Learning Feature Descriptors Using Capsule Networks and Deep Convolutional Neural Networks

Ashish Rao
Stanford University
aprao@stanford.edu

Mai Lan Nguyen
Stanford University
nmailan@stanford.edu

Melinda Zhu
Stanford University
melinda7@stanford.edu

Abstract

Detecting keypoints and tracking them across images is a fundamental task in computer vision. In this paper, we present a novel method using Capsule Networks to produce robust keypoint descriptors given a local image patch centered at a keypoint. We train our CapsNet on a patch-pair classification task using a combination of a reconstruction loss on the produced feature descriptors and a margin loss that moves feature descriptors of the same keypoint closer together. Our method outperforms the widely-used SIFT handcrafted feature descriptor, as well as a ResNet baseline. Although our model does not outperform the current state-of-the-art for producing feature descriptors, we show that CapsNets are a promising architecture for this task, and that further study may yield significant improvements.

1. Introduction

Image matching is an important concept in computer vision and object recognition. As recognizing objects becomes difficult due to images of the same item being obstructed, taken from any angle, with any lighting and scale, many computer vision techniques aim to find descriptive and invariant features in order to categorize the images. Detecting such visually distinct points in images – known as keypoints – and matching keypoints across images are widely used in computer vision tasks such as visual odometry and SLAM. In order to associate keypoints across images, feature descriptors are used to produce a fixed dimensional representation for that keypoint based on a small image patch centered at that keypoint. If the embeddings for keypoints in two different images are closer than some threshold, they are considered to be the same keypoint. Our project investigates how best to produce these representations for any given keypoint.

Most keypoint descriptors in use today are based on hand-crafted heuristics, such as SIFT, to detect corners and capture information about local gradients. In this project, we investigate learning-based approaches to keypoint description. Specifically, we will investigate the capabilities of Capsule Networks and deep neural networks such as ResNet to produce feature vectors for detected keypoints that can be used to form associations between keypoints across different images.

We will train a neural network that takes as input a small 32x32 image patch centered at a keypoint, and outputs a 128- or 160-dimensional (model-dependent) vector that represents a descriptor for that keypoint. The network will be trained to produce similar embeddings for image patches of the same keypoint and different embeddings for image patches of different keypoints using metric learning. We plan to implement and investigate the Capsule Network architecture for this task, and will compare the model's performance against ResNet as a baseline, as well as the hand-crafted method SIFT and the state-of-the-art triplet loss shallow network [10].

2. Related Work

2.1. Hand-Crafted Feature Extraction

Discriminative representation of local image patches is a major topic in computer science. As many patch-level tasks still rely on hand-crafted feature extraction methods, one such implementation, SIFT, as described by the David G. Lowe paper, is a primary method for patch description. [15] SIFT, or Scale-Invariant Feature Transform is a feature detection algorithm that helps locate the local features in an image (i.e. keypoints), which are scale and rotation invariant. SIFT serves as the optimal point of comparison to non-hand-crafted feature extraction, i.e. convolutional methods. The major advantage of SIFT features is that the descriptors are designed to be minimally affected by the size or orientation of the image. A main downside of SIFT is inefficiency, since the features are computed by hand. An implementation by [4] follows Lowe's method and computes decriptors using a scale space, described in 4.1.

2.2. Convolutional Descriptors

Recent work on keypoint matching demonstrated that local feature descriptors based on CNNs could significantly improve the matching performance. Deeper CNNs have been primarily used for highly performing image classification models, but less so for keypoint description. In this work, we use ResNets as an established baseline model for the production of feature vectors for each image. ResNets, which demonstrate the best performance on image classification tasks, can learn rich feature representations by incorporating residual connections and more complex representational power through more parameters, but with high computational costs and difficulties in optimization [13]. Existing attempts to use ResNets for keypoint detection have involved facial keypoint detection, where ResNets performed better than traditional CNNs, as a result of better keypoint position finetuning and an identity mapping shortcut connection which prevents degradation [17]. However, it is uncertain whether shallow CNNs may perform better than ResNets due to the more subtle features and textures as seen from the MVS dataset in Figure 1.

For the task of patch keypoint matching, Serra et al. utilized a shallow CNN which generalized well across different datasets and performs well with image modifications such as scaling, deformation, and extreme illumination changes [16]. They used pairs of positive and negative patches and Siamese networks with three layers and a nonlinear mapping to train convolutional models for the extraction of image descriptors [16]. In their work, they also implemented an "aggressive mining" strategy where they iterated over "hard negatives" (pairs that are close in the descriptor space and have a high pair loss) which are then back-propagated to update weights. This strategy significantly improves the network's discriminative learning [16].

