UNIVERSITY OF OSTRAVA

DOCTORAL THESIS

2021 MGR. VOJTĚCH MOLEK

UNIVERSITY OF OSTRAVA
FACULTY OF SCIENCE
DEPARTMENT OF INFORMATICS AND COMPUTERS

Fusion of Methodologies: F-transform with CNN

Doctoral thesis

Author: Mgr. Vojtěch Molek
Supervisor: Prof. Irina Perfilieva, CSc.

2021

OSTRAVSKÁ UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA
KATEDRA INFORMATIKY A POČÍTAČŮ

Fúze metodik: F-transformace a CNN

Disertační práce

Autor práce: Mgr. Vojtěch Molek
Vedoucí práce: Prof. Irina Perfilieva, CSc.

2021

OSTRAVSKÁ UNIVERZITA

Přírodovědecká fakulta

Akademický rok: 2020/2021

ZADÁNÍ DISERTAČNÍ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Mgr. Vojtěch MOLEK**
Osobní číslo:	**P15150**
Studijní program:	**P1802 Aplikovaná informatika**
Studijní obor:	**Aplikovaná informatika**
Téma práce:	**Fúze metodik: F-transformace a CNN**
Téma práce anglicky:	**Fusion of Methodologies: F-transform with CNN**
Zadávající katedra:	**Katedra informatiky a počítačů**

Zásady pro vypracování

Navrhnout a zdůvodnit nový způsob inicializace neuronových sítí hloubkové struktury. Využít teorii fuzzy (F-) transformace vyššího stupně a její schopnost aproximovat spojitou funkci s libovolnou přesností a metodu výpočtu jejích komponent pomocí operace konvoluce. Potvrdit hypotézu stability vah neuronových sítí inicializovaných pomocí (F-) transformačních jader se stupni 0,1 a 2. K tomuto účelu použít neuronové sítě různých architektur. Popsat výsledky z pohledu výkonnosti, přesnosti, vysvětlitelnosti a dopadu na životní prostředí.

Rozsah pracovní zprávy:
Rozsah grafických prací:
Forma zpracování disertační práce: **tištěná**

Seznam doporučené literatury:

1. Irina Perfilieva, Martina Danková, Barnabas Bede, Towards a higher degree F-transform, Fuzzy Sets and Systems 180 (2011) 3-19
2. I. Perfilieva, Fuzzy transforms: theory and applications, Fuzzy Sets and Systems 157 (2006) 993-1023.
3. Y. LeCun et al. „Gradient-based learning applied to document recognition". In: Proceedings of the IEEE 86.11 (1998), pp. 2278-2324. (LeNet a MNIST)
4. P. Vlašánek and I. Perlieva. „The F-transform in Terms of Image Processing Tools". In: Journal of Fuzzy Set Valued Analysis 2016 (2016), pp. 54-62. (konvoluční kernely F-transformace)
5. K. He et al. „Deep residual learning for image recognition". In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, pp. 770-778. (ResNet publikace)

Vedoucí disertační práce: **prof. Irina Perfiljeva, CSc.**
Centrum excelence IT4Innovations

Datum zadání disertační práce: 1. září 2020
Termín odevzdání disertační práce: 31. srpna 2021

prof. Irina Perfiljeva, CSc.
vedoucí doktorské práce

doc. RNDr. PaedDr. Hashim Habiballa, PhD., Ph.D.
vedoucí katedry

V Ostravě dne 26. ledna 2021

Abstract

This thesis describes the fusion of convolutional neural networks with the theory of F-transform. The key part is the implementation of convolutional layer initialization with F-transform in the form of convolutional kernels. The emphasis is put on the network explainability, accuracy, and speed. Such an initialization was experimentally compared with others, standardly used initialization methods using multiple, state-of-the-art neural networks.

Key Words: Convolutional neural networks, F-transform, initialization, classification, explainability

Abstrakt

Tato práce popisuje spojení konvolučních neuronových sítí s teorii F-transformace. Stěžejní částí je implementace inicializace konvolučních vrstev s využitím F-transformace ve formě konvolučních jader. Důraz je kladen na vysvětlitelnost chování, přesnost a rychlost sítě. Takováto inicializace byla experimentálně ověřena s jinými, běžně používanými inicializačními metodami s využitím několika moderních neuronových sítí.

Klíčová slova: Konvoluční neuronové sítě, F-transformace, inicializace, klasifikace, vysvětlitelnost

Já, níže podepsaný student, tímto čestně prohlašuji, že text mnou odevzdané závěrečné práce v písemné podobě i na CD nosiči je totožný s textem závěrečné práce vloženým v databázi DIPL2.

Prohlašuji, že předložená práce je mým původním autorským dílem, které jsem vypracoval samostatně. Veškerou literaturu a další zdroje, z nichž jsem při zpracování čerpal, v práci řádně cituji a jsou uvedeny v seznamu použité literatury.

V Ostravě dne 1. 3. 2021 .....

podpis

Completing my dissertation would not have been possible without the continuous support and encouragement of my supervisor, Prof. Irina Perfiljeva, CSc. Thank you for all your valuable comments and for showing me how to look at the problems from a different perspective. I also wish to thank Mgr. Petr Hurtík, Ph.D., without your positiveness, I would give up a long time ago. Lastly, I would like to extend my gratitude to all colleagues at the Institute for Research and Applications of Fuzzy Modeling for all the assistance.

Contents

1 Introduction

The recent years have brought the trend of deep learning that pushed the boundaries of many fields. Generally, deep learning is focused on creating, training, and using mathematical models – *artificial neural networks* (ANN), while using a considerable amount of data. ANN application falls into one of two categories: *classification* or *regression*. Both regression and classification are old problem domains with many working solutions for different tasks. Traditionally, there has been a substantial amount of handcrafting work to solve the classification or regression task.

With the advent of machine learning and later deep learning, handcrafted parts of the algorithms, such as *features*, started to become obsolete. The deep learning algorithms are mostly data-driven – able to infer information and patterns from data. In the case of deep ANN, handcrafted feature extractors were replaced with layers of neurons that learn what *descriptive* features are from presented data. The rise of data-driven approaches had several effects. Firstly, many classification and regression problems (medical application, surveillance, generating real-like image data, scene segmentation and understanding, scene re-creation, text translation, natural language processing, and many others) were solved, or their solutions improved significantly. Secondly, because deep learning algorithms learn from the data, they are hard to interpret. There has been a growing demand for *interpretability*, mostly due to ethical reasons.

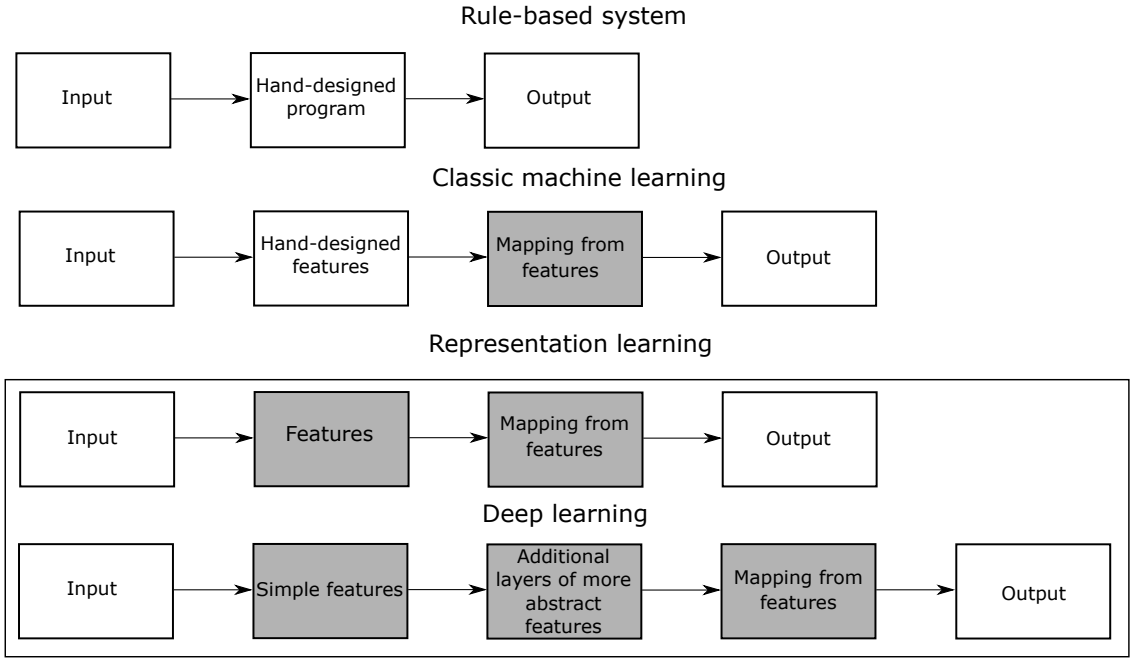


Figure 1: Different kinds of systems, from hand-crafted rules and features to data-inferred features and mappings. Image based on [37].

Image classification and recognition are fields that benefited from deep learning tremendously, and that is because of *convolution*. Convolution works as an image filtering tool, but in the context of ANN, it is considered a *feature extractor*. Whatever feature is extracted depends entirely on the *convolution kernel*. These kernels (matrices) are part of the ANN parameters and, therefore, can be learned.

We propose to initialize ANN convolution kernels with predefined filters. The predefined filters are taken from the particular fuzzy modeling method – *F-transform* [87]. F-transform is a mathematical tool developed in 2001 [87] and since then it has found a lot of application potential in image processing: compression [53], fusion [122, 94], edge detection [90, 91] and image reconstruction [124, 85]. F-transform represents a discrete function in the form of a set of *components*. In our case, a discrete function is an image and the components are image features. Since F-transform can be expressed as a convolution, our initialization filters are those used to get components.

2 Convolutional neural networks and neuro-fuzzy state of the art

2.1 Neural networks as approximators

Neural networks are universal approximators. In 1989, Cybenko [21] proved that a neural network with one hidden layer of sigmoid neurons and potentially unlimited width (number of neurons) can approximate an arbitrary continuous function. This proof leads to an important conclusion: whenever a problem can be expressed by a function, there is a neural network that can be trained to approximate this function. Let us consider a toy example of a noisy function $f(x) = xcos(x) + 2\mathcal{N}(0, 1)$ in Fig. 2(a) and a neural network with a single hidden layer. Using *mean square error* (MSE) as the minimization criterion and a single linear neuron, we can perform *linear regression*, Fig. 2(b). The linear regression finds parameters in an analytical way, in our case we let the parameters converge iteratively. Because the data points do not lie on a line, we use neurons with a sigmoid activation function. In Fig. 2(c) to Fig. 2(h), we can see that adding the neurons going from one to six increases the network ability to approximate the original function.

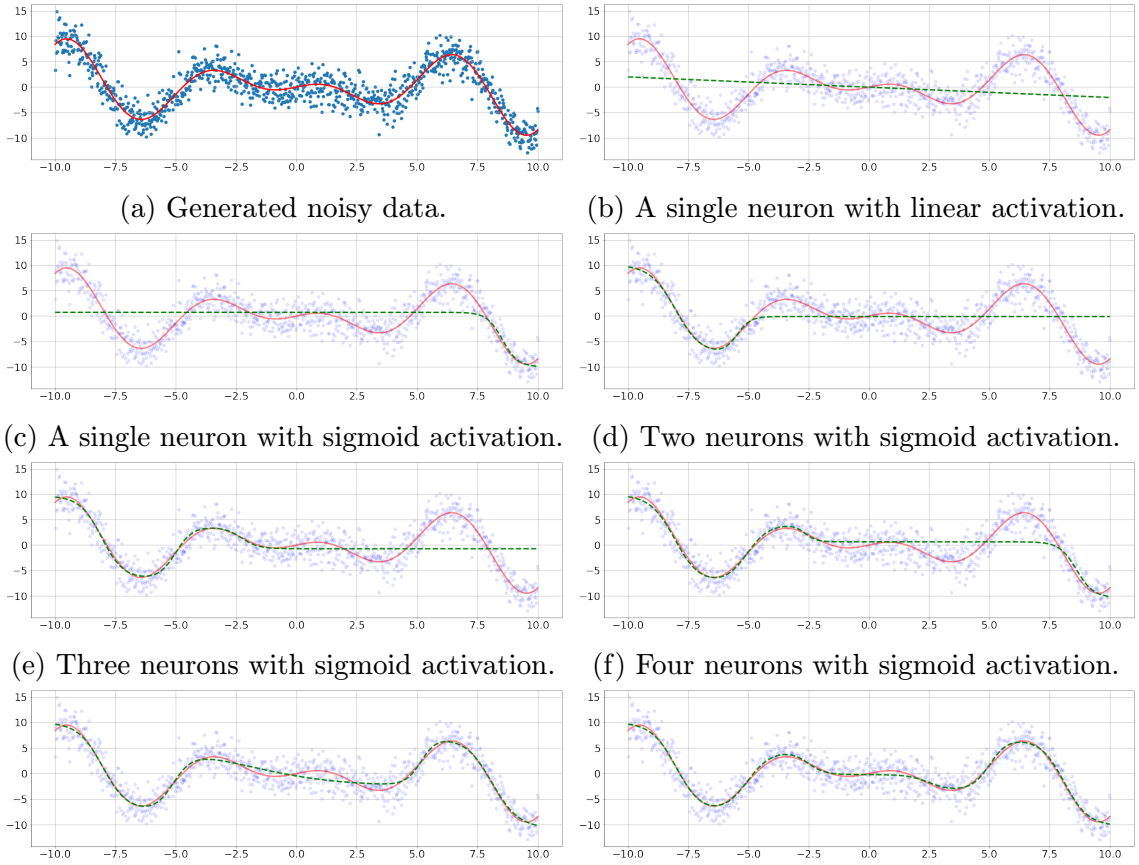
If MLP (multilayer perceptron) with one hidden layer can approximate an arbitrary continuous function, why are there various neural network types?

2.2 Classification and regression problems

Before we get to the types of networks, let us define the problems directly related to this work. Problems being solved with neural networks are *classification* and *regression*. Other problems are their modifications. For example, semantic/instance segmentation is based on pixel-wise classification; object detection is box/polygon regression, time series prediction is regression, playing a game is assigning a correct action to the current state (similar to assigning a label to an object), etc.

Classification is a problem of assigning a D-dimensional data sample $X \in \mathbb{R}^D$ to one or multiple predefined classes enumerated with positive integers $Y \in \mathbb{N}^M$. However, the most common case is to have mutually exclusive classes, where the data sample belongs to only one class, thus $Y \in \mathbb{N}$. Moreover, it is common practice to convert Y to one-hot encoding, a binary vector Y':

$$Y'_i = \begin{cases} 1 \text{ if } i = Y \\ 0 \text{ otherwise} \end{cases} \tag{1}$$



(a) Generated noisy data.
(b) A single neuron with linear activation.
(c) A single neuron with sigmoid activation.
(d) Two neurons with sigmoid activation.
(e) Three neurons with sigmoid activation.
(f) Four neurons with sigmoid activation.
(g) Five neuron with sigmoid activation.
(h) Six neurons with sigmoid activation.

Figure 2: Approximation of $f(x) = xcos(x) + 2\mathcal{N}(0, 1)$ by a neural network with increasing width. Blue dots are data samples, red curve is $xcos(x)$ (without added noise) and dashed green line/curve is network regression result.

The whole classification problem is to find or approximate the true mapping $f : \mathbb{R}^D \to \mathbb{N}$.

Regression is directly related to classification. Regression problem is defined as assigning a real value $Y \in \mathbb{N}$ [55] to a D-dimensional data sample $X \in \mathbb{R}^D$. The goal of regression is to find or approximate the true mapping $f : \mathbb{R}^D \to \mathbb{R}$. Note that regression does not rely on predefined classes; therefore, no apriori knowledge is required.

Let us briefly describe the data format generally used for datasets. In the machine learning supervised setting, a classification problem is represented by a dataset $\{(x_i \in \mathbb{R}^D, y_i \in \mathbb{N}^M) | i \in \{1, \ldots, n\}\}$. In the case of non-imagery data, the most basic structure is a *vector* ($x_i \in \mathbb{R}^{1 \times D}$), where the vector $\boldsymbol{v}$ component v_i is a data sample i-th attribute. Multiple data sample vectors create a *batch* of data and are structured into (data) *matrix* A, where columns are data sample attributes and rows are respective data samples, i.e. $a_{i,j}$ is i-th data sample j-th attribute.

