F-Transform and Convolutional NN: Cross-Fertilization and Step Forward

1st Vojtech Molek
Institute for Research and Applications of Fuzzy Modeling
Ostrava Univesity
Ostrava, Czech Republic
vojtech.molek@osu.cz

2nd Irina Perfilieva
Institute for Research and Applications of Fuzzy Modeling
Ostrava Univesity
Ostrava, Czech Republic
irina.perfilieva@osu.cz

***Abstract*—We propose to assign the F-transform kernels to the CNN weights and compare them with commonly used initialization. By this, we develop a new initialization mechanism where the F-transform convolution kernels are used in the convolutional layers. Based on a series of experiments, we demonstrate the suitability of the F-transform-based deep neural network in the domain of image processing with the focus on classification. Moreover, we support our insight by revealing the similarity between the F-transform and first-layer kernels in certain deep neural networks.**

***Index Terms*—F-transform, convolutional neural networks, pretraining**

I. Introduction

Our goal is two-fold: to extend the deep learning (DL) methodology of convolution neural networks (CNN) by reasonable (F-transform-based) initialization and to contribute to the F-transform technology by learning mechanisms. The first goal is within data-driven machinery, while the second one is an example of a model-driven tool. In short, our idea is to assign the F-transform kernels to the CNN weights and evaluate the accuracy on the benchmark dataset. This motivation is supported by theoretically proved results about the approximation abilities of the F-transform technique. In the language of CNNs, these results assure that the kernels associated with the F-transform can extract a sufficient amount of features for satisfactory reconstruction. On the other hand, the extracted by the F-transform features (called *components*) can be further tailored to a given dataset using the procedure of learning. In this contribution, we show how this cross-fertilization helps to improve both techniques.

In detail, we are focused on a smart and conscious initialization of convolutional kernels in convolutional layers where neurons have restricted receptive fields. Our approach can be named as a “preprocessing of methodology”.

The preprocessing is realized in the convolution layers together with feature extraction. In subsequent fully-connected layer(s), the extracted features are used for classification, recognition, etc. Therefore, the initial objects are modeled by the extracted features, so that the former ones can be approximately reconstructed from the latter.

In our contribution, we discuss a pretraining method for CNN. It is common practice to use CNN trained on large datasets and do final *fine tuning* on a domain-specific dataset. The size of a domain-specific dataset is usually smaller, so the fine-tuning benefits dramatically from the pretrained model [1].

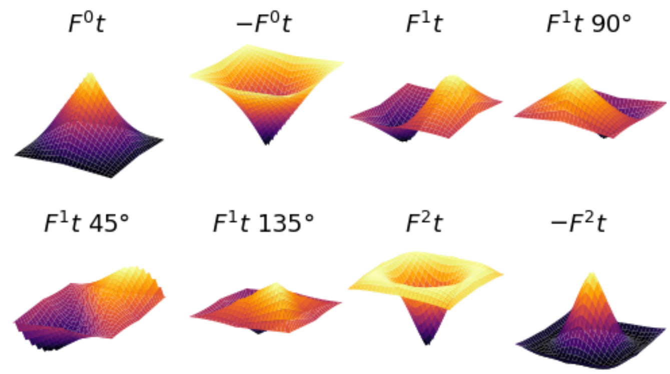

Fig. 1. Visualisation of convolutional kernels used for extracting F-transform components of different degrees.

Using a pretrained model means to use the knowledge extracted from data, as CNNs are data-driven models, in a large dataset, and to apply this knowledge to solve a related problem on a smaller dataset. Pretraining models on a large amount of data come with significant computation requirements, so the possibility of sharing models does not only save time but resources as well.

In our contribution, we propose a different philosophy on pretraining models. Rather than expensively learning weights from data, we initialize a subset of the parameters with *handcrafted* values. Our handcrafted values are convolutional kernels originating from the theory of fuzzy modeling and the analysis of CNN functioning. In particular, we use the F-transform theory that has been successfully applied to image processing tasks [2]–[4]. The F-transform represents image data using the components, which are numerical values, expressing different geometrical properties. These properties are of local influence, as each component is computed over a small neighborhood. By increasing degree n of F^n-transform, each component gains additional expressivity: if $n = 0$, then a weighted average intensity is extracted, if $n = 1$, then an average value of a gradient is obtained, and so forth. The inverse F-transform reconstructs the original data from the computed components with a requested precision. High precision means that components carry a high amount of information about original data. We argue that components can be used as an intermediate representation of data within CNNs.