Balntas et al. extended Serra et al.'s work by utilizing triplets of training samples to learn discriminative CNN representations. This model is considered the current state-of-the-art model, and is implemented as the Tfeat Convolutional Patch Descriptor [9]. This work proved that using triplets results in better descriptors and faster learning. This method still achieved state-of-the-art results while lowering network architecture complexity and reducing dimensionality compared to other CNN descriptors. Balntas et al. also showed that ratio-based loss methods are more suitable for patch pair classification, while margin-loss based methods are more effective for nearest neighbor matching [10]. Attempts at incorporating triplet loss are currently used for tasks such as object tracking and deeper feature representation. Dong et al. used triplet loss in a Siamese network, for the purpose of better capturing relationships between input samples on top of the already powerful feature extraction by the Siamese network [12], which was implemented for the task of learning image embeddings by [8].

2.3. Capsule Networks

Capsule networks have become increasingly prevalent in the area of image processing as a more novel network. Although these networks are not as deep as ResNets, CapsNets were intended as an improvement to CNNs by preserving spatial relationships through transformation matrices and replacing scalar-output feature detectors with vector-output "capsules" instead [14]. In a CapsNet, each convolutional layer is divided into capsules, which represent instantiation parameters such as hue, texture, pose, and deformation. The capsule network from Hinton et al. was trained on MNIST data, but for our project, we adapted and finetuned it to image patches from the MVS dataset. Capsules' ability to capture different entities can be useful for keypoint detection, which requires complex consideration of positioning and spatial features. An existing implementation of CapsNet with PyTorch [5] adapts the network to CIFAR-10 in addition to Hinton et al.'s model which used MNIST.

3. Dataset and Features

We used the MVS Dataset, which includes 32x32 grayscale image patches of the following 3 splits: Statue of Liberty, Notre Dame, and Yosemite, with labels as to which image patches are centered around the same keypoint. To compare results across different methods, we used the Yosemite split. Each split consists of 500k patch pairs extracted around keypoints. For evaluation, we used 100k patch pairs from each split. Following the method from [10], we also incorporated patch resizing and patch augmentation into the data loading step. The Tfeat model was fully configured to train on this data, whereas CapsNet was originally configured for the MNIST and CIFAR-10 datasets, which consist of full images rather than image patches. Furthermore, the original CapsNet implementation outputs a single predicted classification label for each image, but for our application of CapsNet to patch matching, we will modify the training loop to output a multi-dimensional vector of key features directly from the dynamic routing step (described in 4.4). In our experiments involving ResNet with transfer learning, we use the Yosemite split of the MVS dataset for finetuning on top of the pretrained ResNet models which used the ImageNet and CIFAR-10 datasets.

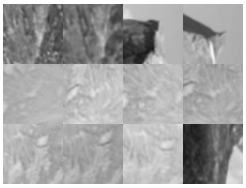

Figure 1. Example images from the MVS Yosemite dataset.

4. Methods

4.1. Hand-Crafted Method: SIFT

SIFT detects image keypoints which are scale and rotation invariant by utilizing three key techniques: **1)** constructing a scale space, where we produce images of different scales and Gaussian blurs. Then, we use the scale space extrema in the difference of Gaussian function to convolve with the image. **2)** We localize the keypoints: we find the local maxima and minima of the images, going through every pixel in the image and comparing it with its neighboring pixels. After this, we remove low contrast keypoints. **3)** We assign the orientation to the keypoints; in this step, we create a histogram for magnitude and orientation. The bin at which we see the peak of the histogram will be the orientation for the keypoint.

Finally, after having found the keypoints that are scale and rotation invariant, SIFT calculates the keypoint feature descriptors (fingerprints that are unique to the keypoint they belong to and are not sensitive to rotation and image illumination) [15]. This is done by first computing the gradient magnitude and orientation at each image sample point in a region around the keypoint location. These samples are then accumulated into orientation histograms summarizing the contents over 4x4 subregions. The descriptor is formed from a vector containing the values of all the orientation histogram entries. The feature vector is also modified to reduce the effects of illumination change; first, the vector is normalized to unit length and then a constant is added to each image pixel. In this way, the descriptor is invariant to affine changes in illumination. We reduce the influence of large gradient magnitudes by thresholding the values in the unit feature vector to each be no larger than 0.2, and then renormalizing to unit length. This means that matching the magnitudes for large gradients is no longer as important, and that the distribution of orientations has greater emphasis.

As the focus of our paper is computing a keypoint feature descriptor for the local image region rather than detecting the keypoints themselves, we simplify the SIFT task and use it only to compute a keypoint descriptor that hopefully is highly distinctive yet is as invariant as possible to remaining variations, such as change in illumination or 3D viewpoint.