In the case of imagery data, the matrix A ($x_i \in \mathbb{R}^{H \times W}$) represents a raster grid of greyscale pixel intensities, i.e., the matrix A element $a_{i,j}$ is the greyscale intensity of the pixel at (i, j) position. Greyscale intensity is represented by 8 bits, $2^8 = 256$ different shades of grey. RGB images have 3 channels with 8 bits for each channel. RGB images are represented by the *tensors* ($x_i \in \mathbb{R}^{H \times W \times C}$) where tensor T element $t_{i,j,k}$ represents k-th channel intensity of a pixel at (i, j) position. Multiple images create a batch represented by the 4D tensor T ($x_i \in \mathbb{R}^{B \times H \times W \times C}$) with element $t_{i,j,k,l}$ which is l-th channel intensity of a pixel at position (j, k) of i-th image within a batch.

2.3 Problems solved by MLP and CNN

MLP has been used for various problems. It has been successfully used for natural language processing (NLP), for example, *Word2Vec* [72]. Word2Vec is an approach that learns to represent words as vectors. Cosine similarity is used to measure semantic similarity. Another NLP (partial) application of MLP is *BERT* [26]. In 2018, BERT achieved state-of-the-art performance on multiple benchmarks (SQuAD 1.1, SWAG, GLUE, and MNLI). BERT uses a popular concept of *attention* mechanism through *transformer* [123]. Another MLP model built on BERT is *ALBERT* [66] that deals with the problem of ever-increasing sizes of the NLP models. It presents an approach that takes into account computational and memory costs. Recently, transformers were successfully implemented in the image processing problems too. *DETR* [10] has successfully reached state-of-the-art performance in object detection and segmentation categories using a transformer. Another general scheme, similar to a transformer, is the *autoencoder* [104]. Autoencoder consists of two parts: an encoder and decoder. Encoder gradually transforms the input into so-called *latent representation*, more compact than the input itself. Decoder transforms the latent representation back to the input. Thus, the autoencoder encodes the input and decodes it back to its original form. Autoencoders are often used for dimensionality reduction, feature learning, and due to the link between latent variables and generative models, for generative modeling too [36]. Apart from the MLP variant, there are also CNN autoencoders [46].

Time series forecasting is a field with many developed methods, including MLP. One instance is [65], which uses a deep belief network with restricted Boltzmann machines. Although the authors did not reach state-of-the-art results on IJCNN'04, they presented a novel way to forecast time series. Time series were the object of study in [69], too. However, the authors were not interested in forecasting but rather classifying (medical time series). They compared LSTM with MLP and showed that

the best results are obtained by combing them together. A new training scheme was proposed in [119] based on *extreme learning machines* (ELM). ELM does not train the weights of hidden nodes; instead, they treat them as random projections. [119] combines ELM with MLP for various problems. An interesting combination of CNN and MLP is *Network In Network*, which replaces the convolution operation with a whole MLP network, thus the name "network in network". From the examples mentioned above, we can see that MLP is frequently used in the domains of NLP time-series processing, and to some degree in image processing.

Let us discuss CNN now. The first CNNs were built on the ideas of Hubel and Wiesel [51, 52]. In their work, they have studied structures in a cat visual cortex. The authors described two different cells: a simple and complex cell. These cells are sensitive to a specific input stimulus from optical sensors (edges and orientations). The study has shown that complex cells are having better spatial invariance than simple cells, i.e., translation and movement in one direction.

Such a network is Fukushima's *Neocognitron* [30]. The Neocognitron features several important concepts: convolution, weight sharing, and subsampling operation. Convolution is realized via a sliding window across the inputs with a given filter(s). Each sliding window position is processed by a single convolutional unit. The window defines the unit receptive field (a spatial area in input where the unit is sensitive to stimulus). The filter is given by a vector of weights on connections to the unit receptive field. The convolutional units receptive fields cover[1] the input vector and form one set of convolutional units sharing the same weights (the same filter). A single convolutional layer contains ≥ 1 set(s) of convolutional units. The convolutional layer output is a two-dimensional array for each filter and serves as an input to the next layer(s).

Subsampling units are connected to a small, spatially close part of the convolutional layer output with fixed, weighted connections. The Neocognitron subsampling unit is averaging its input, in contrast to today popular *max pooling* [126] taking a maximum from the unit receptive field.

Later, in 1998, LeCun et al. introduced CNN *LeNet* [67]. The LeNet is in many ways similar to the Neocognitron: convolutional units, shared weights, subsampling units (average). For training LeNet, the authors have created a famous dataset MNIST from a much bigger dataset available from NIST[2] that has become a popular benchmark. The MNIST consists of 70 000 labeled training/testing greyscale pictures of handwritten digits. The digits are normalized and centered into 28x28

[1] A distance between centers of a two successive sliding window can be specified. Therefore the windows might or might not overlap and cover the entire input.

[2] National Institute of Standards and Technology

images. Unlike Fukushima's Neocognitron, the LeNet was trained in a supervised way using the backpropagation algorithm, an efficient way to apply a *gradient descent*.

The backpropagation was used on a CNN with max pooling subsampling layers for the first time in 2007 [49]. Convolution - max pooling blocks have become the standard components of today's best CNNs. Speed up of training with backpropagation brought the first implementation of CNNs on GPUs. The authors have reported 4 to 20 times higher training speed [81, 12].

The GPU implementation played a crucial role in breaking the 0.4% error of the MNIST in 2010 by Ciresan et al. [17] (0.35%). The authors did not use CNN but a multilayer perceptron, they have still obtained better results. This success is attributed to an augmentation of the MNIST by the following transformations: rotation, scaling, horizontal shearing, and elastic deformations. The five networks have a variable number of layers (2 to 9) and 1.34 to 12.11 million trainable parameters. One year later, almost the same group created a GPU implementation of CNNs [16]. GPU-based CNNs have since been (a part of) winning solutions in many competitions.

In the same year, Ciresan et al. have published their *Multi-Column* CNN [19]. The model consists of multiple ordinary CNNs whose outputs are taken as votes that are averaged into a final output. This concept has surpassed a human accuracy of 1.19% on the traffic sign dataset [112] with an accuracy of 1.02%. The Multi-Column CNN was also applied to the MNIST and has surpassed a human accuracy error of $\approx 0.4\%$ to $\approx 0.2\%$.

A deep CNN was applied to the dataset ImageNet, significantly bigger and more complex than the MNIST, in 2012 by Krizhevsky et al. [63]. The network won the ILSVRC competition by a big margin. The ImageNet training set in 2012 consisted of 1.2 million labeled examples belonging to one of the 10 000 classes. Since then, the ILSVRC has produced many famous networks solving object detection and object classification problems (ZFnet [132] 2013, GoogleNet (Inception v1) [115] and VGG [110] 2014, ResNet [42] 2015, assembly of CUImage[3]).

The GoogleNet was later expanded into Inception v2 and v3 and presented in a single paper [117] and further into Inception v4 and Inception-ResNet, again in a single paper [116]. v2/3/4 improves Inception with traditional elements (RMSprop optimizer, convolution factorization, BatchNorm, etc.) while Inception-ResNet adopts residual connections. The latest iteration of ILSVRC classification and object detection categories were won by entries based on Dual Path Networks [13], combining the

[3]Description available at `http://image-net.org/challenges/LSVRC/2012/results`.

DenseNet (layers have access to the outputs of all previous layers) [50] and ResNet. Most recent papers started to explore networks generating other networks [133] used for classification and detection.

Convolutionally based networks were used for segmentation and recognition problems too. Let us mention a couple of significant architectures: UNet [103], iteration of R-CNN [33, 32, 102] or iteration of YOLO [101, 99, 100][4].

A different type of architecture with convolutional variation is an autoencoder [5]. Autoencoders are usually used as a means of pre-training. Variational autoencoders [60] learn latent representation with the help of Gaussian distribution.

In 2014, [38] introduced now a very popular principle of GAN, two networks competing against each other. Since then, GANs have been successful in many problem domains such as data generation.

Lastly, we will mention Pixel Recurrent Neural Networks [82] used for inpainting and capsule networks, a novel architecture from "Godfather of Deep Learning" Geoffrey Hinton [106].

All recent advances would not be possible without proper support from hardware (IBM PowerAI DDL [14], NVIDIA GPUs, etc.) and software (Tensorflow [1], Keras [15], Caffe [56], PyTorch [84], etc.).

2.4 Multilayer perceptron and Convolutional neural network architectures

MLP and CNN are similar architecture types with a common stage – feature extraction. We can view feature extraction from two different views depending on the type of the network. Firstly, the CNN perspective views the feature extraction in terms of classical computer graphics. Convolutions extract gradually more and more features that are more and more complex or abstract. The result of the stage is a vector of rich and descriptive features, Fig. 3.

Secondly, MLP feature extraction can be interpreted as the transformation of the input space into a space that is more suitable for a given task, i.e makes the data samples separable, Fig. 4.

The main difference is how MLP and CNN transform the input data. The difference is reflected in the data itself. CNN uses data with spatial dependencies, i.e., pixels spatially close to each other are a part of the same object, texture, edge, etc. Convolution is excellent at exploiting these spatial dependencies. Moreover, convolution shifts its window across the data, which leads to invariance to feature

[4]YOLO v3 paper is written quite humorously.

[5]`https://colah.github.io/posts/2014-03-NN-Manifolds-Topology`

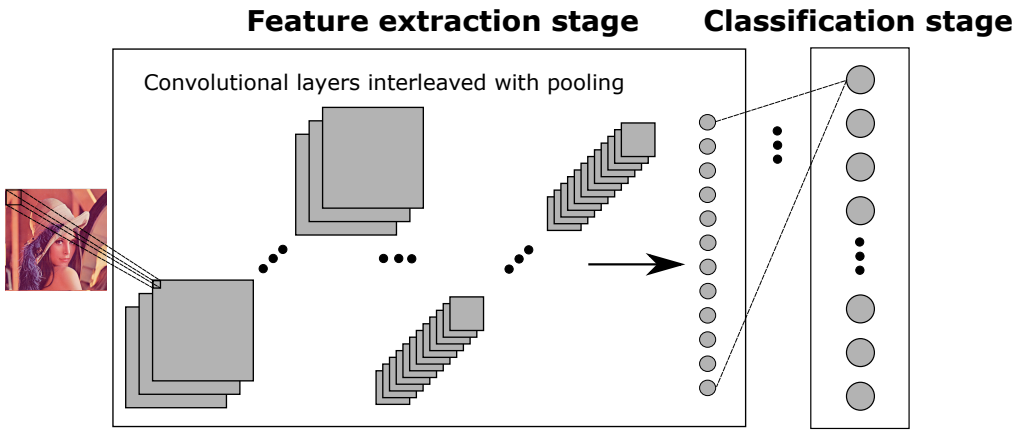


Figure 3: Two stages of the neural network – feature extraction and classification.

translation. MLP is generally used for data without spatial dependencies, such as tabular data. Let us demonstrate the advantage of CNN over MLP on image data (CIFAR-10). We trained a simple CNN and MLP with an almost identical number of trainable parameters, and in Fig. 5 are the results. MLP reached the worst test accuracy out of the three networks trained. For comparison, we included CNN with and without max pooling as the pooling is an essential part of CNN.

Let us take a closer look at MLP. MLP acts as a function f parametrized by θ. θ is a set of all trainable and non-trainable parameters, i.e., weights, biases, batch normalization (γ, β) etc., and x is data.

$$\hat{y} = f(x, \theta) \tag{2}$$

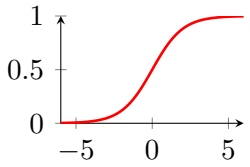
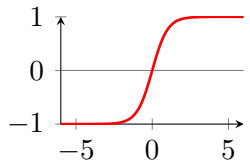
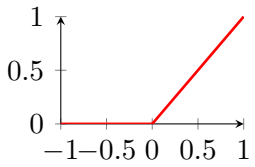
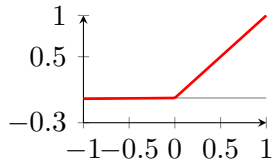
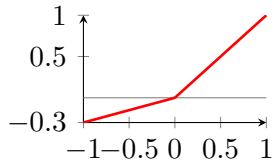
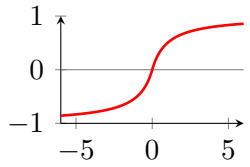
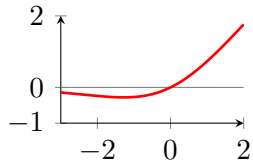
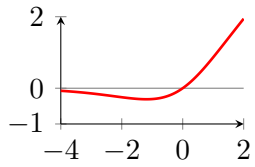
A neuron is a basic computational unit of a neural network. It is determined by the weight vector $\boldsymbol{w} \in \theta$, the bias $b \in \theta$ and the activation function $g : \mathbb{R} \to \mathbb{R}$. A neuron output a is given as:

$$a = g(b + \sum_i w_i x_i) = g(b + \boldsymbol{w} \cdot \boldsymbol{x}). \tag{3}$$

where $(\cdot)$ is a dot product. A set of neurons forms a layer. We will use a_k^ℓ for k-th neuron activated output at the ℓ layer.

An activation function is the main tool to bring non-linearity to the network. The activation function has to be differentiable whenever gradient descent-based optimization is employed. The functions such as *sigmoid* or *tanh* were mostly abandoned in the modern era due to the saturated neuron problem [97] and slow learning [63]. An overview of activation functions is in Table 1.

Table 1: Overview of neuron activation functions.

name	formula	derivative	graph
Sigmoid	$\frac{1}{1+e^{-x}}$	$\frac{1}{1+e^{-x}}(1-\frac{1}{1+e^{-x}})$	
Hyperbolic tangent	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$1-\left(\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}\right)^{2}$	
ReLU	$max(0,x)$	$\begin{cases}1 & \text{if } x>0\\ 0 & \text{othewise}\end{cases}$	
LReLU	$max(0.01x,x)$	$\begin{cases}1 & \text{if } x>0\\ 0.01 & \text{othewise}\end{cases}$	
PReLU[0] [43]	$max(\alpha x,x)$	$\begin{cases}1 & \text{if } x>0\\ \alpha & \text{othewise}\end{cases}$	
Softsign [121]	$\frac{x}{1+\lvert x\rvert}$	$\frac{1}{(1+\lvert x\rvert)^{2}}$	
Swish [98]	$\frac{x}{1+e^{-x}}$	$\frac{1}{1+e^{-x}}+\frac{x}{1+e^{-x}}\left(1-\frac{1}{1+e^{-x}}\right)$	
Mish [74]	$x\cdot tanh(ln(1+e^{x}))$	$\frac{e^{x}4(x+1)+4e^{2x}+e^{3x}+e^{x}(4x+6)}{(2e^{x}+e^{2x}+2)^{2}}$	

[0] $\alpha = 0.3$ in the graph. α is a trainable parameter.