In the following sections, we will recall the fundamentals of F-transform theory and practical results. The results were obtained by using compact CNN architecture for image classification task on CIFAR-10 [5] dataset.

II. The F-transform of a Higher Degree (F^m-transform)

In this section, we recall the main facts (see Ref. [6] for more details) about the higher degree F-transform and specifically F^2-transform - the technique, which will be used in the CNN with the F-transform kernels (FTNet) proposed below.

We aim at expressing the F-transform in the form of a convolution of two functions: a given (object) function and a function that generates a fuzzy partition. The latter serves as a kernel function. We will start by reminding the basic definitions regarding the F-transform. We will focus on the discrete F-transform only.

A. Discrete F-transform

Let us consider the discrete F-transform [7]. We choose an interval $[a, b]$ as a universe, and assume that a function f is given at points $p_0, \dots, p_{l-1} \in [a, b]$.

Below, we recall the definition of a fuzzy partition. Let $a = x_0 < \dots < x_n = b,\ n \geq 3$ be fixed nodes within $[a, b]$. Fuzzy sets $A_1, \dots, A_{n-1}$ identified with their membership functions $A_1, \dots, A_{n-1}$, defined on $[a, b]$, establish a *fuzzy partition* of $[a, b]$ if they fulfill the following conditions for $k = 1, \dots, n-1$:

1) $A_k : [a, b] \to [0, 1],\ A_k(x_k) = 1$;
2) $A_k(x) = 0$ if $x \notin (x_{k-1}, x_{k+1}),\ k = 1, \dots, n-1$;
3) $A_k(x)$ is continuous;
4) $A_k(x)$ strictly increase on $[x_{k-1}, x_k]$, $k = 1, \dots, n-1$; and strictly decrease on $[x_k, x_{k+1}],\ k = 1, \dots, n-1$;
5) $\sum_{k=1}^{n} A_k(x) = 1,\ x \in [x_1, x_{n-1}]$.

$A_1, \dots, A_{n-1}$ are called *basic functions*.

We say that the fuzzy partition given by $A_1, \dots, A_{n-1}$, is an *h-uniform fuzzy partition* if the nodes $x_k = a + hk,\ k = 0, \dots, n$, are equidistant, $h = (b - a)/n$ and two additional properties are met:

6) $A_k(x_k - x) = A_k(x_k + x),\ x \in [0, h],\ k = 1, \dots, n-1$;
7) $A_k(x) = A_{k-1}(x - h),\ k = 2, \dots, n-1,\ x \in [x_{k-1}, x_{k+1}]$.

Assume that fuzzy sets $A_1, \dots, A_{n-1}$ establish a fuzzy partition of $[a, b]$ and $f : P \to \mathbb{R}$ is a discrete real valued function defined on the set $P = \{p_0, \dots, p_{l-1}\}$ where $P \subseteq [a, b]$ and $l > n$. The following vector of real numbers $\mathbf{F}_n[f] = [F_1, \dots, F_{n-1}]$ is the *(direct) discrete F-transform* of f w.r.t. $A_1, \dots, A_{n-1}$ where the $k-$th component F_k is defined by

$$F_k = \frac{\sum_{j=0}^{l-1} A_k(p_j) f(p_j)}{\sum_{j=0}^{l-1} A_k(p_j)},\ k = 1, \dots, n-1. \quad (1)$$

A semantic meaning of an F-transform component is in giving the *best weighted average value* of function f over the area covered by the corresponding basic function. More details are given in the following proposition [7].

Lemma 1: Let function f be given at nodes $p_1, \dots, p_l \in [a, b]$ and $A_1, \dots, A_n$ be basic functions which form a fuzzy partition of $[a, b]$. Then the k-th component of the discrete F-transform gives minimum of the function

$$\Phi(y) = \sum_{j=1}^{l} (f(p_j) - y)^2 A_k(p_j). \quad (2)$$

By using an inversion formula we can approximately reconstruct function f from the vector of components of its direct discrete F-transform. We define [7] the *inverse discrete F-transform* as

$$f_{F,n}(p_j) = \sum_{k=1}^{n-1} F_k A_k(p_j),\ j = 0, \dots, l-1.$$

Moreover, the following Theorem 1 says that the inverse discrete F-transform $f_{F,n}$ can approximate the original function f at common nodes with an arbitrary precision (proved in [7]).

