Gaussian parametrization of deep convolutional network filters

Domen Tabernik, Matej Kristan, Jeremy Wyatt, Aleš Leonardis

CN-CR Centre, School of Computer Science, University of Birmingham, Birmingham, UK

I. Introduction

Deep neural networks have achieved remarkable progress in last years, outperforming many other methods on different computer vision problems. In particular, convolutional variant of neural network enabled visual object recognition on higher resolution images, ReLU non-linearity reduced learning time, and improved regularization with weight decay and drop-out overcame the strong overfitting issues common with neural networks. Nevertheless, representations being learned by ConvNets are not well understood. Krizhevsky et al. [1] showed first-layer filters resemble various blob detectors and Gabor filters of different orientations. Zeiler and Fergus [2] further showed corner and edge/color conjunctions appear on the second layer, a more complex texture pattern on the third layer and class-specific parts, such as dog's face or bird's legs, on the fourth layer. While Mahendran and Vedaldi [3] found lower layers to be more accurate in reconstruction of the image, they also found higher layers invert back to compositions of parts similar to, but not identical to, the ones in the original image.

With the exception of the first layers, such visualization techniques point to neural networks encoding features as object parts that are composed from simpler parts and partially correlate with semantic parts of an object. This is partially inherent in the network with a deep, multilayer organization allowing complex features to be composed from simpler ones. However, filters that define features are free to learn any representation they find the best for the optimization criteria used, and the most optimal one for object representation seems to capture features as a composition of a partially semantic parts. That holds true at least to certain degree since not all features will correlate well with semantic parts and, as pointed before, first layers capture mostly blob-like and edge features. Such representation also appears quite natural since all objects we observe have a natural hierarchy of compositions. For instance, a visual representation of a cat is composed from a head, a tail, a couple of legs, and a fur. Furthermore, each one of those can be further composed from yet more simple features. A head is composed from ears, whiskers, tow eyes, a mouth and a nose. All of those features must not only be present, but also appear in specific spatial relation. This represents a clear structure that defines each object.

Despite such a clear structure present in visual domain neural networks do not directly exploit it and do not constrain to specific structure in the data. The only constraint is the assumption of hierarchical structure enforced through deep, multilayer organization. This makes deep networks ideal for problems where very little can be said about the representation, but in visual domain much can be said about it. One such prior knowledge has already been incorporated by LeCun [4] with weight-sharing enforcing the translation invariance in images. However, ConvNet imposes only hierarchical structure but no spatial structure since spatial relation between sub-features is undefined. Feature's response is depended on having underlying sub-features present in the specific pattern. This is governed by a filter which is learned pixel-by-pixel. The size of the filter defines how much of the local neighborhood needs to be considered, but specific spatial relation between a feature and its sub-feature is unconstrained, since filter is unstructured as well and can take any arbitrary form. This can be important in first layer where Gabor filters can be trained without specifying that we are looking for them, but in higher layers it is oblivious of the existing spatial structure. This prevents any kind of semantical interpretation of features as well, since filters are only a bunch of pixel values without any meaning. Despite of freedom in filter learning many filters end up having smooth values with clearly visible spatial structures. Often smoothness of filters is also a requirement to get good results. Furthermore, learning filters directly in pixel-by-pixel manner adds significant amount of optimization parameters, which in return requires a huge amount of data to avoid overfitting and at the same time limits the size of filters that can be used.

In this work we propose to extend the convolutional networks with the assumption of spatial structure present in visual objects. We constrain the structure of filters in convolutional networks to better capture it. This is achieved by parametrizing the filters with Gaussian distributions. Instead of learning individual pixel values in filters we allow Gaussian distributions to govern the filters. This has multiple implications. Firstly, each feature can now be considered as a composition of sub-features where multiple Gaussian components define its characteristics. Instead of defining an occurrence pattern of a sub-feature with an unconstrained filter, it can now be defined with multiple Gaussian components. On one hand this still allows to have an arbitrary filter definition when over-parametrizing with many components per filter, but on the other hand a sparse solution would allow to have a compositional interpretation, since Gaussian components can be seen as definition of sub-feature's characteristics. In fact, Gaussian component directly governs strength as well as spatial position and variability of a sub-feature, and different configurations of components represent different filters, such as, blob and edge detectors on lower layer, or compositions on higher layers. A second implication of Gaussian parametrization is smoothness of filters. Normally this property is an indication of correct learning and parametrization now directly enforces it. This is done through the variance in Gaussian distribution which directly relates to the smoothness of the filter. The last implication is also a reduction in the number of parameters per filter and thus reduction in parameter search-space. A filter of size 9×9 can be described by 3×3 Gaussian components. Assuming Gaussian components are described with four parameters, weight, mean position in two dimensions and variance, this reduces the number of parameters from 81 to 36, and can be further reduced if sparse solution is required.

In this work we focus on showing how parametrization with Gaussian distribution can be incorporated into convolutional neural network. We further analyze how gradient descent and back-propagation can be applied to learn Gaussian parameters, and show it can be successfully applied to image classification. Furthermore, we show ConvNet with filter parametrization using Gaussian distribution can be on one hand considered as a regular convolutional network, while on the other hand it can be considered as a compositional hierarchical model. That means it can act as a bridge between convolutional networks and compositional hierarchies, and opens an opportunity in future to further improve both of them. For instance, powerful optimization with back-propagation and gradient descent could be applied to improve compositional hierarchies, while learning of convolutional networks could be augmented with the explicit spatial structure from compositional hierarchies to allow to learn from principles of compositional hierarchies or to add more control when combining with the structure from other domains, such as using depth information.