Since we know that the image patches in our dataset are centered around a keypoint, we hardcode the center of our chosen keypoint to be in the center of the image patch and set the keypoint size as 32x32 (which are the dimensions of the input image patch). This way, the feature descriptors extracted from SIFT global to the input image patch can serve as a fair hand-crafted comparison to the feature descriptors extracted from the learning based methods described below which need to learn feature descriptors for the entirety of the patch as well (i.e. ResNet and CapsuleNet).

4.2. CNN Baseline: ResNets

For our baseline, we produced feature vectors for each image patch using ResNets of different depths and applied them to our task of using keypoint descriptors to match image patches. We decided to use ResNets as our baseline because of its rich learned feature representations which are already widely used in various applications. ResNet is a very deep network with residual connections and increased representational power compared to shallow CNNs. The original implementation of ResNet was trained on the ImageNet dataset and classified 224x224 RGB input images into 1000 classifications [13]. To adapt ResNet to our feature descriptor task and the MVS dataset, we used transfer learning by finetuning pretrained ResNets on the grayscale 32x32 input images from the MVS dataset. Since the original ResNet was adapted to 224x224 inputs, we instead used an implementation of ResNet that was finetuned on 32x32 inputs from the CIFAR-10 dataset. The purpose of finetuning was to adapt the model to grayscale inputs, which we pre-processed by duplicating the one existing channel so that each image has 3 channels. The output of the final fully connected layers in the ResNet consists of a 128-dimensional feature vector for each image patch. The adapted 32x32 input model consists of the following modified ResNet [6].

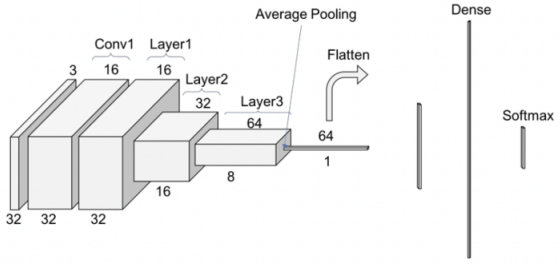

Figure 2. Modified 32x32x16 layer ResNet architecture.

This network starts with a 3x3 convolution with batch normalization (with no max pooling, compared to original ResNet trained on ImageNet, because inputs are now much smaller). Then, the model proceeds with a stack of 6 residual blocks of 3x3 convolutions with batch normalization after each layer and an identity mapping (Figure 3); downsampling is achieved through increasing stride to 2 for the first convolution of each layer. This strategy is used instead of pooling at each layer, which too drastically decreases the input sizes. Max pooling is only used right before the dense layer, instead of average pooling because it is more effective in learning distinct, prominent edges and textures [11].

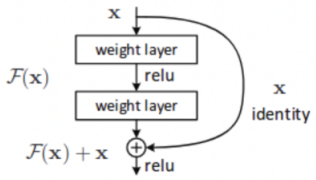

Figure 3. A depiction of residual learning using an identity mapping which in our modified ResNet, is simply zero-padding since the volume should be constant throughout [13].

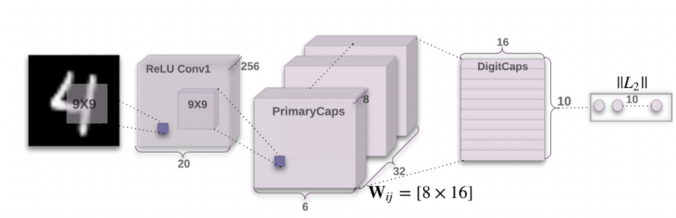

Figure 4. Encoder architecture for CapsNet [14].

4.3. TFeat Shallow Convolutional Patch Descriptors

In addition to deeper CNNs, we use shallow neural networks to compare against SIFT and each other. We use the implementation of the shallow CNN with triplets as described by the Baltnas et al. paper [9], which is the current state-of-the-art model. As input, we feed $x \in R^{n \times n}$ (where $n = 32$), which represents the image patch. The final layer of a convolutional neural network outputs the patch descriptor $f(x) \in R^D$, which represents the D features. Our goal is to learn $f(x)$ s.t. $||f(x_1) - f(x_2)||$ is low if x_1 and x_2 are extracted from the same physical point location (positive match) and high otherwise.

To do this we train from samples of triplets $\{a, p, n\}$ (a - anchor, p - positive match, n - negative match). Here, a and p are different viewpoints of the same physical point, and n comes from a different keypoint. When forming a triplet for training, a positive pair of patches that originate from the same physical point and a patch from another keypoint are randomly sampled. The CNN used for training only has two convolutional layers (which makes it useful for applications requiring real-time processing) and uses Tanh as the activation function.