Theorem 1: Let a function f be given at nodes $p_0, \dots, p_{l-1}$ constituting the set $P \subseteq [a, b]$. Then, for any $\varepsilon > 0$, there exist n_ε and a fuzzy partition $A_1, \dots, A_{n_\varepsilon}$ of $[a, b]$ such that P is sufficiently dense with respect to $A_1, \dots, A_{n_\varepsilon}$ and for all $p \in \{p_0, \dots, p_{l-1}\}$

$$|f(p) - f_{F,n_\varepsilon}(p)| < \varepsilon$$

holds true.

B. F-Transform as Convolution

Let us assume that the interval $[a, b]$ is h-uniformly partitioned by fuzzy sets $A_1, \dots, A_{n-1}$, f is a discrete function, and the F-transform of a discrete function f is given by $\mathbf{F}_n[f]$ with components obtained by(1).

It is easy to see that if the fuzzy partition $A_1, \dots, A_{n-1}$ of $[a, b]$ is h-uniform, then there exists an even function

$$A : [-h, h] \to [0, 1]$$

such that for all $k = 1, \dots, n-1$,

$$A_k(x) = A(x - x_k) = A(x_k - x),\ x \in [x_{k-1}, x_{k+1}].$$

We call A a *generating function* of an h-uniform fuzzy partition.

Let us assume that points $p_0, \dots, p_{l-1}$ are equidistant in the interval $[a, b]$ and moreover $p_j = a + jh/m;\ j = 0, \dots, l-1$, where m and l are connected by the following equality: $l = nm + 1$. Thus chosen points $p_0, \dots, p_{l-1}$ assure that the nodes $x_0, \dots, x_n$ are among them, i.e. for each $k = 0, \dots, n$, there exists j such that $x_k = p_j$. Moreover, the following *Lemma 1* holds true.

Lemma 2: Let $A_1, \dots, A_{n-1}$ establish an h-uniform fuzzy partition of $[a, b]$ and points $p_0, \dots, p_{l-1}$ from $[a, b]$ are chosen as above. Then there exists a constant $c > 0$ such that for all $k = 1, \dots, n-1$,

$$\sum_{j=0}^{l-1} A_k(p_j) = c. \tag{3}$$

Proof 1: In order to prove (3), it is sufficient to show that for all $k = 1, \dots, n-2$,

$$\sum_{j=0}^{l-1} A_{k+1}(p_j) = \sum_{j=0}^{l-1} A_k(p_j). \tag{4}$$

Indeed, the uniformity of our partition and the fact that

$$A_{k+1}(p_{j+m}) = A_k(p_j), j = 0, \dots, l-1-m,$$

leads to

$$\sum_{j=0}^{l-1} A_{k+1}(p_j) = A_{k+1}(p_{km}) + \dots + A_{k+1}(p_{(k+2)m}) =$$

$$A_k(p_{(k-1)m}) + \dots + A_k(p_{(k+1)m}) = \sum_{j=0}^{l-1} A_k(p_j),$$

$$k = 1, \dots, n-2.$$

Remark 1: Let us remark that (3) is not the generalized Ruspini condition, because the sum is taken over points $p_0, \dots, p_{l-1}$. Actually, the sum in (3) is taken over those points that are covered by a single basic function $A_k, k = 1, \dots, n-1$.

By (3), the expression (1) can be rewritten as follows:

$$F_k = \frac{\sum_{j=0}^{l-1} A(x_k - p_j) f(p_j)}{c}; \; k = 1, \dots, n-1. \tag{5}$$

Let us consider F_k as a value of a discrete function F, defined on the set $\mathbf{Z}_{n-1} = \{1, \dots, n-1\}$ with values from $\mathbb{R}$ such that $F : \mathbf{Z}_{n-1} \to \mathbb{R}$ and $F(k) = F_k$. We will use (5) for an analytic extension of F from $\mathbf{Z}_{n-1}$ to $\mathbf{Z}_l = \{0, 1, \dots, l-1\}$, so that

$$F(t) = \frac{\sum_{j=0}^{l-1} A(p_t - p_j) f(p_j)}{c}; \; t = 0, \dots, l-1. \tag{6}$$

