1. BoVW model

(a)

- Identify the training and testing data separately
- Determine the image features of the label
- Apply K-Means clustering and construct a visual vocabulary
- Store the generalized cluster centers for comparison
- Classify the images based on vocabulary
- Obtain the most suitable class for the query image

(b) vocabulary size = number of clusters = 100

```
bov = BOV(no_clusters=100)
# set training paths
bov.train_path = "images/train/"
# set testing paths
bov.test_path = "images/test/"
# train the model
bov.trainModel()
# test model
bov.testModel()
```

(c) 95%

#test examples = 20

#correct classifications= 19

```
no. test cases : 20
no. correct classifications : 19
accuracy : 95%
```

(d) SIFT

```
keypoints, descriptors = self.sift_object.detectAndCompute(image, None)
print("These are the features", [keypoints, descriptors])
```

(e) SURF built-in function in OpenCV is used and feature are changed as follow.

```
def features(self, image, type):
    if (type == "SURF"):
        surf = cv2.xfeatures2d.SURF_create()
        keypoints, descriptors = surf.detectAndCompute(image, None)
    if (type == "SIFT"):
        keypoints, descriptors = self.sift_object.detectAndCompute(image, None)
        print("These are the features", [keypoints, descriptors])
    return [keypoints, descriptors]
```

(f) Accuracy = 85%

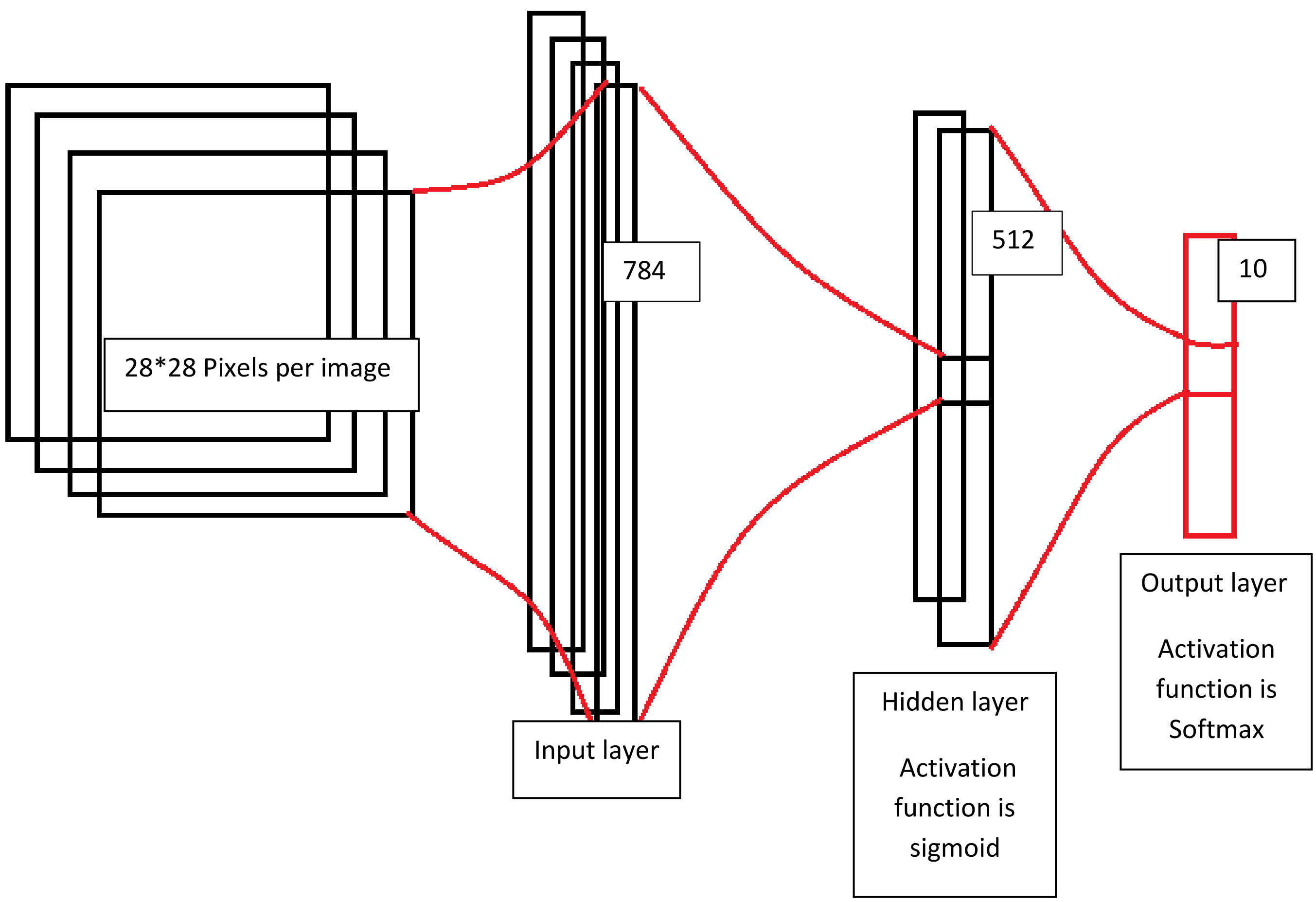
```
no. test cases : 20
no. correct classifications : 17
accuracy : 85%
```

