CV: Lab 04 Writeup

Benedict Armstrong
benedict.armstrong@inf.ethz.ch

Assignment

This weeks assignment was split into two parts:

- Implementing a Bag-of-words Classifier
- Implementing a CNN-based Classifier for the CIFAR-10 dataset

Bag-of-words Classifier

We implemented a bag-of-words classifier using the following steps:

- Generate bag-of-words histogram from training images
- Extract local features "visual words"
- Given a test image calculate bag-of-words histogram
- Find its nearest neighbor training histogram
- Predict: assign it the category of this nearest training image (0/1)

Local Feature Extraction

We start by extracting local features ("visual words") from the images using the SIFT algorithm as discussed in the lecture. We use a 10 by 10 grid of evenly spaced keypoints to extract features from. Most of this code was already implmented in the skelton. Numpy offers a very neat built-in function `arctan2` which allows us to calculate the angle of each gradient vector and the histogram in two lines of code:

```
angles = np.arctan2(grad_y[start_y:end_y, start_x:end_x],
                    grad_x[start_y:end_y, start_x:end_x])
hist = np.histogram(angles, bins=nBins)
```

Figure 1: Keypoint grid for feature extraction on example image

Bag-of-words Histogram

Next we calculate the "dictonary" or codebook to be used for the bag-of-words histogram. We do this by extracting features from all training images and then using k-means clustering to find k clusters. We then use these clusters as the "words" in our bag-of-words histogram. We use $k = 30$ clusters and $i = 30$ iterations.

This leaves us with k cluster-centers described by 128-dimensional vectors each.

We then use our training images to "learn" the distribution of these words in our training images.

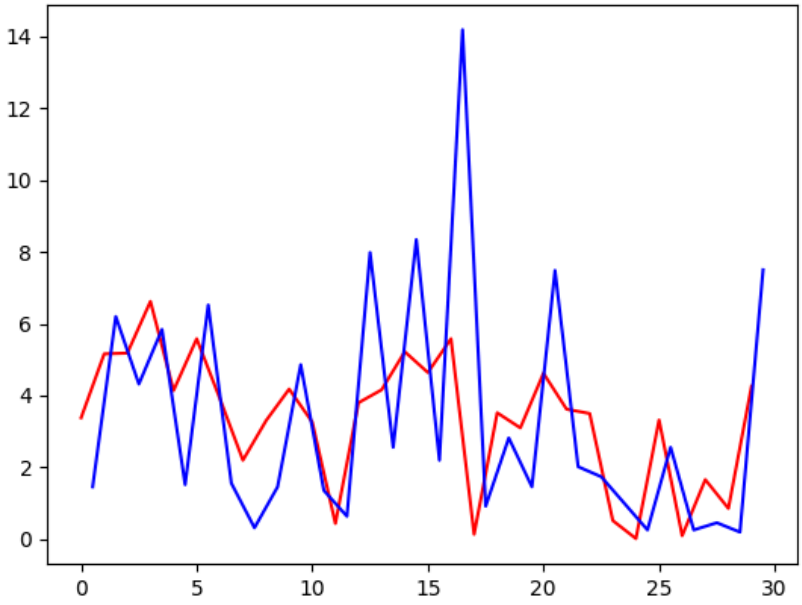

Figure 2: Avergage Bag-of-words occurence for all positive training images (red) and all negative training images (blue)

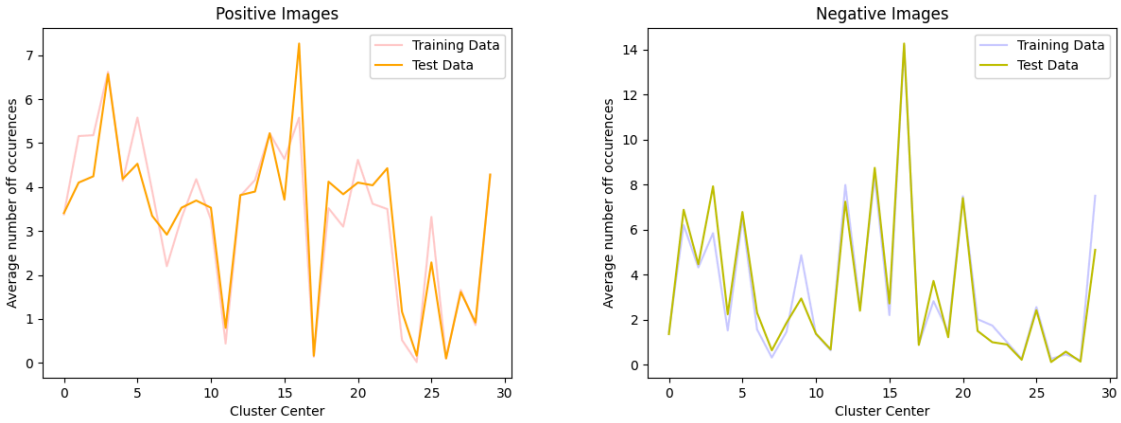

Figure 3: Avergage Bag-of-words occurence training images vs. test images

We can clearly see that the distribution of the bag of words histograms are quite different for the positive and negative training images. We also observe that in both cases the test and the training images have a very similar distribution.

Testing

We then use the bag-of-words histogram to classify the test images. We do this by calculating the bag-of-words histogram for each test image and then finding the nearest neighbor in the training set. We then assign the test image the same label as its nearest neighbor.

The graphs shown above already look promising but what is the actual accuracy? The classifier achieves a test accuracy of 0.97 or more for positive test images and 0.99 or more for the negative test images.

CNN-based Classifier

Architecture

We use a total of 5 convolutional layers with 3 by 3 kernels and 2 by 2 max pooling layers. We use ReLU activation functions and dropout with $p = 0.5$ after the first fully connected layer.

Block Name	Layers	Output Size
conv_block1	ConvReLU (k=3) + MaxPool2d(k=2)	[bs, 64, 16, 16]
conv_block2	ConvReLU (k=3) + MaxPool2d(k=2)	[bs, 128, 8, 8]
conv_block3	ConvReLU (k=3) + MaxPool2d(k=2)	[bs, 256, 4, 4]
conv_block4	ConvReLU (k=3) + MaxPool2d(k=2)	[bs, 512, 2, 2]
conv_block5	ConvReLU (k=3) + MaxPool2d(k=2)	[bs, 512, 1, 1]
classifier	Linear+ReLU+Dropout+Linear	[bs, 10]

Figure 4: CNN architecture (from exercise sheet)

This is easily implemented in code by creating several named layers like for example the first couple of convolutional block:

```
self.conv_block1 = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(2)
)

self.conv_block2 = ...
```

The last step is to implement the `forward()` method. This is done by simply passing the input through the layers in the correct order:

```
x = self.conv_block1(x)
x = self.conv_block2(x)
x = self.conv_block3(x)
x = self.conv_block4(x)
x = self.conv_block5(x)
# reshape to (batch_size, -1)
x = x.view(x.size(0), -1)
score = self.classifier(x)
```

Training

We train the network for 10 epochs using the provided configuration. With this we achieve a test accuracy of 82.62%. The trained model can be found at `runs/4043/last_model.pkl`.