Similarly, we can assume that functions A and f are defined on the set $\mathbf{Z}_l$ and rewrite (6) into

$$F(t) = \frac{\sum_{j=0}^{l-1} A(t - j) f(j)}{c}; \; t = 0, \dots, l-1. \tag{7}$$

Finally, we will normalize values of A dividing them by c and keep the same denotation A for the normalized function. Then without loss of generality, we will continue working with the below given expression for F:

$$F(t) = \sum_{j=0}^{l-1} A(t - j) f(j); \; t = 0, \dots, l-1. \tag{8}$$

Analyzing (8), we see that the function $F : \mathbf{Z}_l \to \mathbb{R}$ is a convolution of two discrete functions: f (referred above as an object-function) and A (referred above as a kernel-function). Let us remark that function F contains the F-transform components F_k, $k = 1, \dots, n-1$, among its values. In order to extract the F-transform components F_k, we shall select the step value m, so that the convolution (8) is computed for $t = 0, m, 2m, \dots$. The value m determines a so called *stride*.

C. F^m-transform

In this section, we define the F^m-transform, $m \geq 0$, of a function f with polynomial components of degree m. For this purpose, we use the integral form of the F-transform. Let us fix $[a, b]$ and its fuzzy partition $A_1, \dots, A_n$, $n \geq 2$.

Definition 1 ([6]): Let $f : [a, b] \to \mathbb{R}$ be a function from $L_2(A_1, \dots, A_n)$, and let $m \geq 0$ be a fixed integer. Let F_k^m be the k-th orthogonal projection of $f|_{[x_{k-1}, x_{k+1}]}$ on $L_2^m(A_k)$, $k = 1, \dots, n$. We say that the n-tuple $(F_1^m, \dots, F_n^m)$ is an F^m-transform of f with respect to $A_1, \dots, A_n$, or formally,

$$F^m[f] = (F_1^m, \dots, F_n^m).$$

F_k^m is called the k^{th} F^m-transform component of f.

Explicitly, each k^{th} component is represented by the m^{th} degree polynomial

$$F_k^m = c_{k,0} P_k^0 + c_{k,1} P_k^1 + \dots + c_{k,m} P_k^m, \tag{9}$$

where

$$c_{k,i} = \frac{\langle f, P_k^i \rangle_k}{\langle P_k^i, P_k^i \rangle_k} = \frac{\int_a^b f(x) P_k^i(x) A_k(x) dx}{\int_a^b P_k^i(x) P_k^i(x) A_k(x) dx}, \; i = 0, \dots, m.$$

Definition 2: Let $\mathrm{F}^m[f] = (F_1^m, \dots, F_n^m)$ be the direct F^m-transform of f with respect to $A_1, \dots, A_n$. Then the function

$$\hat{f}_n^m(x) = \sum_{k=1}^{n} F_k^m A_k(x), \quad x \in [a, b], \tag{10}$$

is called the *inverse F^m-transform* of f.

The following theorem proved in [6] estimates the quality of approximation by the inverse F^m-transform in a normed space L_1.

Theorem 2: Let $A_1, \dots, A_n$ be an h-uniform fuzzy partition of $[a, b]$. Moreover, let functions f and A_k, $k = 1, \dots, n$ be four times continuously differentiable on $[a, b]$, and let $\hat{f}_n^m$ be the inverse F^m-transform of f, where $m \geq 1$. Then

$$\|f(x) - \hat{f}_n^m(x)\|_{L_1} \leq O(h^2),$$

where L_1 is the Lebesgue space on $[a + h, b - h]$.