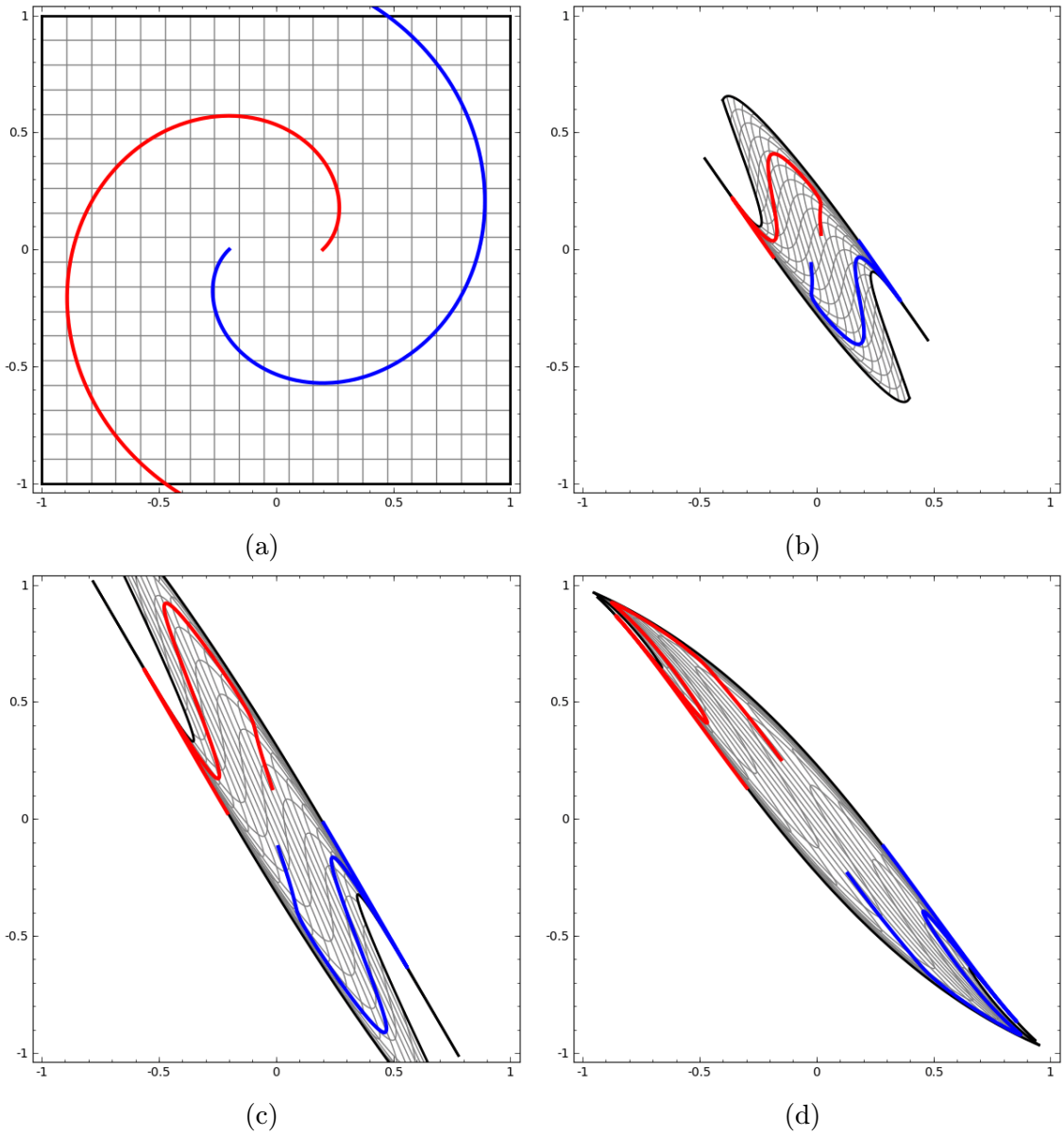


Figure 4: MLP gradually transforming space, such that two classes represented by spirals are linearly separable. Images are taken from webpage[5].

Rewriting (2), MLP can be expressed as:

$$f(\mathbf{x}, \theta) = g_K(...g_2(g_1(\mathbf{x}W^1)W^2)...W^K) \tag{4}$$

where $W^\ell \in \mathbb{R}^{m \times n}$ are a neuron weight matrices in the ℓ-th layer, $g_{1,\cdots K}$: are (non-linear) activation functions (Table 1) and K is the number of network layers. From the equation, we can see that each MLP layer linearly transforms the previous layer output (or input vector $\mathbf{x}$) by multiplying it with its weight matrix W and then applying an activation function g. The last layer activation function g_K is usually

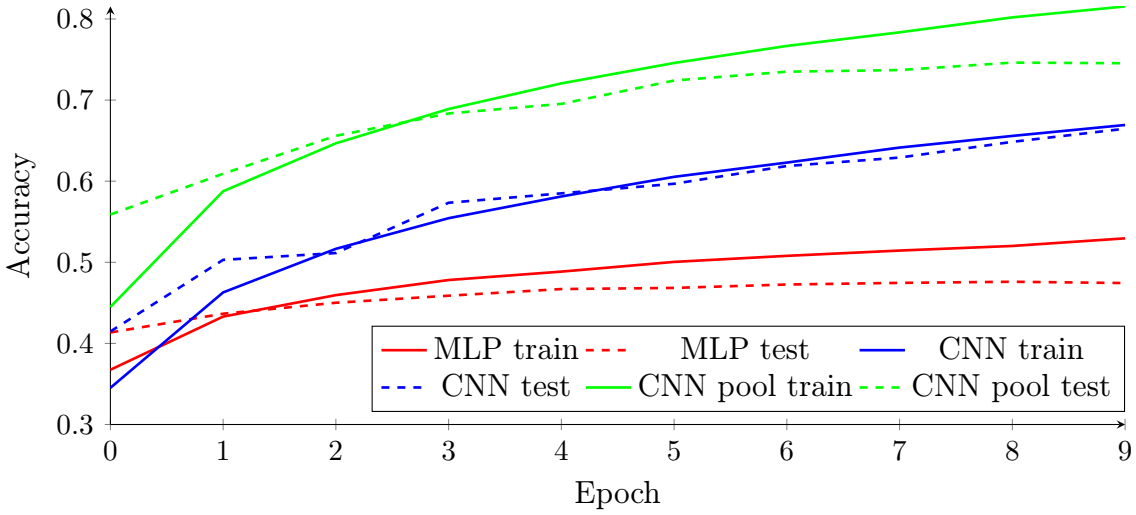


Figure 5: Toy MLP and CNN networks trained on CIFAR-10. CNN was trained with and without pooling.

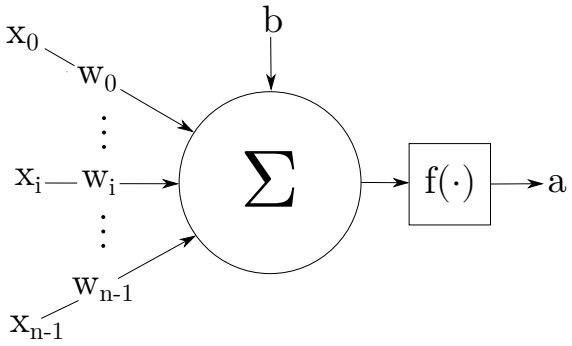


Figure 6: Illustration of a neuron (3).

different from $g_{1\cdots K-1}$ due to the need of a specific format of output values.

CNN changes (4) by replacing some of the matrix multiplications with a convolution operation.

The convolution neuron differs from the typical neuron by having spatially restricted connections. The convolution neuron is only connected to a part of the input vector/tensor, simulating the behavior of a convolution window. In some cases, the convolution neuron can be connected to the whole input data, which is equivalent to a convolution window covering the whole input. We define convolution neuron output a at position (i, j) as:

$$a_{i,j} = g\left(b + \sum_{m=-r_0}^{r_0} \sum_{n=-r_1}^{r_1} \boldsymbol{x}_{i+m,j+n,*} \odot \boldsymbol{w}_{m+r_0,n+r_1,*}\right) \tag{5}$$

where X is a 3D tensor with size $I \times J \times C$, the weight tensor W has size $M \times N \times C$, $\boldsymbol{r} = (\lfloor \frac{M}{2} \rfloor, \lfloor \frac{N}{2} \rfloor)$, $\odot$ is Hadamard (element-wise) product, $g(\cdot)$ is an activation function and b is bias. Note that (5) is for the case of 2D data. Modern frameworks provide 1D and 3D convolution neurons. Let us note that convolution can be expressed as matrix multiplication, where the convolutional kernel is represented by a sparse matrix.

Batch normalization [54] is a normalization block, usually placed between convolutional or fully connected layers. The purpose of batch normalization is to normalize batch data with respect to batch mean and variance. Apart from variance and mean, there are two trainable parameters β, γ. Normalized batch tensor $\hat{T}$ is given as:

$$\hat{T} = \gamma \frac{T - \mu_T}{\sqrt{\sigma_T^2 + \epsilon}} + \beta \tag{6}$$

where T is the input batch tensor, μ_T is the batch mean, σ_T^2 is the batch variance, and ϵ is a numerical constant for stability.

Skip/residual connection [41] is a connection between the input and output of the same/different layer(s). Through the connection, the input is aggregated (usually adding or concatenating) with the output. Formally, we can write that ℓ-th layer output a^ℓ is aggregated with $a^{\ell-i}$: $a^\ell \circ g(a^{\ell-i})$ where $\circ$ is an aggregation function and $g(\cdot)$ is some transforming function such as the convolution layer(s) or plain identity function. The main contribution of residual connections is the ability to successfully train deeper networks.

We will briefly describe optimizers and loss functions, essential parts of NN training, albeit not part of the architecture.

An optimizer is an algorithm updating network parameters according to the loss function. Most of the optimizers are based on the first-order optimization method – gradient descent. Among the modern optimizers are Adam [59], Adagrad [27], AdaDelta [131], RMSprop[6] and others.

Loss function dictates how the difference between the network and the desired output is calculated in a supervised setting. Since the optimizers are mostly first-order methods, the loss functions have to be differentiable. The loss function greatly influences how the network is learning a given task, i.e., learning probability distribution (*cross-entropy*), working with a class imbalanced dataset (*focal loss*), or maximizing margin (*SVM*). **A NN solves an (optimization) problem by minimizing the loss function using the *optimizer*.**

[6]Unpublished. It appeared first in G. Hinton Coursera course. Slides by authors are available at `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

2.5 Neuro-fuzzy systems

A neuro-fuzzy system combines a fuzzy system and a neural network that uses some form of learning. Not many studies have contributed to neuro-fuzzy in computer vision since 2012 until the present time when deep learning became popular. Allow us to begin with MNIST classification contributions. Authors of [6] improved MNIST recognition of neuro-fuzzy systems by optimizing used features and architectures and reached the accuracy of 99.52% with a prediction time of 94.6s for 10 000 testing images. The accuracy is close to the results of [20, 18] from the years 2010 and 2012. The paper [11] approaches feature selection using 2-D scaling moments based on wavelet transform and compares different classifiers (support vector machines/classifiers, artificial neural networks, neuro-fuzzy classifiers, and others). The best accuracy of 99.39% was obtained by support vector classifier. The neuro-fuzzy approach achieved the accuracy of 98.72%. Handwritten character recognition has been the subject of the study [2] as well, although data were taken from Chars74k and the testing numbers were drawn by hand. The main contribution is, again, in selecting the best features for classification. The authors used three-fold cross-validation, achieving 97.22% accuracy. From visual inspection, their data, after preprocessing, are quite similar to MNIST data, Fig. 7. One of the most recent

Figure 7: Example of data used in [2].

publications [29] uses Fuzzy Deep Belief Net (FDBN) to classify MNIST with different types and levels of noise. The FDBN architecture is based on two papers [46, 45] by Hinton et al. The authors compared their results with standard Deep Belief Networks and achieved better results for data with and without noise. The deep architecture was utilized in [4] where a fuzzy modeling module and a long short-term memory network were separated. The paper demonstrated image data utilization to control the pressure of a surgery tool operated by a robotic arm. Shukla and Soorya [109] used an adaptive network-based fuzzy inference system (ANFIS) for determining whether a person is demented or not. The training data consisted of 150 MRI images from which 35 features per image were extracted. The authors then used these features to train ANFIS. A combination of fuzzy rules with the neural network was presented in [24]. The authors apply fuzzy rules and deep neural net-

work to input data, and the output of these two is fused and finally classified with the softmax classifier. The whole assembly is realized as a neural network, and therefore, standard training algorithms are used. The assembly was tested against [68], a fuzzy neural network with SVM, and [57], a self-constructing neural fuzzy inference network on preprocessed data of scenes [25]. The idea of ANFIS was expanded with multiple instance classification capabilities in [58].

2.6 Features

Regression or classification methods based on neuro-fuzzy or deep learning share a common denominator - usage of features. Features are universally used for object description whenever we discuss column attributes in a data matrix, a feature vector produced by a NN, or edges detected in the image. Extraction of good features is a crucial step in designing a successful classification/regression model. So much so, there is a whole field of machine learning dedicated to *representation learning* [7]. Before representation learning became mainstream, feature extraction was dominated by handcrafted methods such as *Principal Component Analysis* (PCA) [86], image edge detectors: *Sobel* [111], *Prewitt* [96], *Laplacian of Gaussian*, *Canny edge detector* [9], or feature descriptors: *histogram of oriented gradients* (HOG) [71] and *scale-invariant feature transform* (SIFT) [70]. However, these handcrafted features are slowly pushed away in favor of learned features from data, as those are suited for given data. Quite often, the learned features match those extracted by handcrafted, so there is an overlap. If one could determine what features will be extracted at the end of the learning, the training time could be decreased significantly. This is the **main idea** of our work. Choosing the right features with certain properties in mind allows the user to control the training process, understand model decisions, and possibly correct unwanted behavior.

For this purpose, we utilize F-transform, as it extracts very descriptive features, usable even for complete object/function reconstruction [92]. Furthermore, F-transform features display hierarchicality, similar to the neural network feature extraction stage (Hubel and Weisel [51, 52]). All of the F-transform properties: different F-transform degrees of features and chainability of F-transform are well mappable to neural networks.

In the case of CNNs, features are extracted through a convolution. Convolution uses a *convolutional kernel* – 2D matrix or a 3D tensor of weights that define the extracted feature. Vlasanek and Perfilieva [125] introduced convolutional kernels for F^0 and F^1-transform and we further added kernels for F^2-transform. We used this set of F-transform kernels to initialize CNN convolutional layer kernels and analyze

their impact on CNN behavior. F^n-transform kernels have experimentally proven to be the same or better feature extractors than the learned ones.

2.7 Commonly used initialization methods

Commonly used methods for weight initialization come from statistics. For example, AlexNet uses initialization from the normal distribution $\mathcal{N}(0, 0.01)$. Such initialization suffers from vanishing and exploding gradients. So while it was a winning solution in 2012, there are better parameters for normal and uniform distributions. In 2010, Glorot and Bengio published their contribution [34] which is today referred to as *Xavier* initialization. Their idea builds on the fact that weights should be scaled according to the number of inbound and outbound connections to the layer. This leads to the propagation of a stable gradient that does not vanish or explode. The results were derived for the linear case but worked well for non-linearities: *hyperbolic tangent*, and *softsign* with range $\in [-1, +1]$. Their initialization comes from two distributions, $\mathcal{N}\left(0, \sqrt{\frac{2}{fan_{in}+fan_{out}}}\right)$ and $\mathcal{U}\left(-\sqrt{\frac{6}{fan_{in}+fan_{out}}}, +\sqrt{\frac{6}{fan_{in}+fan_{out}}}\right)$, where fan_{in} and fan_{out} are number of incoming and outgoing connections.

Probably the most popular activation function – *ReLU* [40, 35] is defined as $f(x) = max(0, x)$ with range $\in [0, \infty]$. So Xavier initialization is not the best match. He et al. saw this problem and modified Xavier initialization with ReLU range in mind and showed that the initialization proposed by them leads to faster convergence. Their initialization is similar to Xavier, except for the *variance* that ignores the outgoing connections: $\mathcal{N}\left(0, \sqrt{\frac{2}{fan_{in}}}\right)$ and $\mathcal{U}\left(-\sqrt{\frac{6}{fan_{in}}}, +\sqrt{\frac{6}{fan_{in}}}\right)$.

Kuhmar, in his paper [64], describes formally how and why Xavier and He initialization works and provides a general initialization framework for different activation functions. Other initialization methods have been published but did not become widely used.

[61] propose to use statistics from the forward pass of a small subset from the dataset. Saxe et al. [108] used orthonormal matrices to demonstrate linear network learning dynamics. The network learned general concepts (animal, flower, etc.) first and then more specific ones (horse, bird, rose, etc.). This was demonstrated on a specially prepared dataset and shown through *modes*. Similar behavior transfer over to non-linear networks as well. The method that is driven by log norm of the gradients – Random Walk Initialization was presented in [114] and relies on finding the correct constants for scaling random matrices that are multiplied with gradients. [80] uses initialization similar to Xavier and He, but the main contribution is a novel way how to propagate gradients. Instead of using a backpropagation algorithm, each layer has a gradient fed straight from the output layer. While this is not an

initialization method per se, most, if not all, the mentioned methods are built on gradient propagation. Delta orthogonal initialization based on mean-field theory was introduced in [128]. By weight initialization in the order-to-chaos (hyperparameter space), as deep as 10 000 layers the network can be trained without skip connection or batch normalization. MetaInit [22] defined a *gradient quotient* that measures the change in gradient after a single step of gradient descent. The (meta) learning is used to find the initialization with the lowest change, meaning that the found initialization produces the lowest error. The authors claim that MetaInit can recover from bad initialization. Mishkin & Matas [73] extended [108] orthonormal initialization such that the layer output has unit variance. Bengio et. al [8] extended [127] to initialize a network with greedy, layer-wise unsupervised autoencoders.

We can see that most of the initialization methods work with the statistical properties of the distributions from which the weights are sampled and they emphasize proper gradient propagation. This point of view is connected with training. While we follow scaling and centering practices, our focus is on the weights and their semantic meaning – not sampling for convectional distributions but using F-transform kernels.