The remaining of this paper is structured as follows: in Section II Gaussian parametrization is presented in detail, in Section III experimental evaluation is given and concluding remarks are draw in Section IV.

II. Gaussian parametrization

In this section we present our model in detail. We also show direct relation to deep convolutional networks on one side and compositional hierarchical models on the other side.

We first provide notation for deep convolutional neural networks and then derive our filter parametrization from it. In deep neural network each neuron's activation is modeled with a linear function wrapped around a specific non-linearity:

$$y_i = f(\sum_s w_s \cdot x_s + b_s),$$

with output activation y, input activations x, bias b and weights w determining linear combination of input activations further modified by a non-linear function $f(\cdot)$. Since we are limiting to image domain neuron's activations are organized as 3-dimensional matrix $N \times M \times S$, where two dimensions, $N \times M$, represent image or feature plane, and the third dimension, S, represent different channels. With the introduction of weight-sharing along 2D feature plane the convolutional neural network has the model of neuron's activation map Y defined as:

$$Y_i = f(\sum_s W_s \odot X_s + b_s),$$

where $\odot$ is convolution operation applied between X_s as $N \times M$ activation map from s-th channel and W_s as $A \times B$ filter for s-th channel. This gives $N - {}^{A}/_{2} \times M - {}^{B}/_{2}$ output map for each channel and element-wise summation over channels represents final activation map Y_i after non-linearity $f(\cdot)$ is applied as well. Typically, several activation maps Y_i are created, and network is organized in layers such that X_s in l-th layer is output activation Y_i from $l-1$ layer.

We propose to further parametrize filter weights W_s using a mixture of Gaussian distributions:

$$W_s = \sum_k \tilde{w}_k G(\vec{\mu}_k, \sigma_k), \tag{1}$$

where filter weights are composed from K number of Gaussian components $G(\vec{\mu}_k, \sigma_k)$, each with $\vec{\mu}_k$ mean, σ_k^2 variance and $\tilde{w}_k$ weight factor. Parameters for Gaussian components are different for each filter channel s, but we omit this in notation. It stems from two dimensional filter W_s of size $A \times B$ that $G(\vec{\mu}, \sigma)$ is two dimensional matrix of the same size as well:

$$G(\vec{\mu}, \sigma)[a,b] = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(a-\mu[o])^2 + (b-\mu[1])^2}{2\sigma^2}}.$$

We use two dimensional means but single dimensional variance for simplification. Commonly used filter sizes are 3×3, 5×5 or 9×9; however, such small sizes can lead to significant discretization errors in $G(\vec{\mu}_k, \sigma_k)$. We avoid this by replacing normalization factor computed in continuous space with one computed in discretized space leading to our final distribution function $G(\vec{\mu}, \sigma)$:

$$G(\vec{\mu}, \sigma)[a,b] = \frac{1}{N(\mu, \sigma)} g(a, b, \vec{\mu}, \sigma),$$

where $g(a, b, \vec{\mu}, \sigma)$ is non-normalized Gaussian distribution and $N(\vec{\mu}, \sigma)$ is a sum over non-normalized Gaussian distribution computed for a filter of size $A \times B$:

$$g(a, b, \vec{\mu}, \sigma) = e^{-\frac{(a-\mu[o])^2 + (b-\mu[1])^2}{2\sigma^2}},$$
$$N(\vec{\mu}, \sigma) = \sum_a \sum_b g(a, b, \vec{\mu}, \sigma).$$

Note, that proposed parametrization is similar to Gaussian mixture model; however, in our case we do not enforce $\sum \tilde{w} = 1$ and component weights can take any value, $\tilde{w} \in [-\infty, \infty]$. Having negative weights is important to approximate edges as differences between neighboring components, where positive components can be interpreted as requirement for a presence of a feature and negative components as requirements for an absence of a feature. Nevertheless, this could be also satisfied by normalizing to a sum of absolute components weights, $\sum |\tilde{w}| = 1$, but this further complicates gradient computation and in our case did not help much.

Learning of filter values as weights W_s is now replaced with the learning of several Gaussian components each defined with parameters weight $\tilde{w}_k$, mean $\vec{\mu}_k$ and variance σ_k^2. The number of Gaussian components per filter can be considered as hyper-parameter. Learning can be performed the same as in