We use the ranking loss for optimization, where μ is a margin parameter $\delta_+ = ||f(a) - f(p)||$ and $\delta_- = ||f(a) - f(n)||$:

$$\lambda(\delta_+, \delta_-) = \max(0, \mu + \delta_+ - \delta_-) \quad (1)$$

and Stochastic Gradient Descent. Optimising the parameters of the network brings a and p close in the feature space, and pushes a and n far apart. The training is done in batches of 128 items, with a learning rate of 0.1 which is temporally annealed, and momentum of 0.9. The network outputs a feature descriptor vector of dimension 128.

4.4. Capsule Networks

As our main experimental method, we also applied CapsNets, as another example of a shallow network, to patch classification and keypoint detection. While CapsNets have previously been utilized in tasks such as the classification of handwritten digits in the MNIST dataset, our application to keypoint detection in image patches is a relatively novel use of CapsNets.

A key component of CapsNets is the implementation of dynamic routing, or "routing-by-agreement" (instead of max pooling), with the intention of encoding more features. Lower-level capsules encode features that are only a part of a larger classification (e.g. eyes, mouth), while higher-level capsules encode the aggregate classifications (e.g. face). In dynamic routing, the lower-level capsules get selectively routed to the higher levels only if a higher-level classification exists for that lower-level capsule. The dynamic routing algorithm, which incorporates non-linear "squashing" and computes the probability that an entity represented by the capsule is present in the current input, is as follows [14]:

Procedure 1 Routing algorithm.

procedure ROUTING($\hat{\mathbf{u}}_{j|i}$, r, l)
  for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
  for r iterations **do**
    for all capsule i in layer l: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$
    for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}$
    for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$
    for all capsule i in layer l and capsule j in layer $(l+1)$:
      $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i} . \mathbf{v}_j$
  return $\mathbf{v}_j$

A. Encoder

- *Convolutional Layer*: outputs feature detectors that are later redefined by the capsule layers. This layer is comprised of 256 filters, each with size 9×9.
- *Primary (Lower) Capsule Layer*: lower-level capsule layer which identifies the components of objects that will later be routed to the higher-level capsule layer. This layer has 32 different capsules, each applying 8 $9 \times 9 \times 256$ convolutional kernels to the output of the first convolutional layer.
- *Digit (Higher) Capsule Layer*: The primary capsules route to this layer using dynamic routing. This layer outputs 16-dimensional vectors for each image, which contain all the information and spatial features necessary to "rebuild" the image [2].

B. Decoder: takes as input the 16-dimensional vector outputted from the Digit Capsule. This is done through a feed-forward neural network with three fully connected layers,

which decode the instantiation parameters and determine which pieces of information can contribute most for classifying the keypoints, using an L2 distance loss function that determines the similarity between the rebuilt features and actual features from the training images.

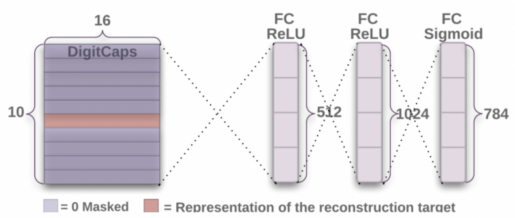

Figure 5. Feed-forward decoder architecture for CapsNet [14].

We will be utilizing aspects of existing implementations from [5] and [1]. We configured the network to train on the MVS dataset, as well as finetuned the architecture of the CapsNet to fit our objective of patch pair matching and incorporating triplet loss evaluation.

5. Experiments, Results & Discussion

Our primary models used for experimentation are the ResNets of various depth, as well as the more novel Capsule Network architecture adapted to the MVS dataset and patch matching task. We compare the performance of these models to each other as well as with the established SIFT algorithm and the state-of-the-art Tfeat convolutional patch descriptor.

5.1. Evaluation Metrics

The performance of the feature descriptors are evaluated on the patch pair classification task. Shown below are ROC curves as well as the performance of each classifier as per the FPR95 metric. The FPR95 metric is computed by computing the ROC curve, finding the point on the curve that corresponds to a 95% true positive rate, and reporting the false positive rate at that value. This method of evaluation avoids the need to set a specific threshold for the distance between the embeddings generated for two image patches under which the prediction is considered to be that the image patches are of the same keypoint.

5.2. SIFT

Currently, the hand-crafted feature descriptor extraction methods such as SIFT remain the most popular tools used for the task. The graph below shows our results for the SIFT method on the 'yosemite' dataset. We see that the AUC is about 0.9067 and the corresponding FPR95 metric is 0.3476. SIFT is a commonly used method in practice, and serves as a strong baseline against which to compare our model.