Table 2 contains the values of the ResNet20 loss function trained on CIFAR10 with combinations of activation functions (Table 1) and different initialization schemes. The left part of the table is sorted with respect to the loss function values on the CIFAR10 test and the right side with respect to the difference between loss function values on the CIFAR10 train and test. We can see that the best performing activation function is Leaky ReLU (LReLU), which has one of the lowest overfit. Interestingly, LReLU works well with Xavier, He, and F-transform initialization (introduced later in section 6.2). We can conclude that in the case of ResNet20, LReLU is the best performing activation function regardless of the initialization.

Table 2: Resnet20 [44] trained on CIFAR-10 for 200 epochs. Reported results are for different combinations of activation functions and initializations. The left part of the table shows the loss values sorted with respect to the test loss. The right part of the table shows loss values sorted with respect to the difference between train and test loss (overfitting/underfitting). Color highlights mark the best performing combinations in both parts.

sorted w.r.t. test				**sorted w.r.t. test/train diff.**			
	train	test	diff.		train	test	diff.
Xav Uni LReLU	0.197	0.399	0.202	He Uni LReLU	0.205	0.401	0.197
Xav Norm LReLU	0.198	0.401	0.203	Xav Norm Softsign	0.231	0.429	0.198
He Uni LReLU	0.205	0.401	0.197	Xav Uni Softsign	0.229	0.428	0.199
Ft Sig	0.198	0.403	0.205	Xav Uni LReLU	0.197	0.399	0.202
Ft LReLU	0.196	0.409	0.213	He Norm Softsign	0.233	0.436	0.202
Xav Uni Mish	0.152	0.41	0.258	Xav Norm LReLU	0.198	0.401	0.203
Ft Swish	0.154	0.411	0.258	Ft Sig	0.198	0.403	0.205
He Norm LReLU	0.2	0.414	0.213	He Uni Softsign	0.226	0.435	0.208
Ft Mish	0.153	0.414	0.262	Ft Softsign	0.22	0.428	0.208
He Norm Sig	0.203	0.416	0.213	He Norm Sig	0.203	0.416	0.213
Xav Uni Tanh	0.182	0.42	0.238	He Norm LReLU	0.2	0.414	0.213
He Uni Sig	0.201	0.422	0.221	Ft LReLU	0.196	0.409	0.213
He Norm Mish	0.156	0.423	0.267	He Uni Sig	0.201	0.422	0.221
Xav Norm ReLU	0.186	0.427	0.241	Xav Uni Sig	0.203	0.432	0.228
Ft ReLU	0.186	0.427	0.241	Xav Norm Sig	0.206	0.438	0.232
Xav Uni Softsign	0.229	0.428	0.199	Xav Uni Tanh	0.182	0.42	0.238
He Norm Tanh	0.184	0.428	0.244	Xav Norm ReLU	0.186	0.427	0.241
Ft Softsign	0.22	0.428	0.208	Ft ReLU	0.186	0.427	0.241
Xav Norm Softsign	0.231	0.429	0.198	He Uni ReLU	0.186	0.429	0.243
He Uni Tanh	0.182	0.429	0.246	He Norm Tanh	0.184	0.428	0.244
He Uni ReLU	0.186	0.429	0.243	He Norm ReLU	0.184	0.43	0.245
He Norm ReLU	0.184	0.43	0.245	He Uni Tanh	0.182	0.429	0.246
He Uni Mish	0.158	0.43	0.272	Xav Norm Tanh	0.183	0.431	0.248
Xav Norm Tanh	0.183	0.431	0.248	Xav Uni Mish	0.152	0.41	0.258
Xav Norm Mish	0.15	0.432	0.282	Ft Swish	0.154	0.411	0.258
Xav Uni Sig	0.203	0.432	0.228	Ft Tanh	0.183	0.442	0.26
He Uni Swish	0.155	0.433	0.278	Xav Uni ReLU	0.187	0.45	0.262
Xav Norm Swish	0.15	0.434	0.284	Ft Mish	0.153	0.414	0.262
He Uni Softsign	0.226	0.435	0.208	He Norm Mish	0.156	0.423	0.267
He Norm Softsign	0.233	0.436	0.202	He Uni Mish	0.158	0.43	0.272
Xav Uni Swish	0.155	0.437	0.282	He Uni Swish	0.155	0.433	0.278
Xav Norm Sig	0.206	0.438	0.232	Xav Norm Mish	0.15	0.432	0.282
He Norm Swish	0.157	0.442	0.285	Xav Uni Swish	0.155	0.437	0.282
Ft Tanh	0.183	0.442	0.26	Xav Norm Swish	0.15	0.434	0.284
Xav Uni ReLU	0.187	0.45	0.262	He Norm Swish	0.157	0.442	0.285

3 The aim of thesis

This thesis focuses on the fusion of two methodologies: F-transform and convolutional neural networks. We analyze the various NNs where the initialization of convolution-based neurons is made using the apparatus of higher degree F-transform. Our analysis is based on the proven ability to represent datasets with F-transform. And finally, we analyze how the kernels initialized by the F-transform are changing throughout the training process.

The obtained fusion results are evaluated using multiple relevant metrics: accuracy, increase or decrease in computational and time complexity, and explainability. Especially explainability is an important topic, let us mention the death of Elaine Herzberg[7], driven over by a self-driving car (multiple oversights on different levels) or Amazon AI[8] gender-biased recruiting tool (bias included in data). Both cases were later explained, but they raised an important point of machine learning explainability. We contribute to the topic of explainable methodologies as F-transform is, by its nature, clearly explainable.

3.1 Goals, methodology, and contributions

Let us summarize the individual goals of the thesis:

- Analyze MLP, CNN, and their differences and briefly describe the evolution of both architecture types.
- Design a way how to integrate F-transform and convolutional neural networks.
- Prove the efficiency of the developed tools through performance evaluation, using several architectures, image datasets, and standard metrics such as accuracy, training time, and the number of free parameters.

To explain how we achieve these goals, the thesis consists of the following parts:

- In the introductory part, we describe the fundamental NN capabilities and follow with a problem statement. Next, we characterize MLP, CNN, their differences, evolution, and usage. We follow up by discussing common initialization schemes.
- In the second part, we give details of the proposed approach for fusing F-transform and convolutional neural networks.

[7] https://en.wikipedia.org/wiki/Death_of_Elaine_Herzberg

[8] https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G

- In the last part, we primarily deal with the setup of experiments and prove our tool capabilities.

4 F-transform

F-transform is a technique to transform a function into a component representation and back. The components can be understood as space localized function features. The components are sufficient to reconstruct the original function with arbitrary precision. The process of generating components is dependant on basic functions (forming a *fuzzy partition* subsec. 4.1), their shape and width as well as the distance in between them. The thesis focuses mostly on basic functions with a triangular shape, constant width, and equidistant spacing – establishing the so-called *uniform* fuzzy partition. The fuzzy partition is used for backward transformation too - *inverse* F-transform. Traditionally, the same fuzzy partitions were used for forward and inverse F-transform. However, it has been shown that choosing different fuzzy partitions for the forward and inverse stages has additional benefits [92].

In relationship to other mathematical transformations, F-transform is closely tied with a convolution. Basic function support corresponds to the convolution kernel's width, the basic function shape corresponds to the convolution kernel shape, and distance between basic function nodes to convolution *stride* (this is non-standard convolution property that comes from the field of convolutional neural networks).

In the following text, we will recall higher degree F-transform properties, especially F_2-transform, as defined in [75], a work heavily inspired by [89], and further adapt the notation to the thesis.

4.1 Fuzzy partition

The F-transform is the result of a sequence of weighted projections of an object function (image, signal, etc.) on blocks of orthogonal basic functions with the corresponding (to blocks) common restricted supports. Each block is determined by the translation of the weight function, which is called *generating*. All considered translations of a generating function constitute a fuzzy partition.

Definition 1. Let $n > 2$, $a = x_0 = x_1 < \ldots < x_n = x_{n+1} = b$ be fixed nodes within $[a, b] \subseteq \mathbb{R}$. Fuzzy sets $A_1, \ldots, A_n : [a, b] \to [0, 1]$, identified with their membership functions defined on $[a, b]$, establish a fuzzy partition of $[a, b]$, if they fulfill the following conditions for $k = 1, \ldots, n$:

1. $A_k(x_k) = 1$;
2. $A_k(x) = 0$ if $x \in [a, b] \setminus (x_{k-1}, x_{k+1})$;
3. $A_k(x)$ is continuous on $[x_{k-1}, x_{k+1}]$;

4. $A_k(x)$ for $k = 2, \ldots, n$ strictly increases on $[x_{k-1}, x_k]$ and for $k = 1, \ldots, n-1$ strictly decreases on $[x_k, x_{k+1}]$;

5. for all $x \in [a, b]$ holds the Ruspini condition

$$\sum_{k=1}^{n} A_k(x) = 1. \tag{7}$$

The elements of a fuzzy partition $\{A_1, \ldots, A_n\}$ are called *basic functions*.

If the nodes $x_1, \ldots, x_n$ are h-equidistant, i.e., for all $k = 2, \ldots, n, x_k = x_{k-1} + h$ where $h = (b-a)/(n-1)$, and hold two aditioinal properties for $k = 2, \ldots, n-1$:

7. $A_k(x_k - x) = A_k(x_k + x)$ for all $x \in [0, h]$;

8. $A_k(x) = A_{k-1}(x - h)$ and $A_{k+1}(x) = A_k(x - h)$ for all $x \in [x_k, x_{k+1}]$;

then the fuzzy partition $\{A_1, \ldots, A_n\}$ is h-uniform, as stated in [47].

In particular, an h-uniform fuzzy partition of $[a, b]$ can be obtained using the so called *generating function*

$$A : [-1, 1] \to [0, 1], \tag{8}$$

which is defined as an even, continuous and positive function everywhere on $[-1, 1]$ except for on the boundaries, where it vanishes. Basic functions $A_2, \ldots, A_{n-1}$ of an h-uniform fuzzy partition are rescaled and shifted copies of A in the sense that for all $k = 2, \ldots, n-1$;

$$A_k(x) = \begin{cases} A(\frac{x - x_k}{h}), & x \in [x_k - h, x_k + h], \\ 0, & \text{otherwise.} \end{cases}$$

Below, we will be working with one particular case of an h-uniform fuzzy partition that is generated by the triangular-shaped function A^{tr} and its h-rescaled version A_h^{tr}, where

$$A^{tr}(x) = 1 - |x|,\ x \in [-1, 1],\ \text{and}\ A_h^{tr}(x) = 1 - \frac{|x|}{h},\ x \in [-h, h].$$

A fuzzy partition generated by the triangular-shaped function A^{tr} will be referred to as *triangular shaped*.

4.2 Space $L_2(A_k)$

Let us fix $[a, b]$ and its h-uniform fuzzy partition $A_1, \ldots, A_n$, where $n \geq 2$ and $h = \frac{b-a}{n-1}$. Let k be a fixed integer from $\{1, \ldots, n\}$, and let $L_2(A_k)$ be a set of square-

integrable functions $f : [x_{k-1}, x_{k+1}] \to \mathbb{R}$. Denote $L_2(A_1, \ldots, A_n)$ a set of functions $f : [a, b] \to \mathbb{R}$ such that for all $k = 1, \ldots, n$, $f|_{[x_{k-1}, x_{k+1}]} \in L_2(A_k)$. In $L_2(A_k)$, we define an *inner product* of f and g

$$\langle f, g \rangle_k = \int_{x_{k-1}}^{x_{k+1}} f(x)g(x)d\mu_k = \frac{1}{s_k} \int_{x_{k-1}}^{x_{k+1}} f(x)g(x)A_k(x)dx,$$

where

$$s_k = \int_{x_{k-1}}^{x_{k+1}} A_k(x)dx.$$

The space $(L_2(A_k, \langle f, g \rangle_k))$ is a *Hilbert space*. We apply the Gram-Schmidt process to the linearly independent system of polynomials $\{1, x, x^2, \ldots, x^m\}$ restricted to the interval $[x_{k-1}, x_{k+1}]$ and convert it to an orthogonal system in $L_2(A_k)$. The resulting orthogonal polynomials are denoted by $P_k^0, P_k^1, P_k^2, \ldots, P_k^m$.

Example 1. Below, we write the first three orthogonal polynomials P^0, P^1, P^2 in $L_2(A)$, where A is the generating function of a uniform fuzzy partition, and $\langle \cdot, \cdot \rangle_0$ is the inner product:

$$\begin{aligned} P^0(x) =& 1, \\ P^1(x) =& x, \\ P^2(x) =& x^2 - I_2, \text{ where } I_2 = h^2 \int_{-1}^{1} x^2 A(x)dx, \end{aligned}$$

If the generating function A^{tr} is triangular shaped and h-rescaled, then the polynomial P^2 can be simplified to the form

$$P^2(x) = x^2 - \frac{h^2}{6}. \tag{9}$$

We denote $L_2^m(A_k)$ a linear subspace of $L_2(A_k)$ with the basis $P_k^0, P_k^1, P_k^2 b, \ldots, P_k^m$.

4.3 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. Let us fix $[a, b]$ and its fuzzy partition $A_1, \ldots, A_n$, $n \geq 2$.

Definition 2. Let $f : [a, b] \to \mathbb{R}$ be a function from $L_2(A_1, \ldots, A_n)$, 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 $L_2^m(A_k)$, $k = 1, \ldots, n$. We say that the n-tuple $(F_1^m, \ldots, F_n^m)$ is an F^m-transform of

f with respect to $A_1, \ldots, A_n$, or formally,

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

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

$$F_k^m = c_{k,0}P_k^0 + c_{k,1}P_k^1 + \cdots + c_{k,m}P_k^m, \tag{10}$$

where

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

Definition 3. Let $F^m[f] = (F_1^m, \ldots, F_n^m)$ be the direct F^m-transform of f with respect to $A_1, \ldots, A_n$. Then the function

$$\hat{f}_n^m(x) = \sum_{k=1}^{n} F_k^m A_k(x), \quad x \in [a, b], \tag{11}$$

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

The following theorem proved in [89] estimates the quality of approximation by the inverse F^m-transform in a normed space L_1.

Theorem 1. Let $A_1, \ldots, A_n$ be an h-uniform fuzzy partition of $[a, b]$. Moreover, let the functions f and A_k, $k = 1, \ldots, n$ 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 $[a + h, b - h]$.

4.4 F^2-transform in the Convolutional Form

Let us fix $[a, b]$ and its h-uniform fuzzy partition $A_1, \ldots, A_n$, $n \geq 2$, 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

$$F^2[f] = (c_{1,0}P_1^0 + c_{1,1}P_1^1 + c_{1,2}P_1^2, \ldots, c_{n,0}P_n^0 + c_{n,1}P_n^1 + c_{n,2}P_n^2), \tag{12}$$

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

$$P_k^0(x) = 1,\ P_k^1(x) = x - x_k,\ P_k^2(x) = (x - x_k)^2 - I_2, \tag{13}$$

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

$$c_{k,0} = \frac{\int_{-\infty}^{\infty} f(x)A_h(x - x_k)dx}{\int_{-\infty}^{\infty} A_h(x - x_k)dx}, \tag{14}$$

$$c_{k,1} = \frac{\int_{-\infty}^{\infty} f(x)(x - x_k)A_h(x - x_k)dx}{\int_{-\infty}^{\infty} (x - x_k)^2 A_h(x - x_k)dx}, \tag{15}$$

$$c_{k,2} = \frac{\int_{-\infty}^{\infty} f(x)((x - x_k)^2 - I_2)A_h(x - x_k)dx}{\int_{-\infty}^{\infty} ((x - x_k)^2 - I_2)^2 A_h(x - x_k)dx}. \tag{16}$$

In [89, 93], it has been proved that

$$c_{k,0} \approx f(x_k),\ c_{k,1} \approx f'(x_k),\ c_{k,2} \approx f''(x_k), \tag{17}$$

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

Without going into technical details, we rewrite (14) - (16) into the following discrete representations

$$c_{k,0} = \sum_{j=1}^{l} f(j)g_0(ks - j),\ c_{k,1} = \sum_{j=1}^{l} f(j)g_1(ks - j),\ c_{k,2} = \sum_{j=1}^{l} f(j)g_2(ks - j), \tag{18}$$

where $k = 1, \ldots, n$, $n = \lfloor \frac{l}{s} \rfloor$, s is the so called stride and g_0, g_1, g_2 are normalized functions that respectively correspond to the generating functions A_h, (xA_h) and $((x^2 - I_2)A_h)$. It is easy to see that if $s = 1$, then the 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 (12) in the following vector form:

$$F^2[f] = ((f \star_s g_0)^T \mathbf{P}^0 + (f \star_s g_1)^T \mathbf{P}^1 + (f \star_s g_2)^T \mathbf{P}^2), \tag{19}$$

where $\mathbf{P}^0$, $\mathbf{P}^1$, $\mathbf{P}^2$ are vectors of polynomials with components given in (13), and $\star_s$ means that the convolution is performed with the stride s, $s \geq 1$.