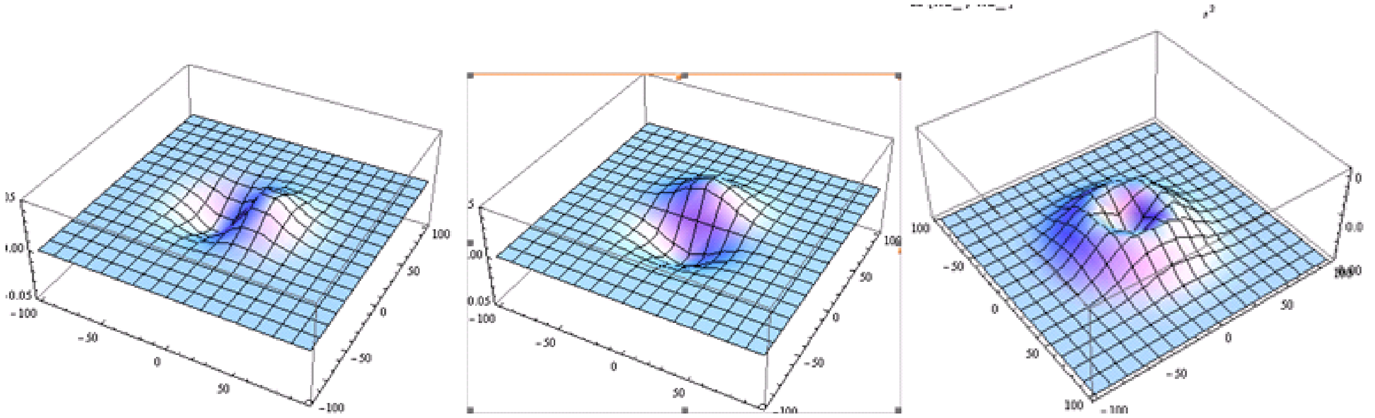

Figure 1. Derivatives for mean $\vec{\mu}_k$ in two dimensions and for standard deviation σ_k used in Gaussian component's gradient for gradient descent learning.

convolutional networks with the gradient descend and back-propagation. Parameters are optimized by computing gradients w.r.t. the cost function $C(y,\bar{y})$, which leads to three different types of gradients. By applying chain rule we can define gradient for component weight ${}^{\partial C}/_{\partial \tilde{w}_k}$ as a dot-product of back-propagated error and input feature X_s convolved with k-th Gaussian component:

$$\frac{\partial C}{\partial \tilde{w}_k} = \sum_{n,m} \frac{\partial C}{\partial Z} \cdot \frac{\partial Z}{\partial \tilde{w}_k} = \sum_{n,m} \frac{\partial C}{\partial Z} \cdot \sum_{a,b} X_s \odot G(\vec{\mu}_k, \sigma_k), \tag{2}$$

where $Z = \sum W_s \odot X_s + b_s$ and ${}^{\partial C}/_{\partial Z}$ is back-propagated error. Note, that only s-th channel of input features are used since weight component $\tilde{w}_k$ appears only in W_s. Back-propagated error is computed the same as in non-parametrized convolutional network:

$$\frac{\partial C}{\partial Z_s^l} = \frac{\partial C}{\partial Z_s^{l+1}} \odot rot(W_s),$$

where back-propagated error from previous layer is convolved with the 180° rotated weight filter $rot(W_s)$ and weight filter itself can be computed from Eq. 1. We can similarly apply chain rule to obtain gradient for mean and variance:

$$\frac{\partial C}{\partial \mu_k} = \sum_{n,m} \frac{\partial C}{\partial Z} \cdot \sum_{a,b} X_s \odot \frac{\partial G(\vec{\mu}_k, \sigma_k)}{\partial \mu_k}, \tag{3}$$

$$\frac{\partial C}{\partial \sigma_k} = \sum_{n,m} \frac{\partial C}{\partial Z} \cdot \sum_{a,b} X_s \odot \frac{\partial G(\vec{\mu}_k, \sigma_k)}{\partial \sigma_k}, \tag{4}$$

where filter derivatives of Gaussian distribution w.r.t. the mean and variance are:

$$\frac{\partial G(\vec{\mu}_k, \sigma_k)}{\partial \mu_k} = \tilde{w}_k \frac{N(\vec{\mu}_k, \sigma_k) \cdot \frac{g(\vec{\mu}_k, \sigma_k)}{\partial \mu_k} - g(\vec{\mu}_k, \sigma_k) \cdot \frac{\partial N(\vec{\mu}_k, \sigma_k)}{\partial \mu_k}}{[N(\vec{\mu}_k, \sigma_k)]^2},$$

$$\frac{\partial G(\vec{\mu}_k, \sigma_k)}{\partial \sigma_k} = \tilde{w}_k \frac{N(\vec{\mu}_k, \sigma_k) \cdot \frac{g(\vec{\mu}_k, \sigma_k)}{\partial \sigma_k} - g(\vec{\mu}_k, \sigma_k) \cdot \frac{\partial N(\vec{\mu}_k, \sigma_k)}{\partial \sigma_k}}{[N(\vec{\mu}_k, \sigma_k)]^2}.$$

Derivatives for mean and variance are also plotted in Figure 1, with first two images depicting derivatives for mean $\vec{\mu}_k$ in two spatial dimensions, and the last image depicting derivative for standard deviation σ_k. Filters can be interpreted as edge detectors for mean positions and as blob detector for standard deviation. In the context of gradient learning filters point to the interpretation of Gaussian components moving away from edges and preferring a more blob-like features.