D. F^2-transform in the Convolutional Form

Let us fix $[a, b]$ and its h-uniform fuzzy partition $A_1, \dots, A_n$, $n \geq 2$, generated from $A : [-1, 1] \to [0, 1]$ and its h-rescaled version A_h, so that $A_k(x) = A(\frac{x - x_k}{h}) = A_h(x - x_k)$, $x \in [x_k - h, x_k + h]$, and $x_k = a + kh$. The

F^2-transform of a function f from $L_2(A_1, \ldots, A_n)$ has the following representation

$$F^2[f] = (c_{1,0}P_1^0 + c_{1,1}P_1^1 + c_{1,2}P_1^2, \ldots, c_{n,0}P_n^0 + c_{n,1}P_n^1 + c_{n,2}P_n^2), \quad (11)$$

where for all $k = 1, \ldots, n$,

$$P_k^0(x) = 1,\ P_k^1(x) = x - x_k,\ P_k^2(x) = (x - x_k)^2 - I_2, \quad (12)$$

where $I_2 = h^2 \int_{-1}^{1} x^2 A(x)dx$, and coefficients are as follows:

$$c_{k,0} = \frac{\int_{-\infty}^{\infty} f(x)A_h(x - x_k)dx}{\int_{-\infty}^{\infty} A_h(x - x_k)dx}, \quad (13)$$

$$c_{k,1} = \frac{\int_{-\infty}^{\infty} f(x)(x - x_k)A_h(x - x_k)dx}{\int_{-\infty}^{\infty} (x - x_k)^2 A_h(x - x_k)dx}, \quad (14)$$

$$c_{k,2} = \frac{\int_{-\infty}^{\infty} f(x)((x - x_k)^2 - I_2)A_h(x - x_k)dx}{\int_{-\infty}^{\infty} ((x - x_k)^2 - I_2)^2 A_h(x - x_k)dx}. \quad (15)$$

In ([6], [8]), it has been proved that

$$c_{k,0} \approx f(x_k),\ c_{k,1} \approx f'(x_k),\ c_{k,2} \approx f''(x_k), \quad (16)$$

where $\approx$ is meant up to $O(h^2)$.

Without going into technical details, we rewrite (13) - (15) into the following discrete representations

$$c_{k,0} = \sum_{j=1}^{l} f(j)g_0(ks - j)$$

$$c_{k,1} = \sum_{j=1}^{l} f(j)g_1(ks - j)$$

$$c_{k,2} = \sum_{j=1}^{l} f(j)g_2(ks - j) \quad (17)$$

where $k = 1, \ldots, n$, $n = \lfloor \frac{l}{s} \rfloor$, s is a stride and g_0, g_1, g_2 are normalized functions that correspond to generating functions A_h, (xA_h) and $((x^2 - I_2)A_h)$. It is easy to see that if $s = 1$, then coefficients $c_{k,0}, c_{k,1}, c_{k,2}$ are results of the corresponding discrete convolutions $f \star g_0$, $f \star g_1$, $f \star g_2$. Thus, we can rewrite the representation of F^2 in (11) in the following vector form:

$$F^2[f] = ((f \star_s g_0)^T \mathbf{P}^0 + (f \star_s g_1)^T \mathbf{P}^1 + (f \star_s g_2)^T \mathbf{P}^2), \quad (18)$$

where $\mathbf{P}^0$, $\mathbf{P}^1$, $\mathbf{P}^2$ are vectors of polynomials with components given in (12), and $\star_s$ means that the convolution is performed with the stride s, $s \geq 1$.

III. FTNet – Embedding Knowledge into CNN

Ref. [9] proposed incorporation of F-transform into convolutional neural network – FTNet. Here, we recall all essential details.

In Ref. [9], we modified the baseline network by replacing convolutional kernels in the first and second convolutional layers with the F-transform kernels according to (17) adapted to functions of two variables. To properly work with colorful image data, we expand our kernels to 3D by processing each color channel separately. Moreover we perform cluster analysis on InceptionResNetV2 [10] first layer weights and use 6 cluster centroids[1] (Fig. 2) as additional kernels. In total, we have a set of 14 kernels for the first convolutional layer initialization. Other convolutional layers are initialized with F-transform kernels as well. However, since the number of channels changes in dependence on a number of previous layer filters, we randomly sample from the set of our 8 F-transform kernels. Cluster centroids are not used beyond first convolutional kernels. Note that F-transform kernels are parametrized by their width, height, rotation, and sign (+/-). This parametrization allows us to freely enlarge the set of F-transform kernels as well as generate kernels for different filter sizes.

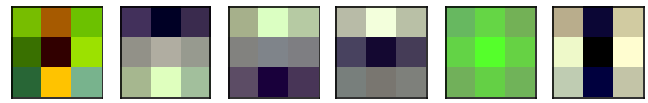

Fig. 2. 6 centroids extracted from InceptionResNetV2.