4.5 F-transform convolution kernels and their semantic meaning

In the previous section, we have defined F-transform of various degrees in general. Let us inspect the convolutional F-transform kernels in the form of a plot. From Fig. 8, we can see that F-transform kernels have similar shapes to some of the standard convolutional kernels from the field of image processing. Namely, we can see a correspondence between F^0 and Gaussian convolutional kernel, F^1 and Sobel convolutional kernel, and F^2 and Laplacian convolutional kernels. While Gaussian and Laplacian have functional forms and, therefore, it is trivial to obtain convolutional kernels of different sizes, it is not the case for Sobel. Sobel operator computes the first partial derivations, using the central difference. Hence, increasing the size of the kernel is not common. F^1 is defined functionally, and while it shares similarities with Sobel, its convolutional kernel can be easily expanded. An example of F-transform kernels in their matrix form of size 5×5 is shown in equation (20). Note that the values are scaled for better readability.

$$F^0 = \begin{pmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{pmatrix}$$

$$F^1 = \begin{pmatrix} -1 & -1 & 0 & 1 & 1 \\ -2 & -2 & 0 & 2 & 2 \\ -3 & -3 & 0 & 3 & 3 \\ -2 & -2 & 0 & 2 & 2 \\ -1 & -1 & 0 & 1 & 1 \end{pmatrix}$$

$$F^2 = \begin{pmatrix} 5 & 4 & 3 & 4 & 5 \\ 4 & -4 & -12 & -4 & 4 \\ 3 & -12 & -27 & -12 & 3 \\ 4 & -4 & -12 & -4 & 4 \\ 5 & 4 & 3 & 4 & 5 \end{pmatrix} \tag{20}$$

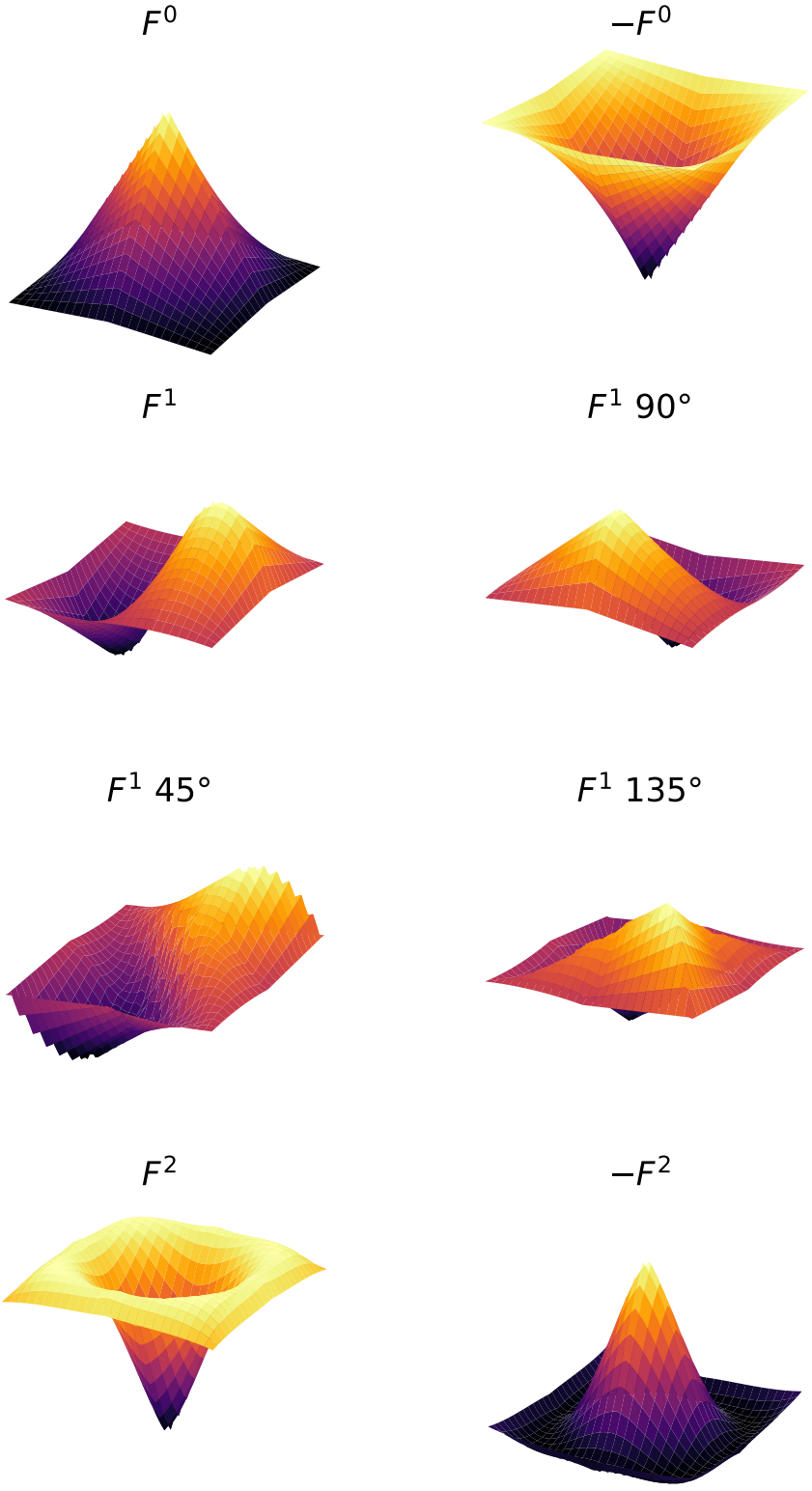


Figure 8: Visualization of F-transform kernels up to second degree and their rigid transformation.

5 Datasets

Data are an essential part of any data-driven technique. For the experiments in this work, several datasets were used. The datasets vary in colorfulness, number of the images, and image sizes. Each of the datasets is described below. The dataset summary is shown in Table 3.

MNIST [23] is a dataset created from the much larger NIST [39]. It contains 70 000 grayscale images (6 : 1 train-test ratio) of handwritten numbers $(0-9)$ that were scaled down to 28×28 pixels from the original 128×128 pixels and anti-aliased. MNIST is an older dataset that does not pose any challenge for modern classification solutions (any remaining error is due to incorrect labels). Due to its simplicity, it is used for proof of concept and prototyping.

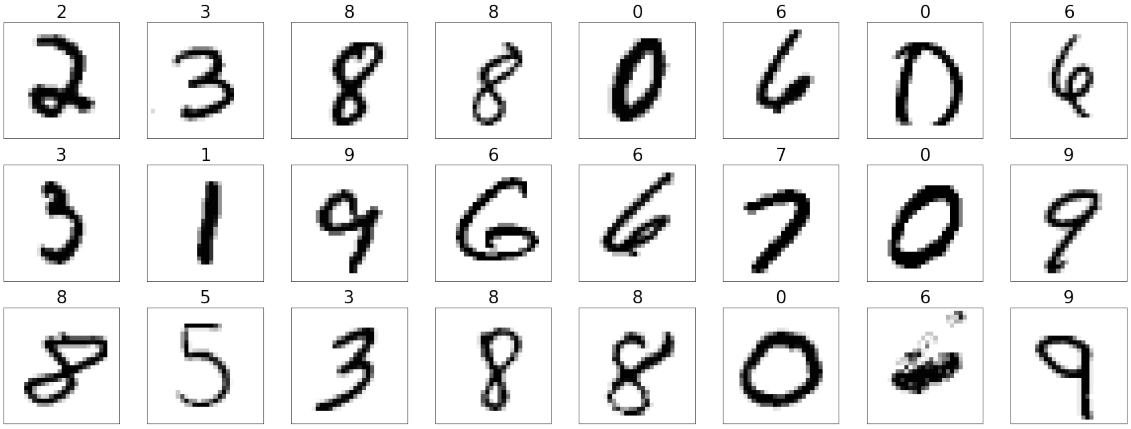


Figure 9: Random samples from MNIST dataset with corresponding labels.

CIFAR-10 [62], similarly to MNIST, is a subset of a much bigger dataset – 80 Million Tiny Images [120]. 80 Million Tiny Images was removed from the public domain in 2020 after some models trained on it exhibited racist and sexist biases. CIFAR-10 consists of 60 000 RGB images (5 : 1 train-test ratio) with resolution 32×32 pixels sorted into 10 classes, hence CIFAR-10. Although more complex than MNIST, according to website *papers with code*[9], the current state of the art reaches as high as 99.7% accuracy.

Caltech101 [28] is a dataset from 2003 divided into 101 classes with 40 to 800 images per class. The total size of the dataset is 9145 RGB images with a variable resolution. Aside from being divided into classes, Caltech 101 has segmentation annotations too.

Intel Scene Classification[10] challenge was a competition published at Ana-

[9] `https://paperswithcode.com/sota/image-classification-on-cifar-10`
[10] `https://www.kaggle.com/puneet6060/intel-image-classification`

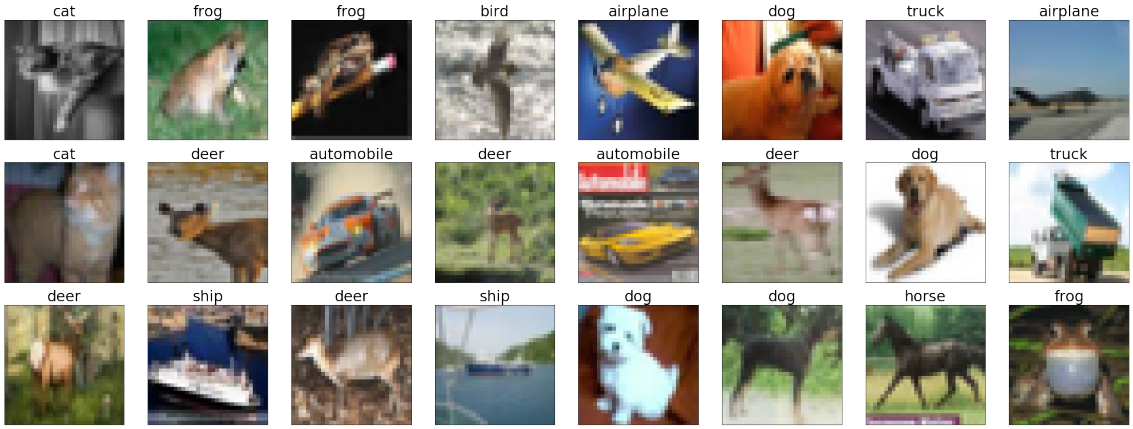


Figure 10: Random samples from CIFAR-10 dataset with corresponding labels.

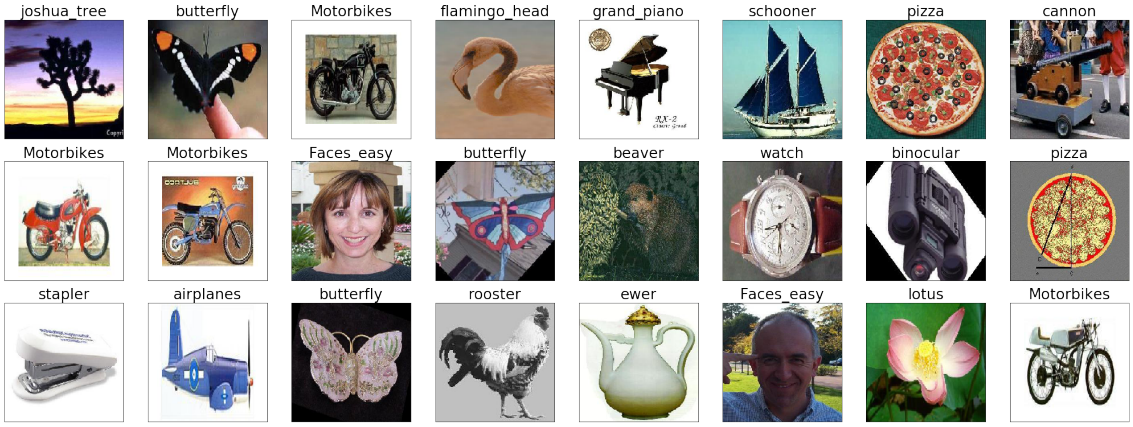


Figure 11: Random samples from Caltech101 dataset with corresponding labels.

lytics Vidhya hosted by Intel. Supplemented materials included a dataset of 14034 train and 3000 test RGB images with a resolution of 150×150 pixels sorted into six classes.

Figure 12: Random samples from Intel Scene Classification dataset with corresponding labels.

ImageNet [105] is a large database of images divided into thousands of classes. These classes are hierarchical, as shown in Fig. 13. ImageNet does not host the images; instead, it provides links to them. From year 2010 to 2017, ImageNet hosted a famous *ImageNet Large Scale Visual Recognition Challenge*. Each year, a subset of the ImageNet database was selected as a training and testing dataset. ILSVRC produced some of the most famous CNN architectures, such as AlexNet, ResNet, InceptionNet, and VGG. By the year 2017, ILSVRC stopped with 2D classification datasets and started to work on creating 3D datasets, as the results of ILSVRC 2017 crossed 96% accuracy (results available at ImageNet website[11]). The ImageNet was not used for training any of F-transform related networks; however, ImageNet pretrained networks are used in the kernel analysis further in the text. Unfortunately, the authors did not provide information about which year of the ILSVRC dataset was used for training.

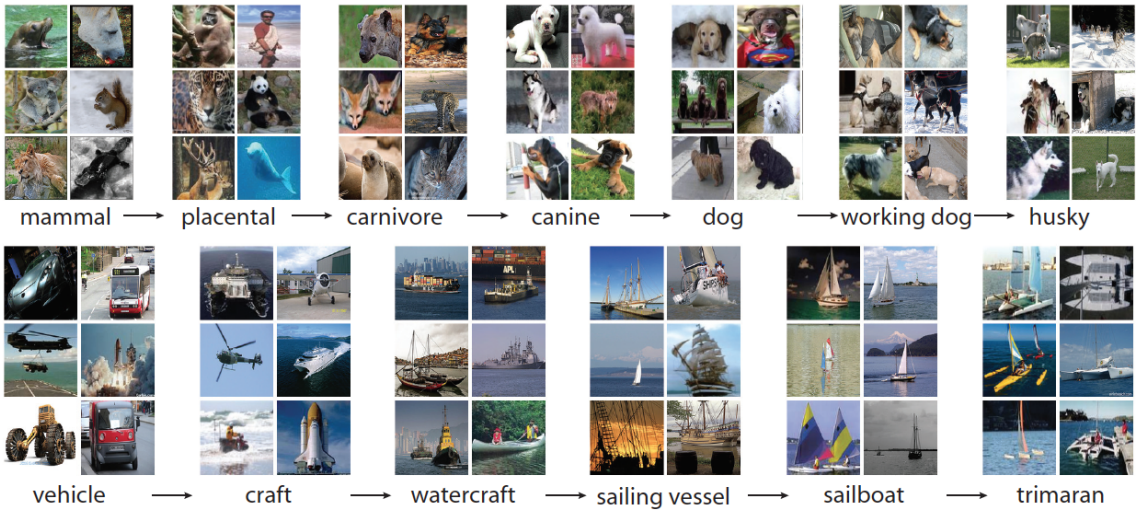


Figure 13: Random samples from ImageNet dataset with hierarchical categories. Image taken from [130].

Table 3: Datasets described in this section and their parameters.