A. Implementation

Upper definition for computing forward and backward pass can naturally be implemented by utilizing existing computational models for neural networks. Forward pass in particular can be performed with existing forward pass implementations in regular convolutional networks, with the only exception, that filters W_s are generated from Gaussian components. This has to be performed at every change of any of the components, therefore at every iteration. The computational requirements for generating filter depends on several factors: number of components, number of sub-features, number of features and the size of filter. First three factors may lead to explosion and increase the computational needs; however, by limiting the size of the filter this issue can be significantly reduced. Backward pass can also be performed with existing implementation, since the difference is in added convolution of input data with different Gaussian filters. Existing implementations must already perform convolution operation therefore adding new convolutions would not present any significant implementation challenges. Note, that from Eq. 3 and 4 gradients for mean and variance appear complicated; however, they are nothing more then a filter obtained by modifying $G(\vec{\mu}_k, \sigma_k)$. As such they can be precomputed for each iteration and computation reduces to implementing the same algorithm as for Eq. 2.

Gaussian modeling opens different implementation approaches as well. Instead of looking at the Eq. 1 as a single filter we can decompose it to every single component as individual filter. Normally, this would be inefficient since we know that there would be several components per each filter, and it would be better to run convolution only once. However, when there are many features per layer each one will have its own filter for given sub-feature X_s. In regular convolutional network convolution with each filter has to be performed separately, since there is no common structure in filters to exploit. But in our model we know that the structure will always be a mixture of Gaussian distributions. Furthermore, the only difference between different components will be mean position and variance. Different positions of filters would easily be possible to collapse to a single filter, since different positions can be obtained by just shifting the output from a convolution with a non-offseted component. While variance can be discretized to several common values, e.g. step 0.1 between ranges of $[0.5, 5]$ would yield 45 filters, making this implementation efficient when more than 50 features per layers are used. Note, that there are possible limitations to this implementation. For instance, offsetting the output instead of component's position would only work for discretized positions, while discretizing variance would not be possible with multidimensional variance. In this work we do not focus on evaluating different computational implementations, but use a simple extension of existing convolutional networks in our evaluation.

B. *Deep convolutional networks as specialization*

Deep convolutional networks can now be considered as a special case of parametrized convolutional network. We can initialize Gaussian model with $N \times M$ Dirac deltas. If Dirac deltas are evenly spread over the filter of the size $N \times M$ such that each one exclusively covers only one pixel, then a single component weight $\tilde{w}_k$ in Eq. 1 would correspond to a single value in filter weights W_s. In our model we can approximate each Dirac delta with a Gaussian component with a small, near-zero variance. Gradients of mean and variance would, it this case, collapse to zero since derivative of Dirac delta is zero, while gradient of weight would simplify to:

$$\frac{\partial C}{\partial \tilde{w}_k} = \sum_{n,m} \frac{\partial C}{\partial Z} \cdot X_s[\vec{\mu}_k],$$

which is an equation for weight gradient in standard convolutional neural network.

C. *Neural network as compositions*

Gaussian parametrization of weight opens the door for compositional interpretation of the network, similar to implementations of [6]. An output activation map Y_i can be considered as a response map to a presence or an absence of a particular part at different image locations. That part is defined as a composition of several sub-parts who's presence is recursively defined by the previous layer. A response to a particular sub-part corresponds to input activation map X_s, and how a set of sub-parts are composed into a part is governed by corresponding weight filters W_s. Since convolution and correlation are related operations a weight filter W_s defines the blueprint for position of sub-part relative to the center of the filter. Furthermore, since weight filter W_s is composed from Gaussian component, they can now be considered as a model governing a particular sub-part. Mean $\vec{\mu}_k$ defines the offset from the center of the part where the sub-part is searched for, variance σ_k^2 defines how much a sub-part is allowed to vary in 2D spatial dimensions, and weight $\tilde{w}_k$ defines the importance of the sub-part. In particular, higher positive weight $\tilde{w}_k$ value correlates strongly with the presence of the sub-part, while higher negative value correlates strongly with the absence of the sub-part. Small, near-zero values do not correlate with neither presence nor absence, and indicate that sub-part is not important for the feature.

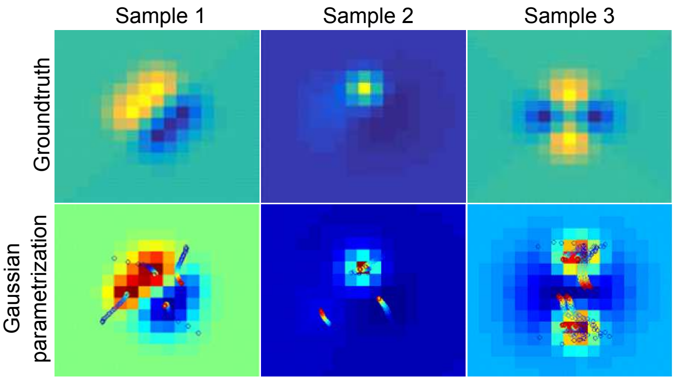

Figure 2.

The above interpretation is missing a key element of neural networks, non-linearity. Depending on non-linearity employed different interpretations can be used. With the sigmoid function a response map is mapped into values between $[0, 1]$ and can be interpreted as a probability map of a specific feature. While ReLU can be considered as a thresholding of parts. If part has response above zero, then they are passed forward and considered as being found with a specific response value, while if they are below zero, they are discarded and considered not present at all. Bias b_s can additionally modify this behavior, and move the threshold from zero to any value deemed appropriate for training data.