A. FTNet architecture

FTNet architecture contains repeating basic block (Fig. 3). We follow the rule of thumb, and with each block repetition, as the spatial resolution decreases, we increase the number of convolutional layer filters. The overall architecture is shown in Tab. I. We use batch normalisation and L_2 weight decay with strength 1e-4 to reduce the overfitting. Throughout our experiments, the architecture does not change.

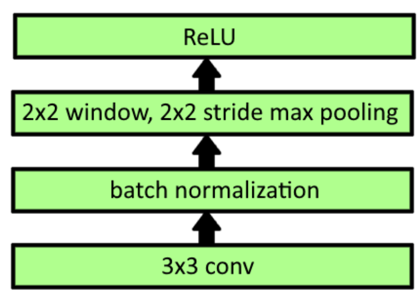

Fig. 3. Building block used in FTNet.

TABLE I
FTNet ARCHITECTURE

Layers	Parameters		
	Weights size	*Output size*	*Activation*
block_1	$3 \times 3 \times 3 \times 14$	$16 \times 16 \times 14$	ReLU
block_2	$3 \times 3 \times 14 \times 28$	$8 \times 8 \times 28$	ReLU
block_3	$3 \times 3 \times 28 \times 56$	$4 \times 4 \times 56$	ReLU
dense	896×256	256	ReLU
dense	256×10	10	softmax

[1] We have used dendrogram to estimate the number of clusters.

B. Training

We train FTNet on 80% of the training part of CIFAR-10 and use the remaining 20% for validation purposes. The training data are randomly augmented[2] during the training to further prevent overfitting. We use SGD optimizer with momentum m=0.9, variable learning rate with decay γ=1e-2. Two conditions are used during training: one to stop training and one to decrease learning rate, both depended on the decreasing value of validation loss. Since hyperparameters fundamentally influence training procedure, in further section with the results, we report accuracies for different optimizers, learning rates, and batch sizes, 500 being default one.

C. Weights initialization

FTNet uses two types of initialization: one for the first convolutional layer and second for other convolutional layers. The difference is in the input of layers; while the first convolutional layer input is batch of 3 channel images, all successive convolutional layers have batches of n feature maps. n depends on the previous layer parameters. We initialize the first convolutional layer with the set of previously described 6 centroids and set of F-transform kernels (Fig. 1).

Other convolutional layers weights are uniformly sampled from the set of F-transform kernels $\mathbf{A}$ such that $w^k_{i,j} = A_\ell$, where $\ell \sim \mathcal{U}(0, |\mathbf{A}|)$ and $w^k_{i,j}$ is j-th filter of k-th convolutinal layer convolving i-th input feature map. With such a initialization, each successive convolutional layer perform F-transform on components extracted by previous convolutional layer, creating unique descriptors.

Lastly, we follow the idea of Xavier initialization [11] and rescale w such that $w \in [-\sqrt{\frac{6}{X*Y*I+J}}, \sqrt{\frac{6}{X*Y*I+J}}]$ where X, Y are width and height of F-transform kernels, I is number of input feature maps and J is number of filters. Note that rescaling the kernels is crucial for good convergence, and without rescaling, FTNet underperforms.

D. Performance of FTNet

We compare two different initializations: initialization with F-transform kernels, described in Sec. III-C and Xevier initialization. In both cases, we do not use biases. The test accuracy for different hyperparameters is shown in Tab. II. Fig. 4 contains process of training in terms of training/validation loss and accuracy.

IV. Similarity between F-transform kernels and other kernels

We performed a clustering analysis of 6 CNN first layer weight vectors to confirm our hypothesis that networks learn certain groups of filters known in image processing. Since F-transform kernels shares similarities with said known image processing kernels, that would further support F-transform suitability for weights initialization. Fig. 5 shows the results of clustering; 6 medoids per network. We can see that medoids are roughly from the classes of *Gaussian, edge detection, and texture detection* kernels. F-transform kernels can be assigned to these respective classes as well.

[2]Rotation, width/height shift, shear, zoom, and horizontal flips.

TABLE II
FTNet and Xavier initialization test accuracies.