	MNIST	CIFAR-10	Caltech101	Intel	ImageNet
Res.	28×28	32×32	*var*	150×150	224×224
Color	gray	RGB	RGB	RGB	RGB
Train	60k	50k	7281	14034	?
Test	10k	10k	1863	3000	?
Classes	10	10	101	6	1000

[11] `http://image-net.org/challenges/LSVRC/2017/results`

6 Convolutional neural networks and F-transform

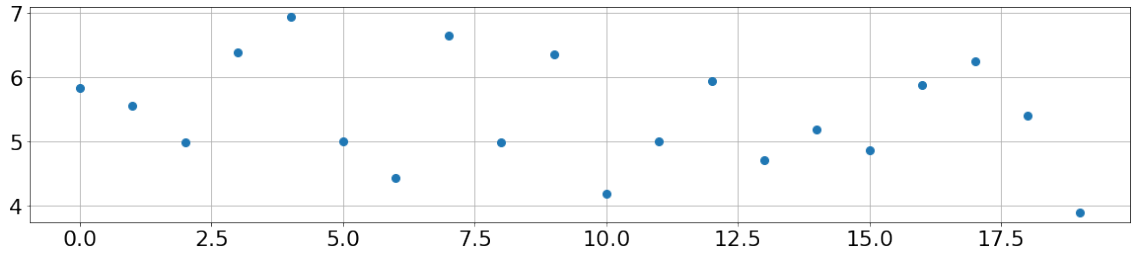
In section 4.1, we have introduced a fuzzy partition. Throughout section 4.4, we have shown how to use convolution to compute F-transform components that we treat as local image features. It has been shown (18) that F-transform components are the result of convolution with the kernel expressed from a generating function. This can be extended to the initialization schemes (section 2.7) of the convolutional layers – a kernel sampled from a distribution is a kernel potentially corresponding to a generating function of a partition. The kernel, of course, changes during the training; however, it is apparent that random initialization at the beginning creates diverse partitions for each filter with little to no semantic structure (in contrast to traditional operators such as Sobel operator [111] or Gabor filters [31]). The F-transform kernels (applied to a function of one variable) have the semantic (F^0 - weighted average, F^1 - first partial derivative, F^2 - second partial derivative) and the property of reconstructing the original function from components with arbitrary precision. Thus, the F-transform components are information-rich and suitable for image representation. Due to the mentioned semantic meaning of F-transform kernels, their sequential application performs mixed partial derivatives. This corresponds to initializing consequent convolutional layers with F-transform kernels (simplified view, where activation functions are not taken into account). This motivation is used to create a different kind of initialization based on F-transform and to install the F-transform initialized kernels in various known CNN architectures: LeNet5, ResNet, EfficientNet, and MobileNetV2. We expect our initialization to achieve higher classification accuracy, shorter training time, and better explainability after training.

6.1 FTNet

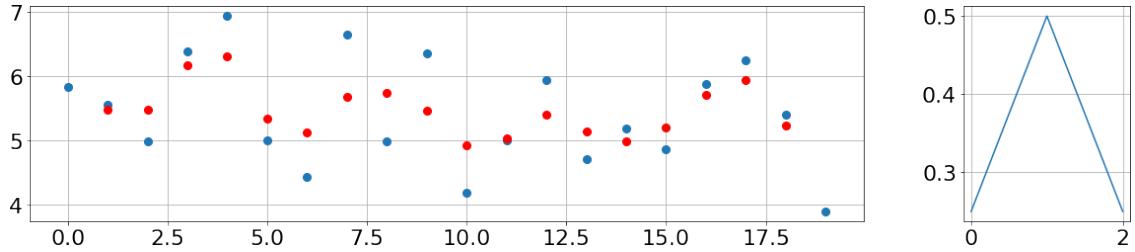
In this section, we introduce our CNN with F-transform kernels, dubbed **FTNet**. FTNet architecture is inspired by Yann Lecun LeNet-5 [67] (Fig. 15).

FTNet is a CNN with two blocks of convolutional and max pooling layers followed by two fully connected layers, where the second is the output. The convolutional layers and fully connected layers use the ReLU activation function; the output layer uses the softmax activation. The convolutional layers in the first and second block have their filters initialized with F-transform kernels. The architecture is summarized in Table 4 where the LeNet-5 naming scheme is used.

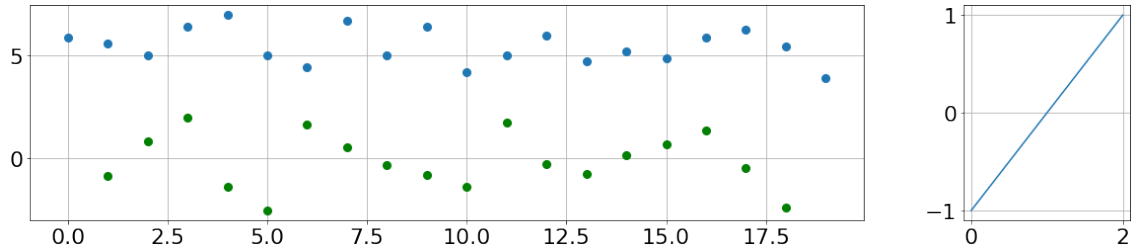
C_1 initialization is straightforward. Each of the 8 filters is initialized with one of the 8 F-transform kernels. C_3 is initialized so that every C_1 feature map is convolved with each of F-transform kernels.



(a) A function sampled from the normal distribution $\mathcal{N}(5, 1)$.



(b) The input function (blue) convolved with the triangular function shown on the right side. The result is in red.



(c) The input function (blue) convolved with a discrete differentiation operator shown on the right side. The result is in green.

Figure 14: Example of convolving function with two different kernels. First kernel is weighted average and second approximates gradient.

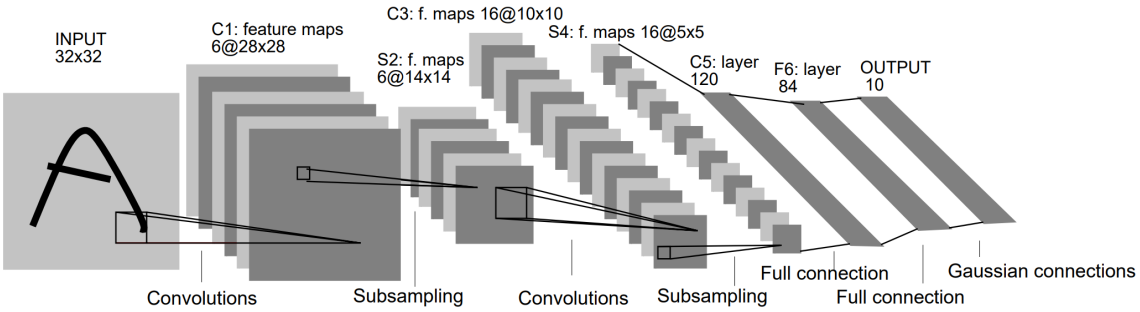


Figure 15: LeNet-5 architecture [67].

Fig. 16 shows the process of C_3 initialization. i-th slice of C_3 k-th filter $k = 8(i+0), \ldots, 8(i+7)$ are initialized with F-transform kernels, and the rest of the slices are initialized to 0. Thus, the filters process only i-th feature map, and the rest of the feature maps is zeroed during multiplication. The total number of C_3 filters is $8 \cdot 8 = 64$. This initialization scheme creates all possible combinations of

Table 4: FT-Net architecture.

Hyper-parameter	**Layers**					
	C_1	S_2	C_3	S_4	FC_5	FC_6
kernel size	5×5	-	5×5	-	-	-
# kernels	8	-	64	-	-	-
stride	1×1	2×2	1×1	2×2	-	-
pooling size	-	2×2	-	2×2	-	
# FC units	-	-	-	-	500	*var*

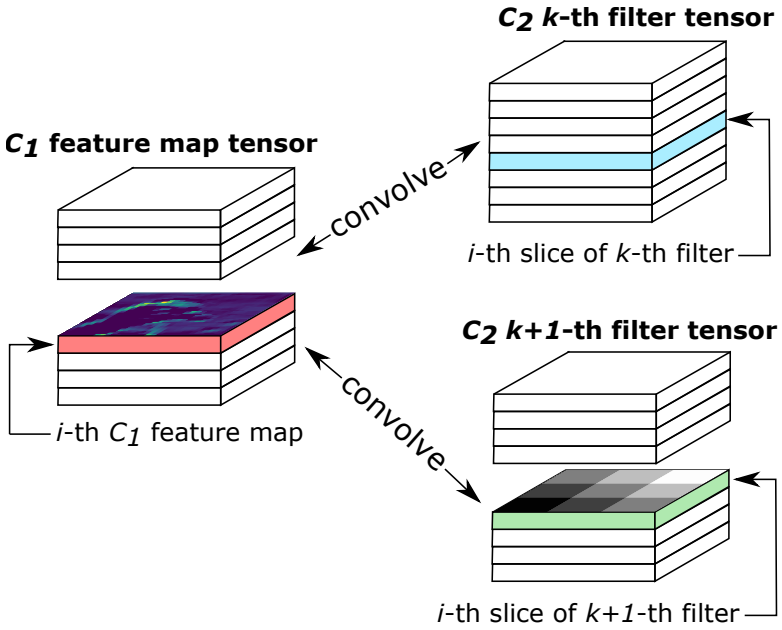


Figure 16: Illustration of FTNet C_3 initialization scheme.

F-transform kernels (including the mixed derivatives).

We will use FTNet architecture to test the hypothesis that F-transform kernel initialization yields better accuracy than He initialization (*baseline*). Let us describe the experiment setting more closely.

Datasets selected for experiments are MNIST, CIFAR-10, Caltech101, and Intel Scene Classification. The datasets images are converted to greyscale if needed, resized to 64×64 pixels, and normalized [83], and finally, each of the datasets is shuffled. Then, we train both FTNet, baseline and *static* FTNet **10 times**. Static FTNet C_1 and C_3 parameters (F-transform kernels) are not updated, thus staying identical throughout the training process.

The training is performed in two different initialization settings:

- Set_1 – FTNet and static FTNet C_1 are initialized with F-transform and C_3 with He initialization.

- Set_2 – FTNet and static FTNet C_1 and C_3 are initialized with F-transform kernels (C_3 as illustrated in Fig. 16).

The model test classification accuracy, as well as test loss value, are monitored during training. The models are trained for two epochs to analyze how the networks behave at the beginning of the training, and if using F-transform kernels helps the models to converge faster. For this purpose, we log said statistics after each *step*[12].

Training parameters are following: Adam optimizer with learning rate $\alpha = 1e-3$ and weight decay $\gamma = 1e-3$, cross entropy loss function, FC_5 and FC_6 with $L_2(\lambda = 1e-3)$ regularization and batch size = 50.

Finally, to match He initialization, we normalize and rescale F-transform kernels following He normal initialization formula.

Fig. 17 shows the normalized results of the training. From the graphs, we can see that **FTNet with F-transform kernels initialization reaches better accuracy and lower loss than He initialization.** *Static* FTNet that does not allow F-transform kernels to learn has lower loss values at the beginning of training but falls behind later. The advantage of static FTNet is higher training speed and fewer trainable parameters; this can be particularly beneficial to the overfitting problem and the problem of small datasets. Lastly, the static FTNet kernels and features they extract are clearly interpretable.

We compare FTNet with other approaches on MNIST, as it is the most common dataset. In Table 5 below, you can see a comparison of FTNet (C_1 and C_3 initialized with F-transform kernels) and the results reported in selected publications (denoted by their reference numbers in the first row). To fully train FTNet, we use Adam ($\alpha = 1e-4$), light data augmentation (rotation, shear, width/height change, zoom) [3]. The MNIST training data are split to train/validation with a 4:1 ratio.

Table 5: Comparison of FTNet with other approaches on MNIST. The most right-hand side result is FTNet initialized as described in 6.2.

FTNet	[11]	[29]	[6]	[95]	[129]	**FTNet** (sec. 6.2)
99.55%	99.39%	99.01%	99.52%	99.23%	99.58%	99.54%

Table 6 shows the average training times of the results shown in Fig. 17. The unusually long processing is caused by evaluating the testing data in each step. So the relevant information is the difference, especially between static FTNet and baseline. This confirms the advantage of FTNet – **decreasing trainable parameters by excluding layers from training, speeding up the training**, and in the case of

[12]One step is the processing of a single batch and followed by parameters update.

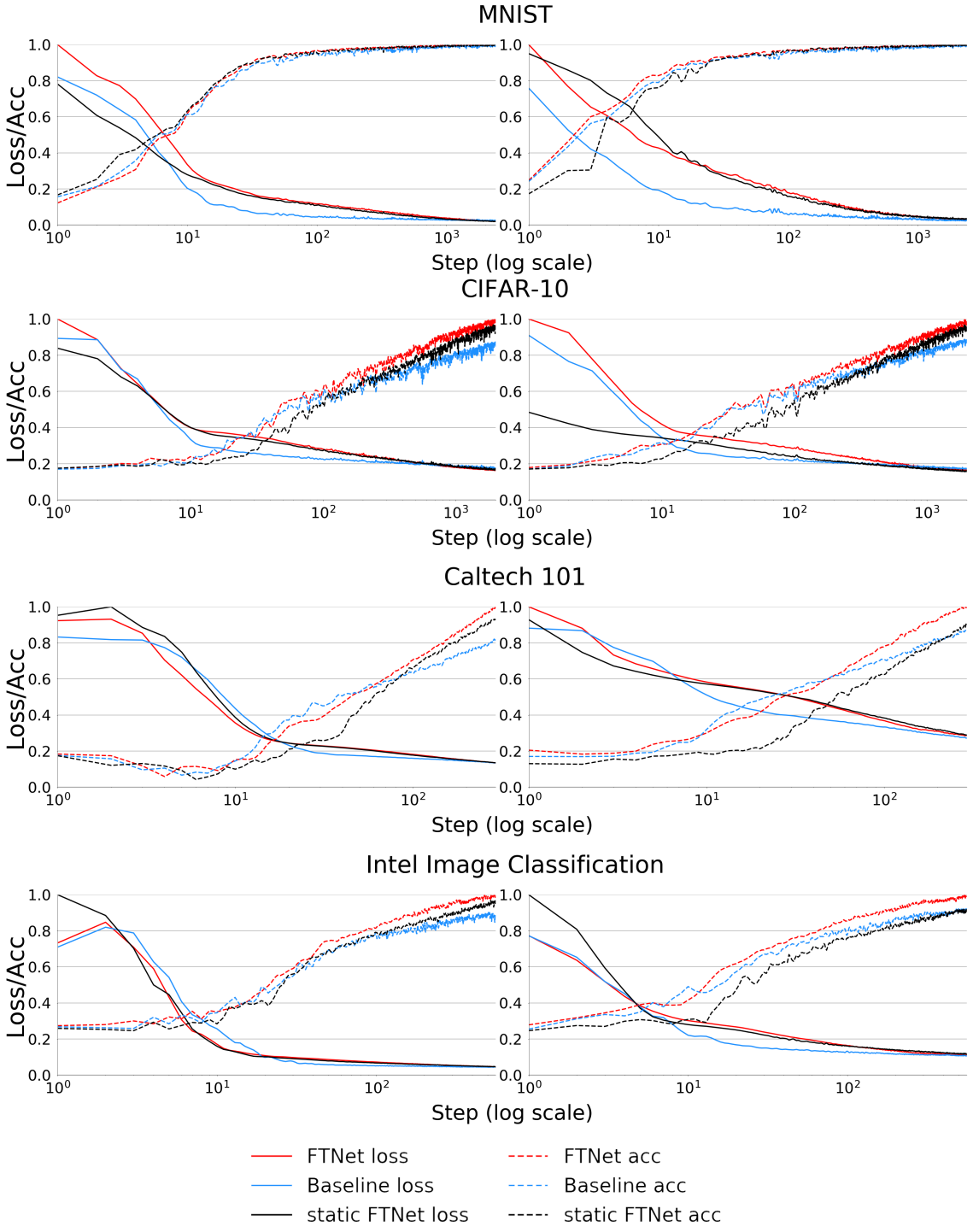


Figure 17: Loss and accuracy of models trained for 2 epochs. In the left column are results of training with FTNet C_1 initialized with F-transform kernels (Set_1). The second column contains results of training with both C_1 and C_3 initialized with F-transform kernels (Set_2). Note that both accuracy and loss are scaled to $[0, 1]$ and averaged over 10 runs.

Table 6: Average training time of FTNet architecture with F-transform kernels and He initialization (baseline).

Dataset	**FTNet** trainable	**FTNet** static	**Baseline**
MNIST	320s	299s	315s
CIFAR-10	290s	252s	289s
Caltech 101	20s	15s	20s
Intel	65s	53s	66s

FTNet preserving reasonable accuracy. Any divergence between trainable FTNet and baseline times can be attributed to the measuring error. Set_1 and Set_2 settings do not change the training time and are not distinguished in the table.

Let us comment on the relation between **explainability** and shorter training times. In 2019, Strubell et al. published [113] where the authors discussed the energy consumption of training deep neural networks and their impact on the environment (CO_2 emissions). Their work contains Table 7, which shows the amount of CO_2 produced by various activities, including the training of the deep neural network. From the table, we can see how demanding the training of deep neural networks is. Our approach to intelligent initialization with clearly explainable kernels helps to understand CNN and leads to shorter training times (Table 6), which contributes to less CO_2 emissions.

Table 7: Estimated CO_2 emissions from training common NLP models, compared to familiar consumption. This table is taken from [113].

