2	161	0	0	0
7	0	3	1	0	3	1	0	138	0	1
8	6	3	0	6	0	2	1	1	145	2
9	1	0	0	0	2	1	0	5	0	168



k 9

Accuracy 0.9372197309417041

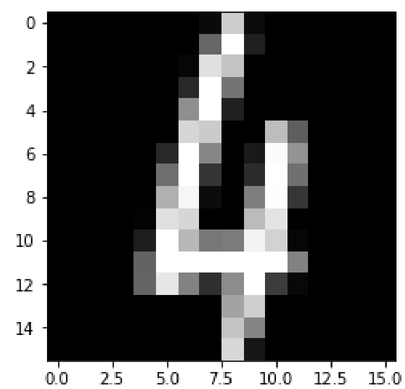
Predicted	0	1	2	3	4	5	6	7	8	9
Actual										
0	354	0	2	0	1	0	1	0	0	1
1	0	259	0	0	3	0	2	0	0	0
2	7	2	180	2	1	0	0	2	4	0
3	3	0	2	154	0	5	0	1	0	1
4	0	4	2	0	179	0	2	2	0	11
5	5	0	1	4	0	145	0	0	1	4
6	3	0	2	0	2	1	161	0	1	0
7	0	3	1	0	4	1	0	137	0	1
8	7	3	2	3	0	3	1	0	144	3
9	1	0	1	0	1	1	0	5	0	168



k 10

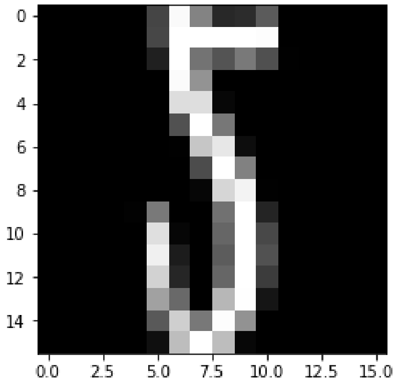
Accuracy 0.9357249626307922

Predicted	0	1	2	3	4	5	6	7	8	9
Actual										
0	354	0	2	0	1	0	1	0	0	1
1	0	258	0	0	3	0	3	0	0	0
2	7	2	180	2	1	0	0	2	4	0
3	3	0	2	153	0	6	0	1	0	1
4	0	4	2	0	181	0	2	2	0	9
5	5	0	1	5	0	143	0	0	1	5
6	4	0	2	0	2	1	160	0	1	0
7	0	3	1	0	4	1	0	137	0	1
8	7	3	0	6	0	2	1	1	144	2
9	1	0	0	0	1	1	0	5	1	168



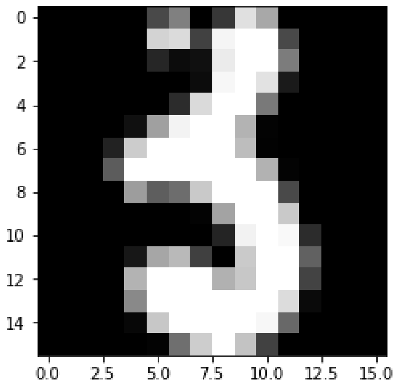
k 11
Accuracy 0.9312406576980568

Predicted	0	1	2	3	4	5	6	7	8	9
Actual										
0	354	0	2	0	1	0	1	0	0	1
1	0	258	0	0	3	0	3	0	0	0
2	7	2	179	2	1	0	1	2	4	0
3	3	0	2	153	0	6	0	1	0	1
4	0	4	2	0	178	0	2	2	0	12
5	6	0	0	4	1	143	0	0	1	5
6	5	0	2	0	2	1	159	0	1	0
7	0	4	1	0	4	1	0	135	1	1
8	7	3	1	6	0	2	2	2	142	1
9	1	0	0	0	1	0	0	5	2	168