Hyperparameters	F-transform (%)	Xavier (%)
SGD(lr=1e-2, γ=1e-2, m=0.9) L_2(1e-2), batch=500	66.32%	60.03%
SGD(lr=1e-1, γ=1e-2, m=0.9) L_2(1e-2), batch=500	71.52%	71.25%
SGD(lr=1e-1, γ=1e-2, , m=0.9) L_2(1e-2), batch=100	67.30%	66.99%
SGD(lr=1, γ=1e-2, m=0.9) L_2(1e-2), batch=500	66.28%	66.61%
Adam(lr=1e-1, γ=1e-2) L_2(1e-2), batch=500	77.03%	78.06%
Adam(lr=1e-2, γ=1e-2) L_2(1e-2), batch=500	75.73%	75.08%

V. Conclusion and the future work

We have introduced new pretraining/initialization method based on the fuzzy modeling technique – F-transform. F-transform proved themselves being suitable for initialization of convolutinal layer filters. Their parametrization makes them versatile alternative to existing methods, mostly founded on statistical tools. We have shown accuracies of F-transform and Xavier initialization on the CIFAR-10 test set. F-transform almost exclusively achieved higher accuracy and more stable learning process.

Lastly, we provided evidence of a certain grouping of convolutional filters in the first layers of known CNNs. F-transform kernels belong to these groups as well.

Future work includes performing experiments with ”noisy” F-transform kernels as some of the kernels includes 0 values that might cause slower or inconsistent training. Another direction is to use F-transform initialization on some of the state-of-the-art networks such as ResNet-18 and other optimizers too.

Acknowledgment

The work was supported from ERDF/ESF ”Centre for the development of Artificial Intelligence Methods for the Automotive Industry of the region” (No. CZ.02.1.01/0.0/0.0/17_049/0008414)

References

[1] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, “Large scale learning of general visual representations for transfer,” *arXiv preprint arXiv:1912.11370*, 2019.

[2] P. Hurtik and S. Tomasiello, “A review on the application of fuzzy transform in data and image compression,” *Soft Computing*, pp. 1–13, 2019.

[3] I. Perfilieva, P. Hodáková, and P. Hurtík, “Differentiation by the f-transform and application to edge detection,” *Fuzzy Sets and Systems*, vol. 288, pp. 96–114, 2016.

[4] V. Pavel and P. Irina, “Interpolation techniques versus f-transform in application to image reconstruction,” in *Fuzzy Systems (FUZZ-IEEE), 2014 IEEE International Conference on*. IEEE, 2014, pp. 533–539.

[5] A. Krizhevsky, G. Hinton *et al.*, “Learning multiple layers of features from tiny images,” Citeseer, Tech. Rep., 2009.