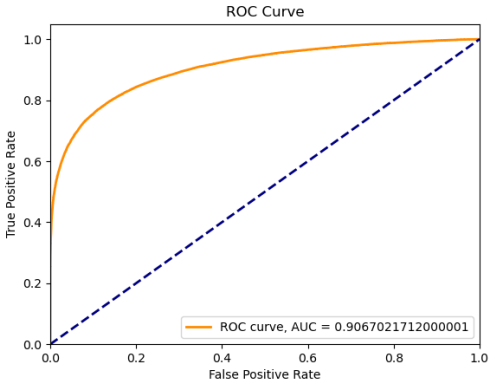

Figure 6. SIFT evaluated on 'yosemite' dataset, FPR95 Metric: 0.3476

Model	AUC	FPR-95
SIFT on Liberty	0.9078	0.3417
SIFT on NotreDame	0.9157	0.3321
SIFT on Yosemite	0.9067	0.3475

Table 1. SIFT results on 'liberty', 'notredame', 'yosemite' datasets.

Table 1 summarizes the results obtained running the SIFT method described in section 4.1. on different datasets. We see that the results are comparable across datasets.

5.3. Baseline: ResNets

For the ResNet baseline, we maintained the experimental configurations used in the original ResNet, using 3x3 convolutions, ReLU activations, and an identity mapping that consisted of zero-padding since we aimed to keep the activation volume (32x32x16) constant throughout the operations of each individual layer. The reasoning for this is that the original ResNet reduces the activation volume by half in addition to downsampling using a stride of 2, since the input images are much larger, i.e. 224x224. In the modified ResNet trained on CIFAR-10 (which we finetuned on the MVS dataset), it prevents such drastic downsampling of the much smaller 32x32 inputs, thus still capturing robust features in the full images. We experimented with different depths of ResNet (18, 32, 50, and 101 layers). The ResNet results for the Yosemite split of the MVS dataset are reported in Table 2 and Figure 6.

As demonstrated through the FPR-95 and AUC metrics, the ResNets, even when finetuned on the MVS dataset, performed poorly overall on the patch pair matching task. Another notable outcome is that as the networks got deeper, the model's performance worsened, which contradicts the general claim that deeper networks can learn more complex,

Model	AUC	FPR-95
ResNet-18	0.7527	0.3731
ResNet-34	0.7562	0.3886
ResNet-50	0.7512	0.4532
ResNet-101	0.7462	0.4532

Table 2. ResNet model results, averaged over dataset.

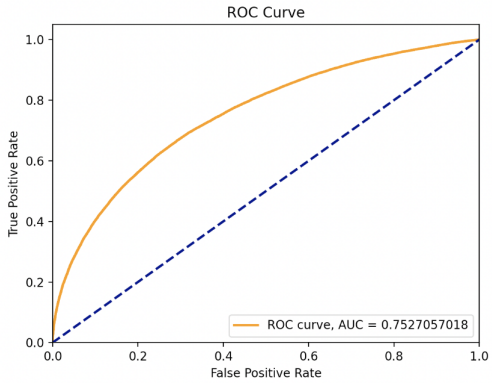

Figure 7. ResNet-18 (the highest performing model for this task) evaluated on 'yosemite' dataset, FPR-95 Metric: 0.3731.

robust feature descriptors. We believe that the reason for the overall poor performance as well as the performance from different network depths is due to the nature of the MVS dataset images; compared to the CIFAR-10 dataset, which presents distinct, full objects in color, the MVS images consist of very subtle object textures in grayscale that do not clearly indicate recognizable object shapes. Furthermore, such subtle features likely led to the significantly deeper networks (i.e. ResNet-50 and ResNet-101) not performing well. Quantitatively, this is likely because the MVS patches are not as complex in color, texture, or hue as the ImageNet or CIFAR-10 datasets, which demonstrated extremely high performance with deeper ResNets for classification tasks (8). Throughout the transfer learning, the feature descriptors learned in the pretained ResNet likely did not translate well to the relatively ambiguous features in the MVS set. As a result, the output feature vectors from ResNet were not precise enough to be matched as pairs with other images due to the lack of consistency and accuracy of the descriptors. For this particular dataset and the application of keypoint matching, other shallow models trained from scratch produced better results in comparison.