Compositional interpretation makes the most sense when weights are sparse. Each part feature Y_i is always composed from *all* sub-part features from the previous layer. Having high weights $\tilde{w}_k$ for all sub-parts would correspond to a part with many compositions. No meaningful semantical interpretation may be given when there are 100+ sub-features. However, a key element in making deep neural networks work properly is the regularization of weights. This is implemented as weight decay and enforces a sparse solution of weights. Such solution would have a better semantical interpretation as only a handful of sub-parts would be required by each part.

III. Evaluation

In this section we evaluate the viability of proposed method. We analyze it on a small synthetic toy case where we can control what kind of filters the network needs to learn. Furthermore, we apply it on a classification problem by evaluating it on CIFAR-10 and PaCMan datasets.

A. *Toy/synthetic case*

Gaussian parametrization introduced two new types of parameters that have to be optimized, mean and variance. In Eq. 3 and 4 we showed that gradients can be computed for both of them, and gradient descent can theoretically be applied

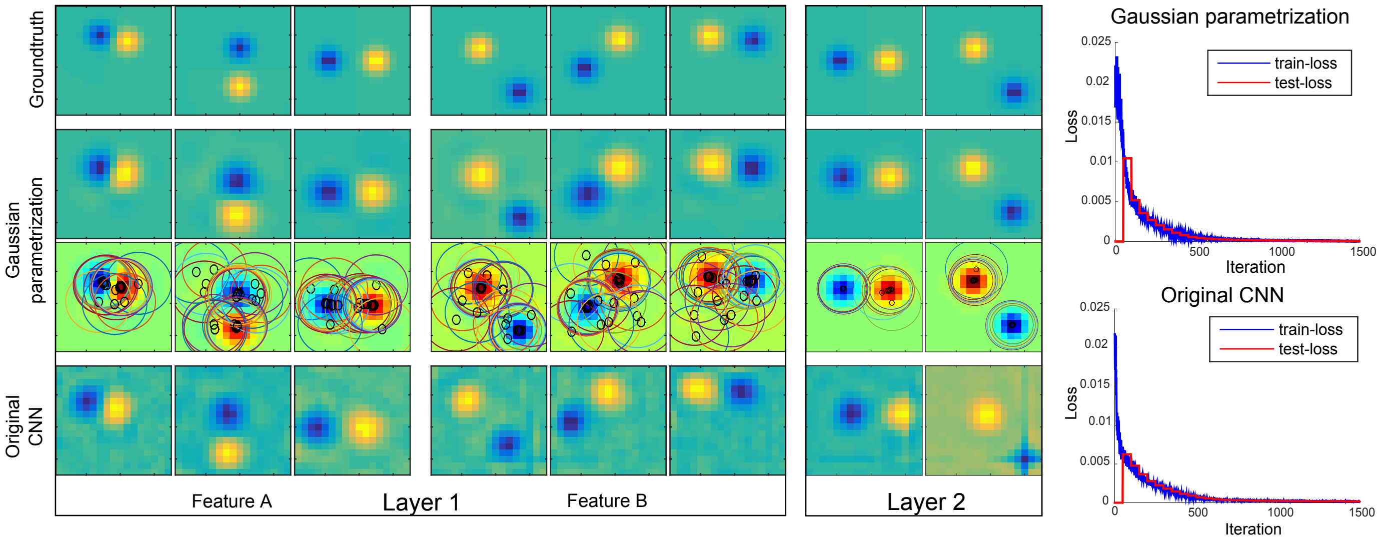

Figure 3.

to optimize them. However, equations show that gradients depend on weight, mean and variance, indicating that a proper setting for hyper-parameters, in particular learning rate, will be important to make the gradient descent work in practice. In this section we analyze optimization of our model with the gradient descent method and show which hyper-parameters are important to achieve proper convergence.

All experiments in this section are performed on synthetic models. In particular, we generate a set of filters representing the groundtruth model that our method would need to learn. We assume that the groundtruth model is composed from several Gaussian components. In classification this may not necessary hold true, but we are currently analyzing how hyper-parameters for gradient descent need to be designed to properly converge, and want to avoid adding additional noise. We use generated filters to compose groundtruth convolutional network. A forward pass on a sample image with the groundtruth network would represent the desired output, and cross-entropy loss between desired output and output of the model being optimized represents the cost function. Note, there is not a single output activation neuron, but multiple neurons each corresponding to specific input image's pixel location. This holds true only because we always use stride of one and no max-pooling between the layers. All filters are of size 17×17 giving enough space for components to move around without falling outside of the filter. We use 1000 images from CIFAR-10 dataset as training samples and track the difference between groundtruth filters and trained filters as the measurement of convergence.

Single layer: We first experimented with a single layer and a single feature, meaning, input data has only one channel, i.e., gray-scale input image, and only one first-layer feature. Three examples of groundtruth filters are shown in the first row in Figure 2. The second row shows optimized Gaussian components after 10 epochs. In all three samples small blue/red circles indicate the movement of components, with initial positions in blue and final positions in red. Clearly, optimizing position with gradient descent converges to correct solution in this simple case. Variances in all three cases seams to be accurately learned as well; however, there are noticeable deviations in negative components in the third sample.