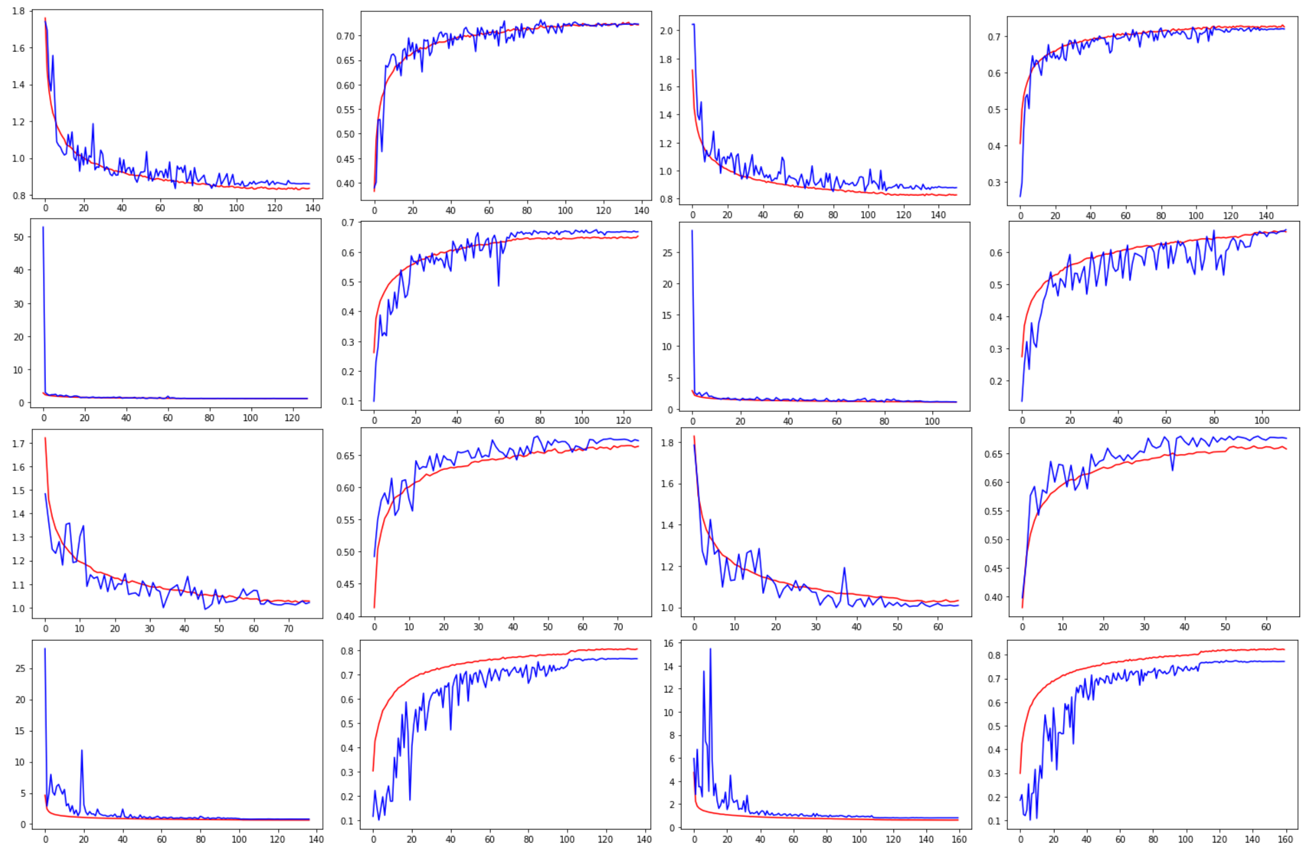

Fig. 4. Process of training (red curve) and validating (blue curve) of Tab. II. First column is training and validation loss of F-transform kernels, second column is F-transform training and validation accuracy. Third and fourth column is training and validation loss and accuracy of Xavier initialization. First to fourth row correspond to second to fifth row of Tab. II.

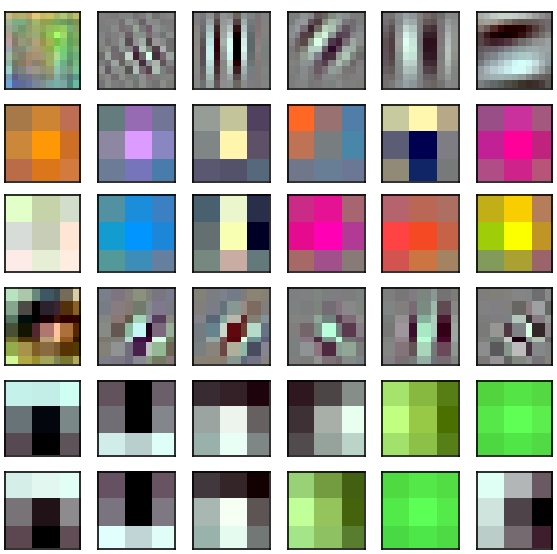

Fig. 5. Each row contains 6 cluster medoids of one of the selected CNN: 1st AlexNet [12], 2nd InceptionV3 [13], 3rd MobileNet [14], 4th ResNet [15], 5th VGG 16 [16], 6th VGG 19 [16].

[6] I. Perfilieva, M. Danková, and B. Bede, "Towards f-transform of a higher degree." in *IFSA/EUSFLAT Conf.*, 2009, pp. 585–588.

[7] I. Perfilieva, "Fuzzy transforms: Theory and applications," *Fuzzy sets and systems*, vol. 157, no. 8, pp. 993–1023, 2006.

[8] I. Perfilieva and V. Kreinovich, "Fuzzy transforms of higher order approximate derivatives: A theorem," *Fuzzy Sets and Systems*, vol. 180, no. 1, pp. 55–68, 2011.

[9] V. Molek and I. Perfilieva, "Convolutional neural networks with the f-transform kernels," in *International Work-Conference on Artificial Neural Networks*. Springer, 2017, pp. 396–407.

[10] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[11] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[14] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.