5.4. TFeat Descriptors

Based on our results from the ResNet models, we hypothesized that a shallow CNN may produce more suitable feature descriptors. The reported results from the Tfeat convolutional patch descriptor (the current state-of-the-art

Figure 8. Comparison of images from the MVS (left) vs. CIFAR-10 dataset (right).

model) are for the triplet-loss CNN model trained on the 'liberty' split of the MVS dataset (training features image patches of the Statue of Liberty), and the model is evaluated on all three splits of the MVS dataset (below are the results for the Yosemite split). Compared to ResNet, TFeat significantly outperforms the deeper network on all evaluation metrics (AUC = 0.9818, FPR-95 = 0.0873). We believe this is the case because (1) the Tfeat model was trained from scratch exclusively on patches from the MVS dataset, rather than utilizing pretrained feature representations from other more widely used object classification datasets, and (2) the shallower network is able to better capture the more subtle, frequently appearing textures in the MVS patches, while deeper networks such as ResNet may have too pany parameters and complexities for the more vague, simplistic features in the image patches.

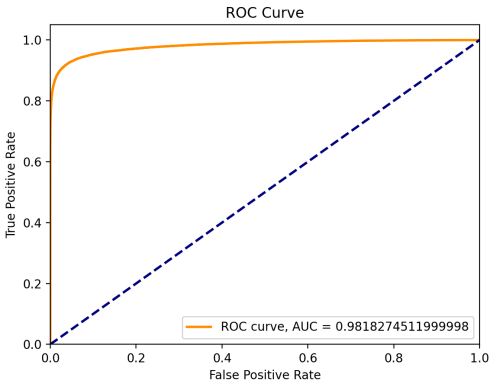

Figure 9. Tfeat evaluated on 'yosemite' dataset, FPR-95 Metric: 0.0873

5.5. CapsNets

Our experimentation with the Capsule Network model primarily consisted of adapting a previous implementation for image classification to the feature descriptor task we study; this involved removing the final layer that produced a 10-dimensional logits vector and taking the 160 dimensional intermediate values produced by the final capsule layer as the network's output. We experimented with dif-

ferent learning rates, as well as different different loss functions – we trained the network using a margin loss and reconstruction loss, as well as just the margin loss. We observed improved performance upon using both loss functions; more details about these loss functions are included below.

The CapsNet architecture we used consisted of a 256 channel 9x9 convolutional layer followed by two layers of capsules. The first layer of capsules consists of 8 capsules that each output 32 channels, and the second layer of capsules consists of 10 capsules that each output 16 channels. This results in a 160-dimensional feature descriptor being produced for each image patch. This architecture was chosen because of its similarity to other CapsNet architectures that have been successfully applied to classification tasks such as MNIST and CIFAR-10.

To supervise the training of the CapsNet, we used a loss function consisting of the sum of a triplet margin loss (the same used by Baltnas et al.) and a reconstruction loss (the L2 distance between input and the CapsNet decoder output). The triplet margin loss incentivizes the model to bring feature descriptors of the same keypoint close together, and the reconstruction loss helps provide additional supervision to ensure that the produced feature descriptors encode important information regarding the input image patches.

To train the network, we used the Adam optimizer with a learning rate of 5e-4 and a batch size of 100. The Adam optimizer was configured to use the PyTorch defaults for the internal β_1 and β_2 parameters (0.9 and 0.999 respectively). We train the network for 5 epochs over our data. We did not perform significant hyperparameter tuning beyond manually adjusting the hyperparameters to ensure a steadily decreasing train loss. The Adam optimizer was used because it maintains per-parameter learning rates and has been shown to empirically result in fast training times.

The graphs pictured below show the model's train loss and validation FPR95 over the course of training on the 'yosemite' split of the MVS dataset. Also pictured is the final ROC curve; the final FPR95 achieved was 0.3134.