Consumption	**CO_2 (lbs)**
Air travel, 1 passenger, NY $\leftrightarrow$ SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155

Section summary: In this section, we introduced the architecture **FTNet**. FTNet convolutional layers are specifically initialized with F-transform kernels. We showed a non-standard initialization scheme of C_3 layer in Fig. 16. The F-transform kernels initialization was compared with He initialization using FTNet and four

datasets: MNIST, CIFAR-10, Caltech 101, and Intel Image Classification. Using F-transform kernels initialization, we reached a lower loss, higher accuracy, and freezing layers led to faster training. Freezing is possible because F-transform kernels are already well developed and well-suited for feature extraction, which is not the case with random (He) initialization. FTNet result on MNIST dataset was compared with other fuzzy approaches where it achieved an excellent result. Lastly, we discussed the explainability and impact of deep learning on the environment. Lower training times decrease energy consumption and CO_2 emissions.

6.2 F-transform as initialization method for arbitrary CNN

In section 6.1 we used F-transform kernels to initialize the first two convolutional layers with the intent to convolve the input with F-transform kernels and then convolve each feature map again with all F-transform kernels. In this section, we will modify the initialization scheme illustrated in Fig. 16 by initializing the whole filters with F-transform kernels. We will demonstrate effectiveness on selected state-of-the-art CNNs.

Initializing the convolutional layer with F-transform is done by randomly selecting F-transform kernels and stacking them into a filter tensor; this procedure is done for all given convolutional layer filters. In the case of the first convolutional layer that takes as an input an image with three channels, the number of all possible non-duplicated filter combinations is $8^3 = 512$. Generally, we consider the filter tensor T valid iff $T_{*,*,*,j} \neq T_{*,*,*,k}$, where $j \neq k$. However, F-transform kernels can be duplicated within a single filter: $T_{*,*,i,j} = T_{*,*,k,j}$, where $i \neq k$. Therefore F-transform kernels can repeat within a single filter tensor (two different input channels can be convolved with the same kernel), but two different filters within the same layer cannot be identical. In general, the number of unique filters is 8^n, n is the number of input channels, so running out of combinations is not an issue. A special case, where the filter is the size of 1×1 is solved by using He initialization instead, as a single value does not carry semantic information as the F-transform kernels do.

We compare F-transform kernels with one of the most common initializations – He initialization. For this purpose, we train EfficientNet[13] [118] and ResNet[14] with different depth and on CIFAR-10. Let us describe both networks and training setup.

Preprocessing and augmentation is done by normalizing and subtracting

[13]Source code adapted from `https://github.com/qubvel/efficientnet`

[14]Source code adapted from `https://github.com/keras-team/keras/blob/master/examples/cifar10_resnet.py`

the mean image. Images are randomly horizontally flipped with a probability of 50%, and their width and height is randomly changed by a maximum of 10% during the training.

ResNet is trained in 3 depth variants: ResNet-20, ResNet-32 and ResNet-44. The network is trained for 200 epochs with batch size 50 in 20 separated runs. At the end of each epoch, we log the test loss and classification accuracy. The network uses Adam for optimization with a learning rate scheduler:

Table 8: ResNets learning rate scheduler. Baseline learning rate $\alpha = 1e-3$.

Epochs ep	$0 \leq ep < 80$	$80 \leq ep < 120$	$160 \leq ep < 180$	$180 \leq ep < 200$
Learning rate	$\alpha 1e-1$	$\alpha 1e-2$	$\alpha 1e-3$	$\alpha 0.5e-3$

EfficientNet variant B0 is trained for 50 epochs with batch size 20 in 20 separated runs. Due to architecture, the images are upscaled to resolution 128×128 pixels. We again log the test loss and classification accuracy at the end of each epoch. The network uses Adam for optimization with learning rate scheduler:

Table 9: EfficientNet learning rate scheduler. Baseline learning rate $\alpha = 1e-3$.

Epochs ep	$0 \leq ep < 20$	$20 \leq ep < 30$	$30 \leq ep < 40$	$45 \leq ep < 50$
Learning rate	$\alpha 1e-1$	$\alpha 1e-2$	$\alpha 1e-3$	$\alpha 0.5e-3$

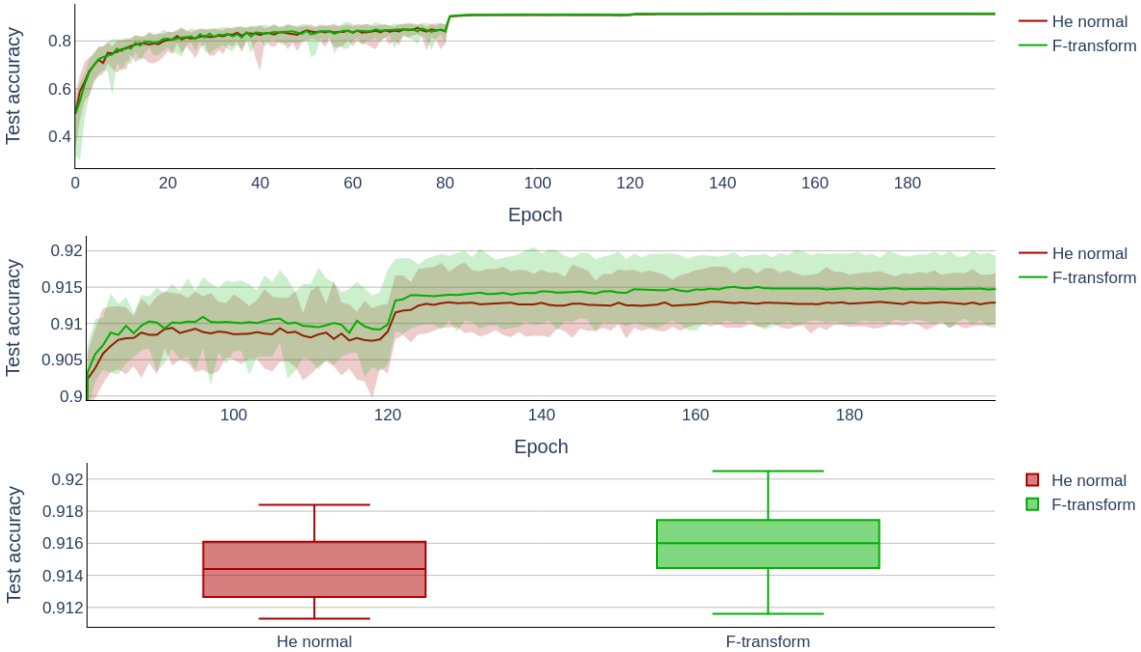


Figure 18: ResNet20, t-test $p = 0.041$.

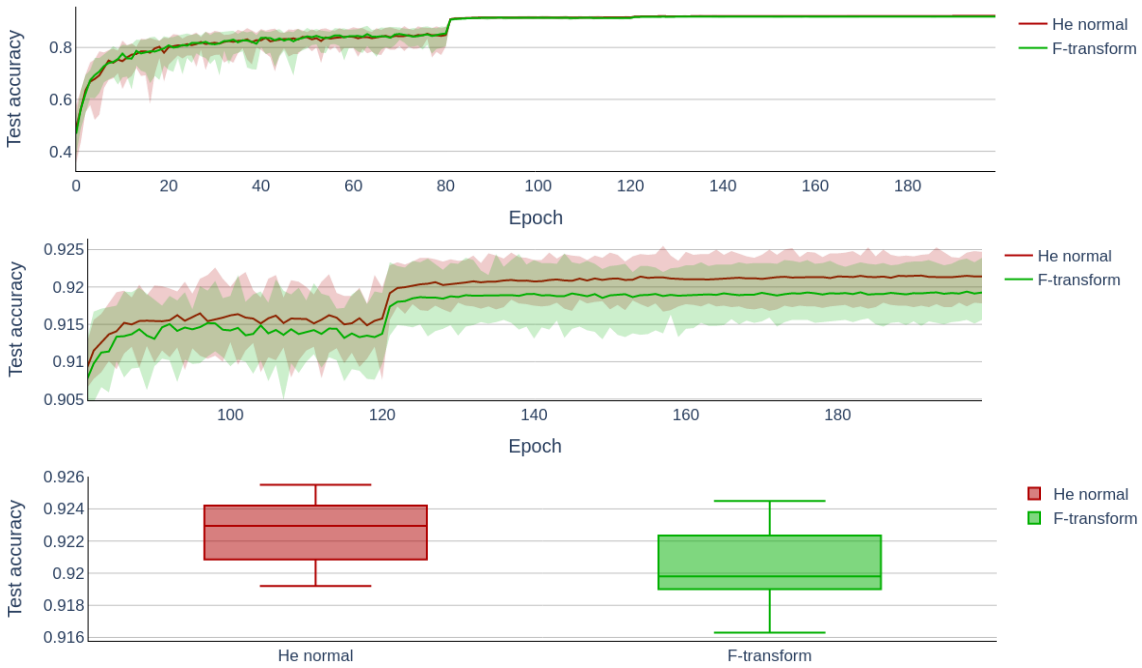


Figure 19: ResNet32, t-test $p = 0.004$.

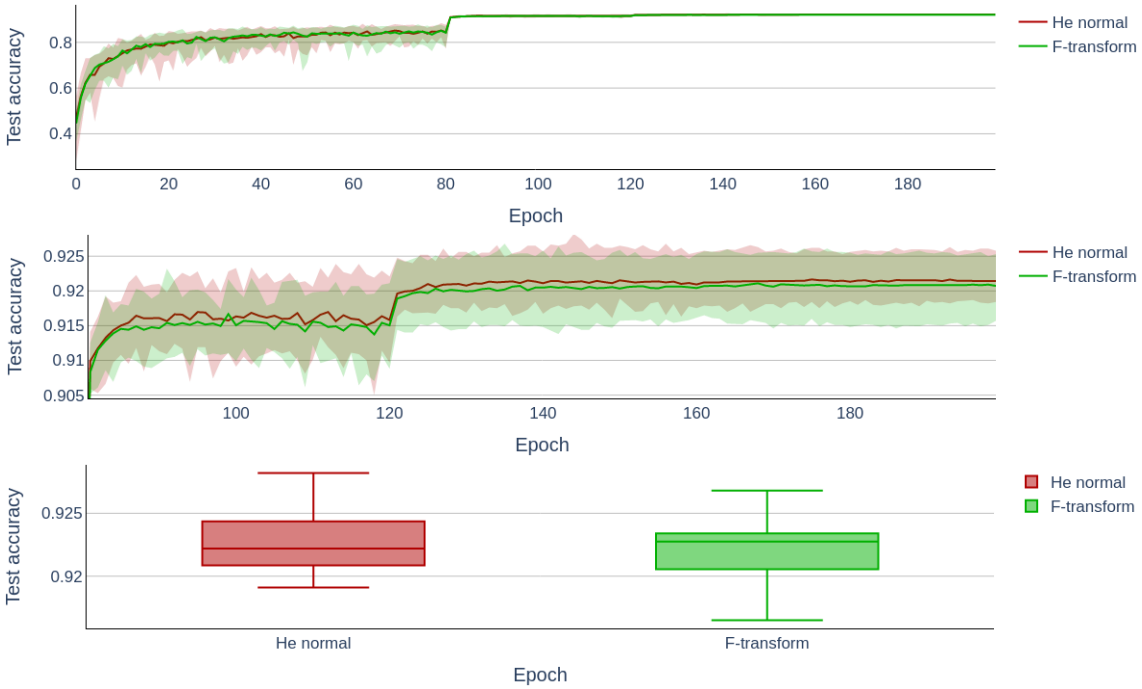


Figure 20: ResNet44, t-test $p = 0.427$.

In Figures 18 to 21, we see the results of training both ResNet and EfficientNet. The second graph in each figure shows the progression of training towards the end when the learning rate is lowered. The curves in the graphs are averages in each epoch. The filled area around is between minimums and maximums. Boxplots are

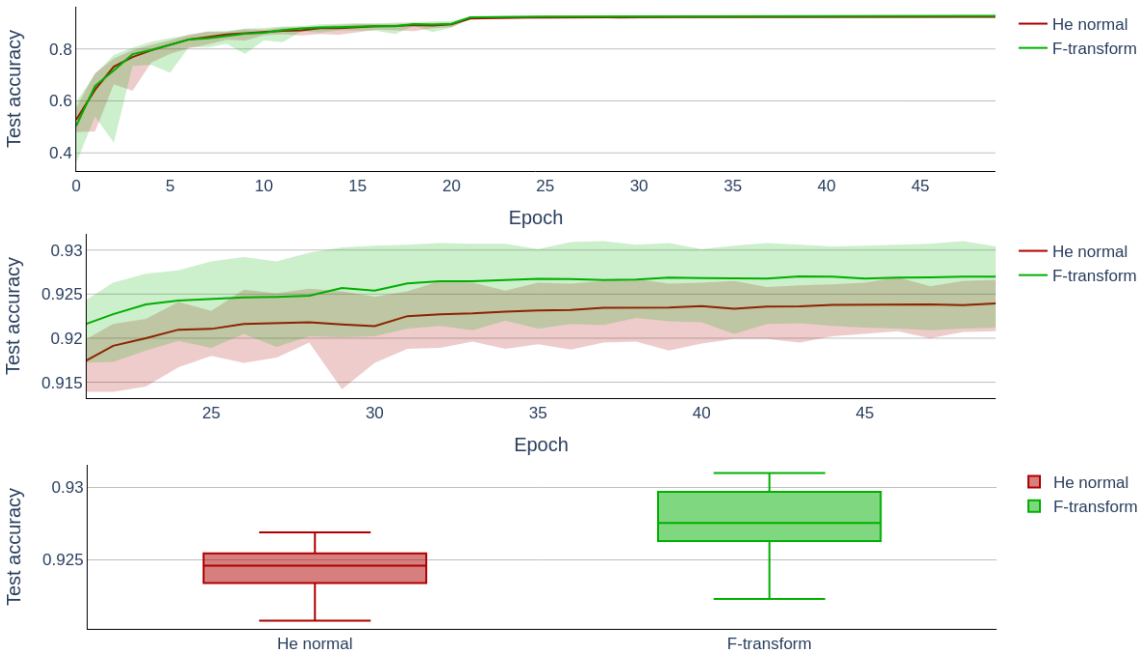


Figure 21: EfficientNetB0, t-test $p = 1.16e - 5$.

created from the best test accuracy in each of the 20 runs. We also t-test the best accuracies of F-transform and He normal initialization to see whenever the results have the same means, and the difference is statistically insignificant. This is the case only for ResNet44 in Figure 20. From the results, we can conclude that F-transform initialization works very well for smaller networks, and as the network gets deeper, it becomes comparable with He initializer.

Section summary: In this section, we modified F-transform kernels initialization to create convolutional layer filters by randomly picking from F-transform kernels to assemble them. We impose a condition that randomly picked filters cannot be duplicated in the same layer. We applied this initialization to the ResNet network of different depths, 20, 32, and 44, and the EfficientNetB0 network. When compared with He initialization, F-transform kernels achieve better results on the ResNet of a lower depth. The main contribution is the adaptation of our initialization scheme from section 6.1 to initialize almost arbitrary convolutional layer filters. Our imposed restriction is $8^n > m$, where n is the number of input channels and m is the number of layer filters.

6.3 Semantic meaning of principal kernels in convolutional layers

In this section, the first convolutional layer kernels are analyzed. The kernels are collected from multiple CNNs. The section is divided into kernel cluster analysis and individual kernel examination. The purpose is to find general semantic meanings of kernels and establish possible matching with F-transform kernels.

For the cluster analysis, the following networks were selected: VGG16[110], VGG19[110], InceptionV3[117], MobileNet[48], ResNet[41], and AlexNet[63]. The networks have their weights after training on ImageNet[105] publicly available and were used as data for cluster analysis.

We start with extracting the first convolutional layer weights for all networks. Then, for each extracted set of kernels, hierarchical clustering is performed. The results of hierarchical clustering are *medoids* of the found clusters. These medoids represent a given cluster and are shown in Fig. 22.