k 12
Accuracy 0.9307424015944196

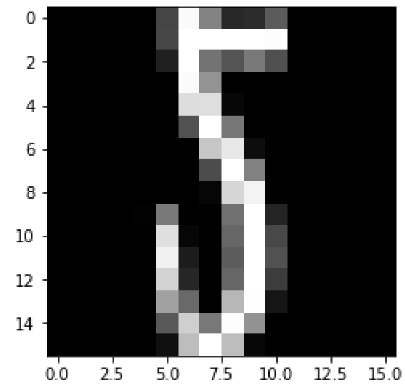
Predicted	0	1	2	3	4	5	6	7	8	9
Actual										
0	354	0	2	0	1	0	1	0	0	1
1	0	258	0	0	3	0	3	0	0	0
2	8	2	179	2	1	0	1	2	3	0
3	3	0	2	151	0	7	0	1	0	2
4	0	4	2	0	179	0	2	2	0	11
5	6	0	2	5	0	141	0	1	1	4
6	5	0	2	0	2	1	159	0	1	0
7	0	4	1	0	4	1	0	135	1	1
8	7	3	0	4	0	3	2	2	144	1
9	1	0	0	0	2	0	0	5	1	168



k 13
Accuracy 0.929745889387145

Predicted	0	1	2	3	4	5	6	7	8	9
Actual										
0	354	0	2	0	1	0	1	0	0	1
1	0	258	0	0	3	0	3	0	0	0
2	8	2	178	2	1	0	1	2	4	0
3	3	0	2	151	0	7	0	1	0	2
4	0	4	2	0	178	0	2	2	0	12
5	7	0	1	6	0	141	0	0	1	4
6	4	0	2	0	2	1	160	0	1	0
7	0	4	1	0	4	1	0	135	1	1
8	7	3	0	6	0	2	2	2	143	1

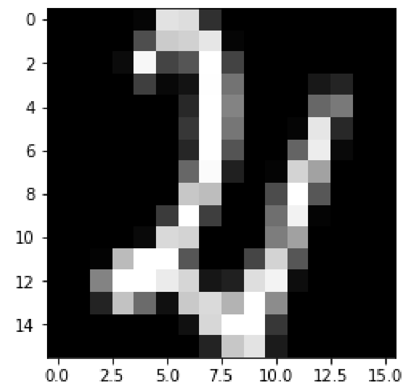
9	1	0	0	0	1	0	0	5	2	168



k 14

Accuracy 0.9292476332835077

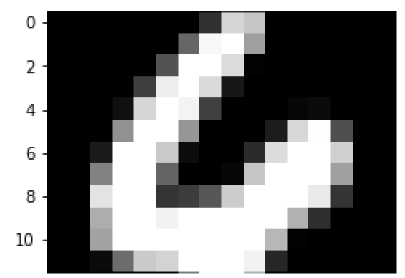
Predicted	0	1	2	3	4	5	6	7	8	9
Actual										
0	354	0	2	0	1	0	1	0	0	1
1	0	258	0	0	3	0	3	0	0	0
2	8	2	179	2	1	0	1	2	3	0
3	3	0	2	151	0	7	0	1	0	2
4	0	4	2	0	179	0	2	2	0	11
5	7	0	1	5	0	140	0	1	1	5
6	5	0	2	0	2	1	159	0	1	0
7	0	4	1	0	4	1	0	136	1	0
8	7	3	1	7	0	2	2	2	141	1
9	1	0	0	0	1	0	0	5	2	168

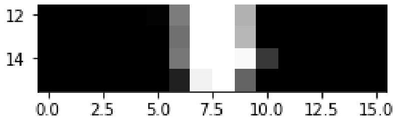


k 15

Accuracy 0.9302441454907823

Predicted	0	1	2	3	4	5	6	7	8	9
Actual										
0	354	0	2	0	1	0	1	0	0	1
1	0	258	0	0	3	0	3	0	0	0
2	8	2	178	2	1	0	1	2	4	0
3	3	0	2	152	0	6	0	1	0	2
4	0	4	2	0	179	0	2	2	0	11
5	7	0	1	4	0	141	0	1	1	5
6	5	0	2	0	2	0	160	0	1	0
7	0	4	1	0	4	1	0	136	1	0
8	7	3	1	7	0	2	2	2	141	1
9	1	0	0	0	1	0	0	5	2	168





In [ ]:

```
Beste Ergebnisse wurden mit dem Werten k=3 und k=5 erreicht.

Vorteile des kNN Classifiers
    - Einfach zum Implementieren
    - Keine Trainigsphase
Nachteile des kNN Classifiers
    - Kurze trainingsphase dafür aber lange testphase
    - Das testen verbraucht sehr viele Resourcen
    - Ungenau bei höheren Dimensionen, wegen kleinen Unterschied der Abstände
      der jeweiligen Datenpunkte
```