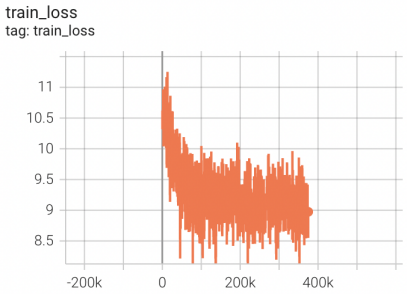

Figure 10. Capsule Net train loss over the course of training

The Capsule Network outperforms the ResNet and SIFT

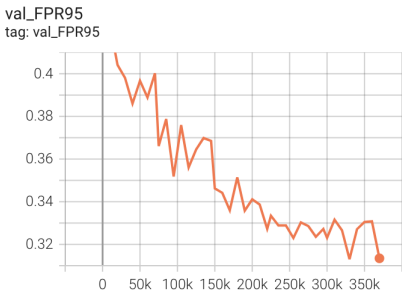

Figure 11. Capsule Net validation FPR95 while training

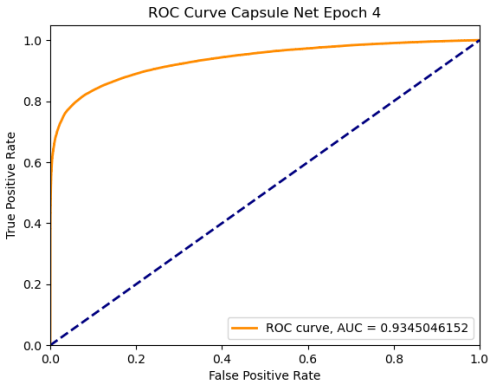

Figure 12. Capsule Net ROC curve

baselines, but fails to reach the performance of the TFeat model. Since the TFeat model is the current state-of-the-art, we did not expect to outperform this method. Additionally, when comparing the capsule network to SIFT, it is important to note that a direct comparison is difficult. This is because SIFT is a method designed to detect keypoints as well as produce feature descriptors. For ease of comparison, we hardcode the keypoint for SIFT to produce a descriptor for to be the center of the image; a similar hardcoding is not present in the ResNet or CapsNet models.

5.6. Overall Analysis

Model	AUC	FPR-95
SIFT	0.9067	0.3475
ResNet-18	0.7527	0.37311
TFeat	0.9818	0.0873
Capsule Network	0.9345	0.3134

Table 3. Summary of baseline and model results on 'yosemite' data.

Among the models that produced the best results closest to the state-of-the-art Tfeat model, we found that us-

ing shallower networks, training from scratch on the MVS dataset, and the Capsule Networks' incorporation of spatial information were the most notable aspects that demonstrated improvements in performance. Comparing the Tfeat model and CapsNet with ResNet, the former two models' shallow network architecture resulted in better learned feature descriptors, possibly due to ResNets' complexities in producing feature vectors not particularly suitable for the grayscale image patches in the MVS dataset. Furthermore, we found pretrained models on common image classification datasets such as ImageNet and CIFAR-10 are ineffective even after performing fine-tuning on the MVS dataset, most likely due to the significant differences in learned features among the various types of images, as demonstrated in 8. We also conclude that CapsNet is able to outperform ResNet and SIFT due to the dynamic routing method in the network that contributes to the preservation of spatial information. Finally, in comparison with the SIFT method, the performances of CapsNet and Tfeat demonstrate that methods using convolutional neural networks to produce feature descriptors indeed perform better than hand-crafted methods.

6. Conclusion & Future Work

Detecting visually distinct keypoints and associating them between images is a highly relevant problem in computer vision and robotics, particularly with object detection. In this paper, we investigate the potential of a learning-based method using Capsule Networks to replace handcrafted feature descriptors, which are widely used in practice today. Our approach outperforms the handcrafted SIFT feature descriptor as well as a baseline ResNet model. The model we propose produces more robust feature descriptors for keypoints than SIFT or ResNet as measured by the FPR95 on a patch-pair classification task as well as the AUC of the resulting classifiers. Our approach does not outperform the current state-of-the-art learning based method (known as the TFeat model), but our results show that Capsule Networks are a competitive choice.

Future work in this direction may include performing hyperparameter tuning to ensure the Capsule Network performs at its best, performing Neural Architecture Search to optimize the model, or investigating modifications to the Capsule Network architecture such as an alternative to the agreement-based routing algorithm used here. Our results suggest that such future work can lead to improvements which may outperform traditional CNN-based architectures on producing robust feature descriptors.