Comparison between SIFT vs SURF

SIFT	SURF
Slower than SURF	Faster than SIFT
Invariance for scale transformations is higher	Lower than SIFT
Invariance for rotation and transform is lower	Higher than SIFT
Invariance for Blur is lower	Higher than SIFT
Accuracy (in my case) = 95%	Accuracy (in my case) = 85%

2. MNIST model

(a)



(b)

```
model = Sequential()
model.add(Conv2D(32, kernel_size=3,padding='same', activation='relu', input_shape=(28,28,1)))
model.add(Dropout(0.2))
model.add(Conv2D(32, kernel_size=3,padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=3,padding='same', activation='relu'))
model.add(Dropout(0.2))
model.add(Conv2D(64, kernel_size=3,padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=3,padding='same',activation='relu'))
model.add(Dropout(0.2))
model.add(Conv2D(128, kernel_size=3,padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.2))
```

(c)

#total parameters: 9,266

#trainable parameters: 9,266

#non-trainable parameters: 0

Test loss: 1.95

Test accuracy: 30.27%

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 3)	228
conv2d_2 (Conv2D)	(None, 26, 26, 1)	28
max_pooling2d_1 (MaxPooling2	(None, 13, 13, 1)	0
flatten_2 (Flatten)	(None, 169)	0
dropout_1 (Dropout)	(None, 169)	0
dense_3 (Dense)	(None, 50)	8500
dense_4 (Dense)	(None, 10)	510

```
Total params: 9,266
Trainable params: 9,266
Non-trainable params: 0

Test loss : 1.9537124439239502
Test accuracy : 0.3027
```

(d)

```
import matplotlib . pyplot as plt
import numpy as np
import keras
from keras . datasets import cifar10
from keras . models import Sequential
from keras . layers import Dense , Dropout , Flatten
from keras . layers import Conv2D, MaxPooling2D
from keras import backend as K
from mnist_helper import *

batch_size = 16
num_classes = 10
epochs = 2

# input image dimensions
img_rows , img_cols = 32 , 32

# the data , s p l i t between t r a i n and t e s t s e t s
(x_train , y_train) , (x_test, y_test) = cifar10.load_data()
print('x_train shape:',x_train.shape)
class_names = ['plane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
# Showing a few examples
show_image_examples(class_names, x_train, y_train)

if K.image_data_format() == 'channels_first' :
    x_train = x_train.reshape(x_train.shape[0], 3, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 3, img_rows, img_cols)
    input_shape = (3, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 3)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 3)
    input_shape = ( img_rows , img_cols , 3)

# Pick every 100 th sample to speed-up ( Set t h i s to 1 in the f i n a l run . )
```

```
step = 1
x_train = x_train[::step, :, :]
y_train = y_train[::step]
x_train = x_test[::step, :, :]
y_train = y_test[::step]

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print ('x_train shape :', x_train.shape)
print (x_train.shape[0], 'train samples')
print (x_test.shape[0], 'test samples')

# convert c l a s s v e c t o r s to binary c l a s s matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
model = Sequential()
model.add(Conv2D(filters=3,kernel_size=(5,5),input_shape=input_shape))
model.add(Conv2D(filters=1,kernel_size=(3,3)))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(50,activation = 'sigmoid'))
model.add(Dense(num_classes, activation = 'softmax'))

model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(),
metrics=['accuracy'])
model_info = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose =1,
validation_data=(x_test,y_test))
model.summary()
score = model.evaluate(x_test, y_test, verbose=0)
print ('Test loss :', score[0])
print ('Test accuracy :', score[1])
plot_model_history( model_info )
```

(e)

#total parameters: 407, 050

#trainable parameters: 407, 050

#non-trainable parameters: 0

Test loss: 0.2141

Test accuracy: 93.9%

```
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
Test loss : 0.2141170334994793
Test accuracy : 0.939
```

we are predicting only for the 10 prominent classes. On the other hand, recognizing hand written digit data can be done easily since they have simple shapes. If we are to predict things such as CIFAR10 with higher accuracy, we need to have a deep CNN with more convolutional and fully connected layers so that these deep layers can predict highly complex shapes and patterns.