Adding additional sub-features, i.e., using 3-channel image instead of gray-scale, shows similar results. Note, that in our case it was important to avoid RGB input image and use Lab color-space instead to get correct results. This proved to be important for regular CNNs as well, since RGB channels are highly correlated, and cause ambiguity when back-propagating the error.

It is also worth noting that gradient descent is sensitive to incorrect initializations. To achieve the above results we initialized component weights with normal distribution and evenly spreading the component's positions over the filter. We also initialized standard deviations with 0.5. Smaller values then that would collapse component to a single pixel due to discretization.

Two layers: Next, we experimented with two-layer network. We used two first-layer features composed from three sub-features, corresponding to 3-channel input data. On the second layer we used one feature composed from two sub-features. The results of optimization are shown in Figure 3. There are several important details allowing to achieve such results. It proved important to slightly over-parametrize the model by using 4×4 and 5×5 Gaussian components, for second and first layers respectfully. Components also need to be evenly spread over the filter. If this is ensured gradient descent will quickly push component weights at appropriate positions to at least correct positive or negative sign. Incorrect weights can quickly have negative effect on mean and variance gradients. However, this is not sufficient to achieve correct result. In most cases the variance converged to incorrect values and many times collapsed close to zero. Position of some components converged correctly in some cases, but stalled in other cases due to collapse of variance to zero. We were able to partially mitigate this problem by adjusting learning rates for means and variance; however, it turned out learning rate should be adjusted on a per-component basis, since adjusting learning

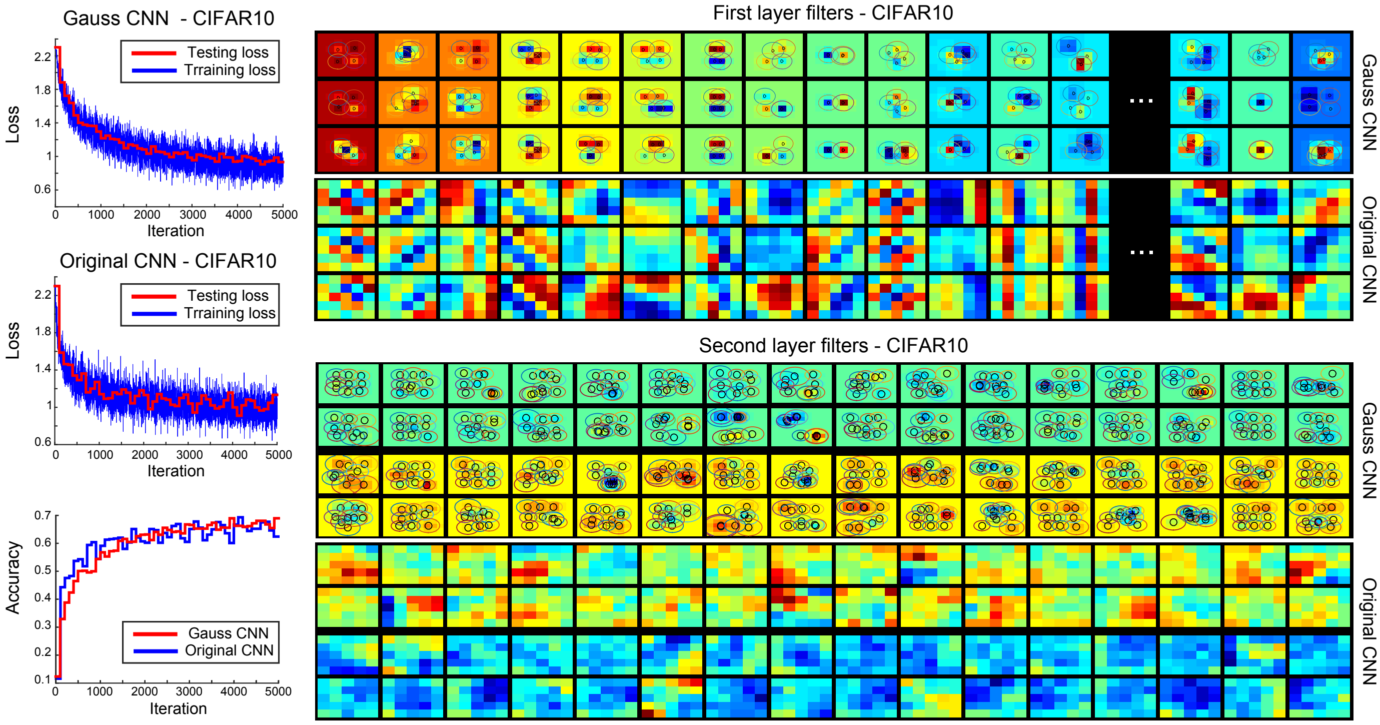

Figure 4.

rate helps in some examples, but does the opposite in others. We achieved proper per-component adjustment by utilizing AdaDelta [5] with momentum of 0.8 and no weight-decay. AdaDelta adjusts learning rate based on history of individual gradient, thus allowing to smooth-out the gradient over time and prevent them from overshooting if higher learning rate needs to be used for other gradients.