7. Appendix

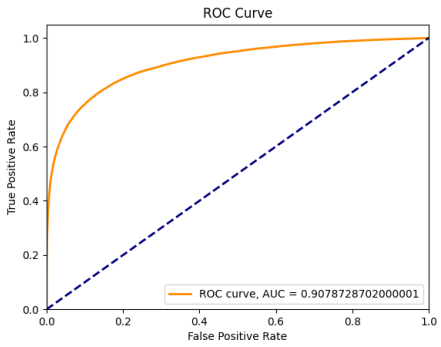

Figure 13. SIFT evaluated on 'liberty' dataset, FPR95 Metric: 0.3417

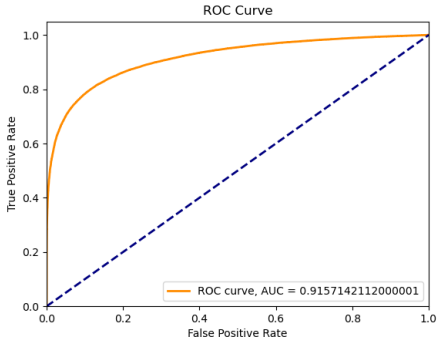

Figure 14. SIFT evaluated on 'notredame' dataset, FPR95 Metric: 0.3321

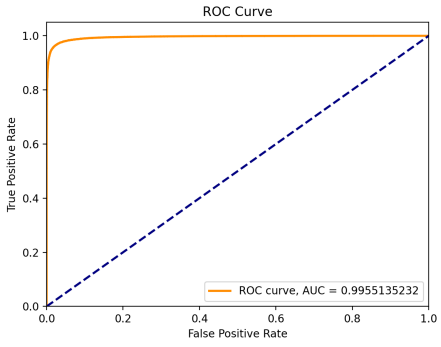

Figure 15. Tfeat evaluated on 'liberty' dataset, FPR95 Metric: 0.0141

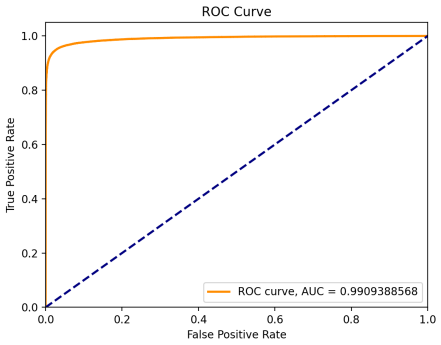

Figure 16. Tfeat evaluated on 'notredame' dataset, FPR95 Metric: 0.0285

8. Contributions & Acknowledgements

We would like to acknowledge our mentor Yinan Zhang, the CS231N course staff for support and guidance, as well as AWS for computing resources. We adapted the following code repositories for implementations of the SIFT, CapsNet, ResNet, and Tfeat models: [3], [5], [7], and [9], respectively.

All team members contributed equally to the final report and poster. Maia was responsible for producing the results, adaptation, description, evaluation and analysis of the SIFT hand-crafted feature extraction method, as well as involved in researching the Triplet Loss Siamese network. Ashish worked on producing the results for the TFeat feature descriptor, adapting the Phototour dataloader to ensure its compatibility with the other code for the project, and running experiments using the Capsule Network architecture. Melinda was responsible for the experimentation, implementation, evaluation, and analysis of the ResNet transfer learning, as well as involved in researching and adapting implementations of the Capsule Network architecture.

References

[1] Barebones cuda-enabled pytorch implementation of capsnet. https://github.com/gram-ai/capsule-networks. Accessed: 2022-05-04. 5

[2] Capsule networks: The new deep learning network. https://towardsdatascience.com/capsule-networks-the-new-deep-learning-network-bd917e6818e8. Accessed: 2022-05-04. 4

[3] Hpatches/sift implementation. https://github.com/hpatches/hpatches-benchmark/blob/master/python/extract_opencv_sift.py#L43. Accessed: 2022-05-23. 9

[4] Introduction to sift (scale-invariant feature transform). https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html. Accessed: 2022-05-23. 1

[5] Pytorch-capsulenet. https://github.com/jindongwang/Pytorch-CapsuleNet. Accessed: 2022-05-03. 2, 5, 9

[6] Resnets for cifar-10. https://towardsdatascience.com/resnets-for-cifar-10-e63e900524e0. Accessed: 2022-05-25. 3

[7] Ross wightman's timm image model library. https://github.com/fastai/timmdocs. Accessed: 2022-05-15. 9

[8] Siamese and triplet learning with online pair/triplet mining. https://github.com/adambielski/siamese-triplet. Accessed: 2022-05-23. 2

[9] Tfeat shallow convolutional patch descriptor. https://github.com/vbalnt/tfeat. Accessed: 2022-05-02. 2, 4, 9

[10] Vassileios Balntas, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. *Proceedings of the British Machine Vision Conference (BMVC)*, pages 119.1–119.11, 2016. 1, 2

[11] Florentin Bieder, Robin Sandkühler, and Philippe C. Cattin. Comparison of methods generalizing max- and average-pooling, 2021. 3

[12] Xingping Dong and Jianbing Shen. Triplet loss in siamese network for object tracking. *European Conference on Computer Vision (ECCV)*, 2018. 2

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 2, 3, 4

[14] Geoffrey E. Hinton, Nicholas Frosst, and Sara Sabour. Dynamic routing between capsules. *31st Conference on Neural Information Processing Systems*, 2017. 2, 4, 5

[15] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004. 1, 3

[16] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. *IEEE International Conference on Computer Vision (ICCV)*, 2015. 2

[17] Shaoen Wu, Junhong Xu, Shangyue Zhu, and Hanqing Guo. A deep residual convolutional neural network for facial keypoint detection with missing labels. *Signal Processing*, 144, 2018. 2