The experiments also showed that multiple components will be used to describe the same area of the filter when there are more Gaussian components available than are needed. In our case this effect was stronger in the second layer as seen in the last two samples of the third row in Figure 3. This may not necessary be the desired behavior since components are not efficiently used. We experimented with forcing Gaussian components not to overlap. This was implemented by adding a penalty function to the cost criteria to capture the intersection area between two components and penalize solutions with significant component overlaps. However, this proved not to work as expected, since penalty function prevented components from even converging to correct positions at all. We therefore did not address the issue of overlap, but it may be possible to address it later by, e.g., merging or disabling components close to each other or moving one of them to other random position. Nevertheless, this issue does not affect the overall performance, since the difference is only in compactness of the solution.

Note, that regular CNNs achieve similar results. Last row in Figure 3 shows filters learned by the regular CNN. Filters converge to visually correct representation similar to Gaussian model, but are slightly more noisy and, more importantly, their values are of several of orders of magnitude smaller than the groundtruth values. On the second layer one filter also seems to have two artifacts, a vertical and a horizontal line. This is also reflected as lines on in the first layer filters. Nevertheless, the optimization correctly converges on independent testing set that was not seen during the training indicating that solution is equivalent to the groundtruth.

			CNN with Gaussian parametrization		non-parametrized CNN
	num features	stride	filter size	num components	filter size
conv1	32	1	7 × 7	2 × 2	5 × 5
relu1	/	1	/	/	/
pool1		1	3 × 3		
conv2	32	1	9 × 9	3 × 3	5 × 5
relu2	/	1	/	/	/
pool2		2	3 × 3		
ip1	10	/	15 × 15	/	15 × 15

Table I

B. *Classification on CIFAR-10*

Next, we applied the model to the classification problem by evaluating it on CIFAR-10 dataset. Evaluation is performed with a network containing three layers. First two layers were convolutional and the third one fully-connected. We used softmax with multinominal logistic loss as cost function. Detailed network configuration is shown in Table I. Three-channel RGB image was used as input data with additional zero-mean normalization. Data was not needed to be normalized to unit variance, since between each layer we used ReLU non-linearity which is less sensitive to data variance. Note, we used slightly bigger filters size in parametrized CNN, but used less components to approximately match the number of parameters in both models. We also restricted component's positions and

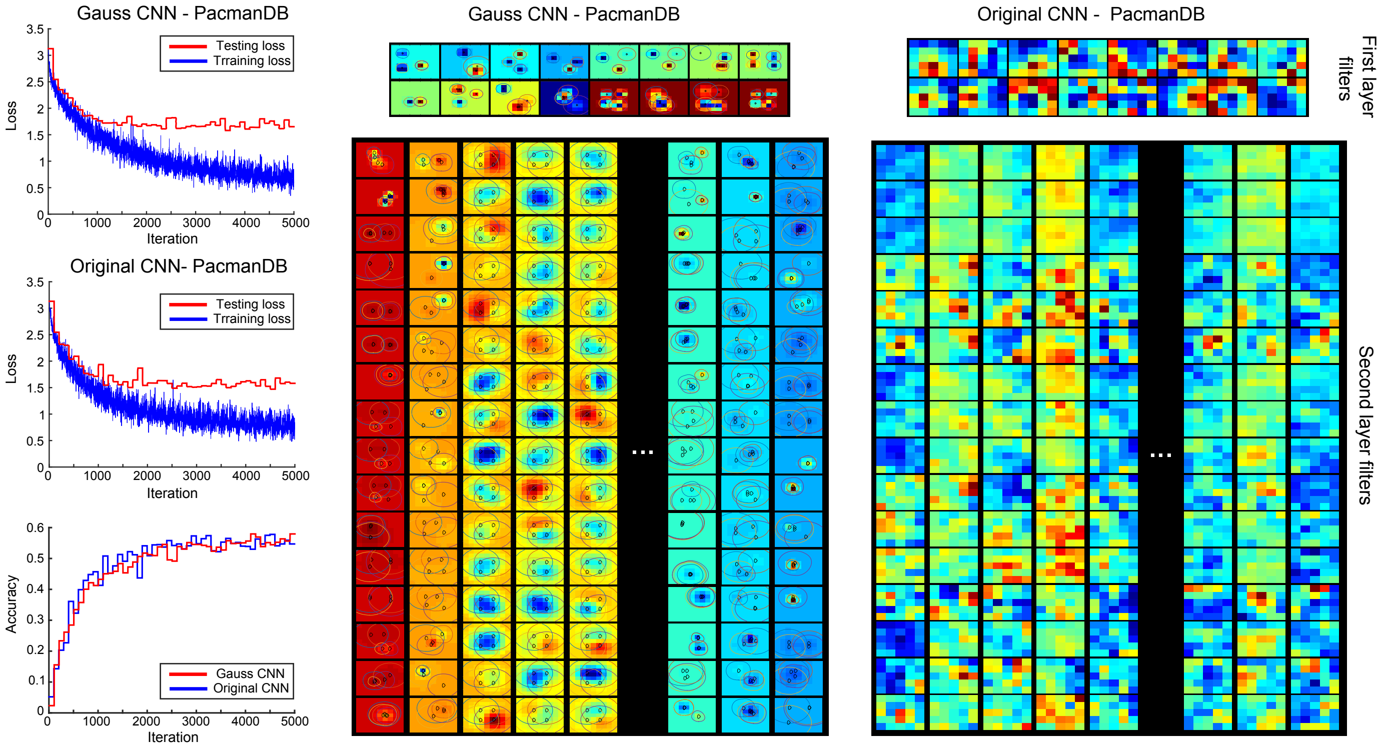

Figure 5.

standard deviation to ensure derivative of filters would not have non-zero values fall outside of the valid filter window, and also preventing them from collapsing to a single point and stalling. Positions were restricted to at least 1.5 pixels away from filter borders, and standard deviations were restricted to above 0.5.

Figure 4 shows results when training CNN with and without Gaussian parametrization after 50 epochs, i.e., after 5000 iterations. Comparing performance of both models in bottom left graph, we can see the same accuracy with both achieving slightly less then 70% on testing set. Top two graphs on the left side also reveal similar speed of learning with original CNN learning slightly faster, but they both converge to a similar loss in the end. Visualization of the first and the second layer filters on the right side in Figure 4 reveals the main difference between both models. Filters in original CNN have visible spatial structure but they are still noisy and incoherent. It would also be difficult to capture this structure without human interpenetration. On the other hand Gaussian models explicitly captured the structure as can be particularly well seen in first layer filters. Many components converged to the same location and configuration between components directly points to different edge or blob detectors. On the second layer only a smaller set of components had high weights therefore indicating that most are not relevant.

C. *Classification on PaCMan database*

We evaluated similar network on PaCMan dataset as well. Dataset contains gray-scale and depth images generated from 3D models of 23 categories of various kitchen objects, with each category containing between 15 and 30 samples. Each object is captured at dense, regular viewpoint intervals, but we use only 28 different viewpoints, summing to a total of around 40.000 images. We used only gray-scale images. Dataset was split into around 20.000 samples for training and 20.000 for testing. We ensured all viewpoint images of one sample are in the same split and each category has proportionally the same number of objects in testing and training. Input images were resized to 128×96 to fit the network into the GPU. Network configuration was slightly modified to accommodate higher resolution images. Detailed network configuration is shown in Table II.

			CNN with Gaussian parametrization		non-parametrized CNN
	num features	stride	filter size	num components	filter size
conv1	16	1	9×9	2×2	5×5
relu1	/	1	/	/	/
pool1		2	3×3		
conv2	16	1	9×9	2×2	5×5
relu2	/	1	/	/	/
pool2		2	3×3		
ip1	23	/	15×15	/	15×15

Table II

Results on PaCMan database are show in Figure 5. Performance of both models is the same as can be seen in plots on the left side. Both models achieved an accuracy of 55%. As with previous experiment the difference is in the structure of filters visualized on the right side in Figure 5. Same as before, in the first layer components often converge to same positions thus producing compact filters. Their configurations are reminiscence of edge detectors, and configuration between them can be used to find the orientation of an edge detector. On the second layer components also converge to similar positions, but not in all features. Compared to filters produced without parametrization they are more compact. In original CNN spatial position of sub-features would be difficult to determine, since filter values can be noisy and arbitrary filter values may not even represent spatial position. With Gaussian parametrization a spatial position of a sub-feature is a lot more clear and easy to be determinate as can be clearly seen in visualized filters.

IV. Conclusion

In this work we proposed to add structure into convolutional neural networks. In particular, we proposed to incorporate the spatial structure with the parametrization of filter values. We introduced a mixture of Gaussian distributions as parametrization model, and showed how to apply back-propagation and gradient descend to incorporate Gaussian parametrization into the learning process of convolutional neural networks. As we have shown, this can lead to a model which can be considered as regular convolutional network, as well, as a compositional hierarchical model, such as [6], at the same time. We believe this is an important property, since it leads to a merging of two fields, compositional hierarchy fields, and convolutional neural network fields. It opens the doors to combine positive properties from both models, such as combining a strong discriminative learning from neural networks with a compositional interpretation and generative learning from compositional hierarchies. We showed on two classification databases that this approach can achieve the same performance as non-parametrized CNNs, but at the same time enables compositional interpretation of features. Many features learned with Gaussian parametrization had more compact filter values, and their interpretation as compositions was more plausible since only a subset of components remained active as well.

We plan to continue this work by further exploiting compositional interpretation and explore several venues for adding information from compositional hierarchies into the learning process. It would be possible to use a generative pre-training from compositional hierarchies, but it would also be possible to better steer the optimization of parameters using the statistics of compositions. It is also worth considering how other properties of compositional hierarchies, such as top-down reasoning or efficient indexing, can be utilized to improve convolutional networks.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances In Neural Information Processing Systems 25*, 2012, pp. 1097–1105.

[2] M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” in *European Conference on Computer Vision*, nov 2013, pp. 818–833. [Online]. Available: http://arxiv.org/abs/1311.2901

[3] A. Mahendran and A. Vedaldi, “Understanding Deep Image Representations by Inverting Them,” in *Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[4] Y. LeCun, “Generalization and network design strategies,” *Connectionism in perspective*, pp. 143–155, 1989.

[5] M. D. Zeiler, “ADADELTA: An Adaptive Learning Rate Method,” 2012. [Online]. Available: http://arxiv.org/abs/1212.5701

[6] S. Fidler and A. Leonardis, “Towards Scalable Representations of Object Categories: Learning a Hierarchy of Parts,” in *Computer Vision and Pattern Recognition*. IEEE Computer Society, 2007, pp. 1–8.