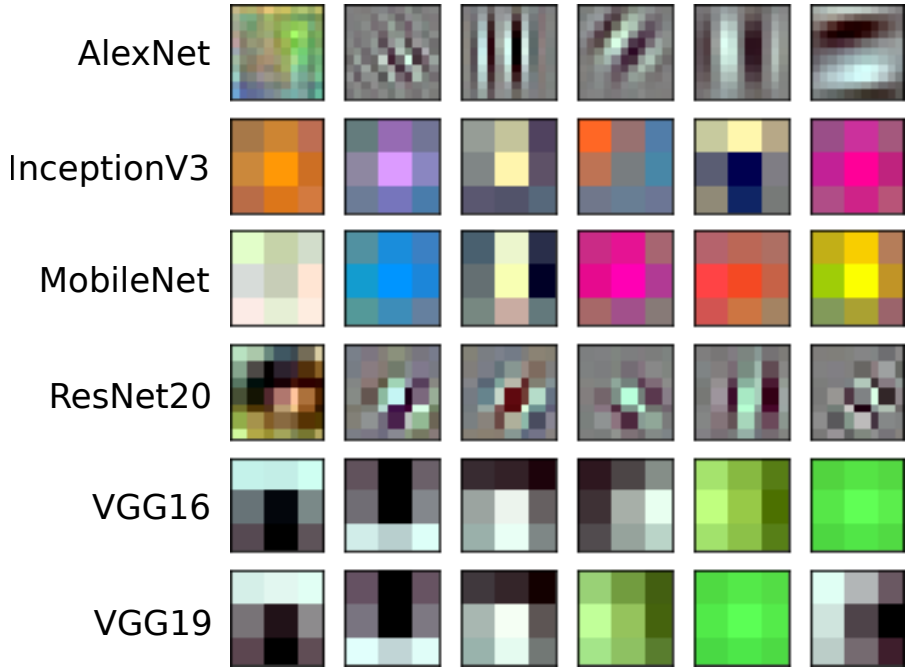


Figure 22: The medoids of the clustered kernels from the first convolutional layer of AlexNet, InceptionV3, MobileNet, ResNet, VGG16 and VGG19.

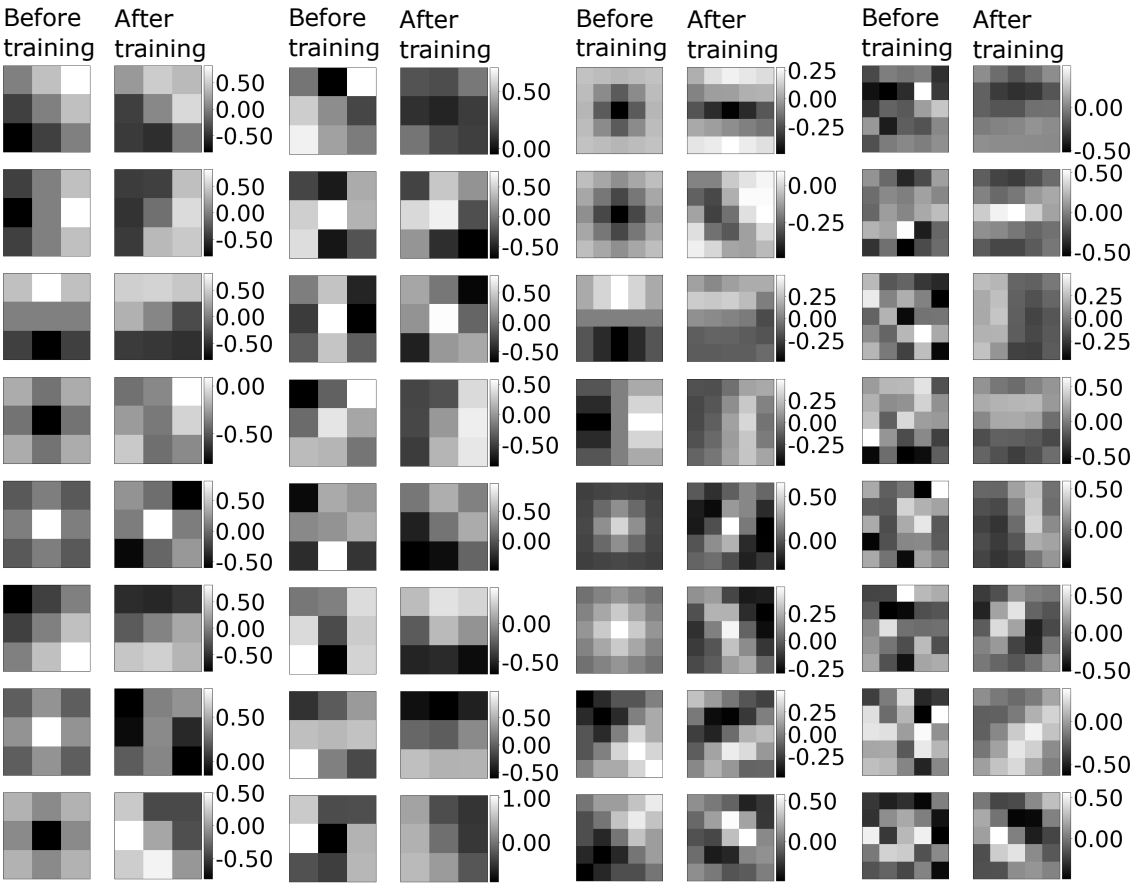
From Fig. 22, one can observe that the extracted clusters contain similar elements (kernels) across different networks that share one of the following characteristics/functionalities:

- **Gaussian-like** - filters used for blurring (InceptionV3 sixth kernel, VGG16 sixth kernel, MobileNet second kernel),

- **edge detectors** - kernels have different orientations that correspond to edges under different angles (VGG16 fourth kernel, AlexNet sixth kernel),
- **texture detectors** - kernels similar to Gabor filters (AlexNet second, third, and fourth kernel, ResNet second to sixth kernels),

The same properties can be found in F-transform kernels, the F^0-transform kernel is Gaussian-like, and the F^1-transform kernels are (horizontal or vertical) edge detectors. This similarity adds another indication to why F-transform kernels are suitable as a form of initialization. The ability to describe the semantic of kernels helps with the interpretability of CNNs.

In the following text, we examine the first convolutional layer kernels before and after training. Networks selected for analysis are from sections 6.1 and 6.2. We follow the training procedure from 6.2, except FTNet does not have a learning rate scheduler. Moreover, we add one more network, MobileNetV2 [107], as a representative for networks created for mobile devices. MobileNetV2 follows the same training procedure as the ResNet (section 6.2). We save the kernels of the respective networks before and after training, then examine the changes in their functionality during training. The comparison is twofold – visual comparison and statistical. We hypothesize that F-transform kernels should not change drastically; they should be stable.



(a) F-transform kernels initialization. (b) He normal initialization. (c) F-transform kernels initialization. (d) He normal initialization.

Figure 23: FtNet C_1 kernels before (left column) and after (right column) learning. 23(a) and 23(b) are 3×3 kernels, 23(c) and 23(d) are 5×5 kernels. The rows are sorted with respect to the sum of absolute element-wise differences between before, and after training.

Fig. 23 shows the change of FTNet C_1 kernels for kernel sizes 3 and 5. The 3×3 F^0 and F^2 kernels after training changed towards edge detection (23(a) rows 4, 7, and 8), while F^1, edge detector, stayed the same or similar. The presence of edge detectors is expected, as FTNet is trained on MNIST which consists of strokes forming numbers. There are not any complex structures or textures in the images. This effect is even more evident in Fig. 23(c). To compare F-transform kernels and He initialization kernels after training, we can pick similar kernels and measure their absolute distance from the initial state. For example, Fig. 23(a) the first row has 1,42 distance versus Fig. 23(b) the fifth row has 3,47 distance or the sixth row with 4,1 distance. These examples show how F-transform initialization starts closer to the resulting kernels.

Note that FTNet used to produce Fig. 23 kernels was initialized according to

section 6.1.

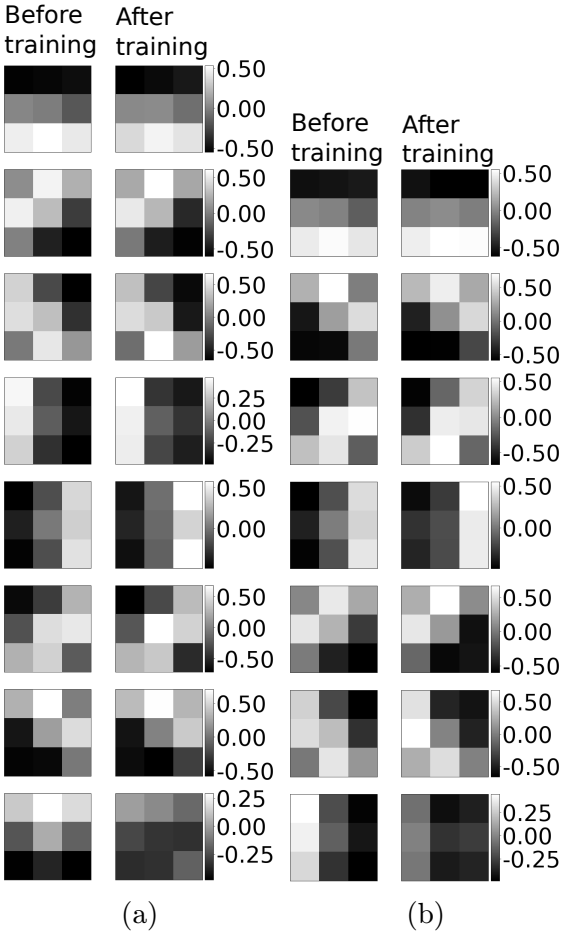


Figure 24: FTNet C_1 kernels before (left column) and after (right column) learning. C_3 is initialized according to Fig. 16. In (a) are pre-trained kernels (left column) and pre-trained kernels after training. We can see that F-transform kernels kept the shapes after training even when the rest of FTNet was re-initialized. The only outlier is the last row kernel. In (b), we removed the last row and retrained FTNet again; the kernels stay the same (except for scale). The rows are sorted with respect to the sum of absolute element-wise differences between before, and after training.

After examining Fig. 23 and concluding that F^1 kernels are prevalent, we trained FTNet where F^0 and F^2 were removed. After the training, we took these modified kernels and used them to initialize and train FTNet again. In Fig. 24(a) we can see that only the last kernel has changed. After removing this kernel and retraining again, Fig. 24(b), the kernels changed only slightly.

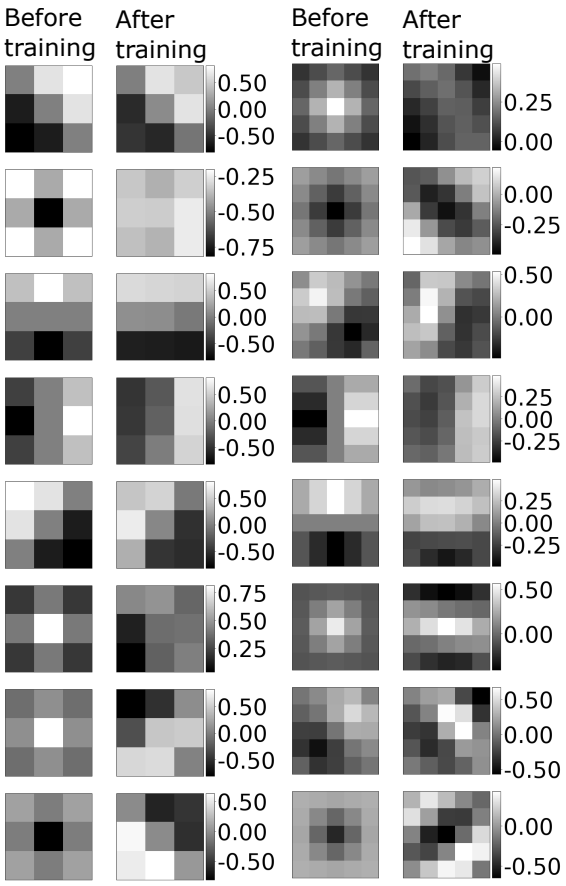
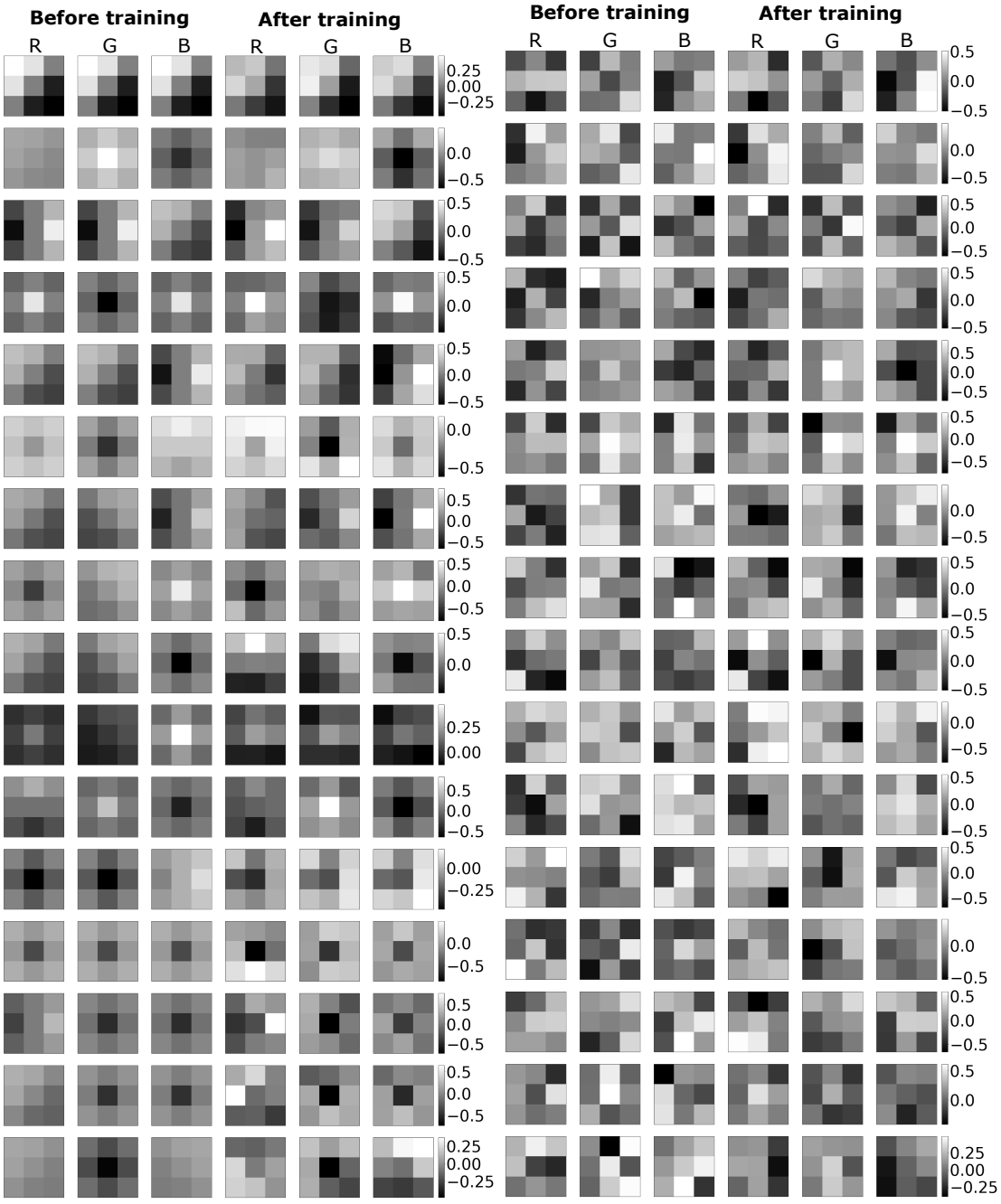


Figure 25: FTNet C_1 kernels before (left column) and after (right column) learning. C_3 is initialized entirely with F-transform kernels without zeroed kernels (section 6.2). The rows are sorted with respect to the sum of absolute element-wise differences between before, and after training.

Fig. 25 contains F-transform kernels in C_1 layer for kernel sizes 3 and 5. C_3 was initialized as described in section 6.2, i.e., without zeroed kernels (unlike Fig. 23). Fig. 25 displays the same trends as Fig. 23. Therefore different C_3 initialization schemes, in this case, are without any noticeable impact on C_1.



(a) F-transform kernels initialization. (b) He normal initialization.

Figure 26: ResNet20 first convolutional layer kernels before (first three columns) and after (last three columns) learning. Note that three columns visualization corresponds to RGB channels in the respective order. The rows are sorted with respect to the sum of absolute element-wise differences between before, and after training.

FTNet in previous figures was trained on greyscale MNIST images. ResNet20 kernels shown in Fig. 26 are from training ResNet20 on CIFAR-10, thus have 3 columns corresponding to RGB channels. Generally, these kernels are harder to interpret due to convolution not being applied separately per channel (depth-wise

convolution), but the results from convolving each channel with its respective part of the kernel are summed. F-transform kernels after the training appear to retain functionality, especially the edge detection (26(a) row 1, 3, 5, and 9). Moreover, Gaussian/Laplacian-shaped kernels are prevalent in the green and blue channels (26(a) rows 4, 6, 8, and 10).

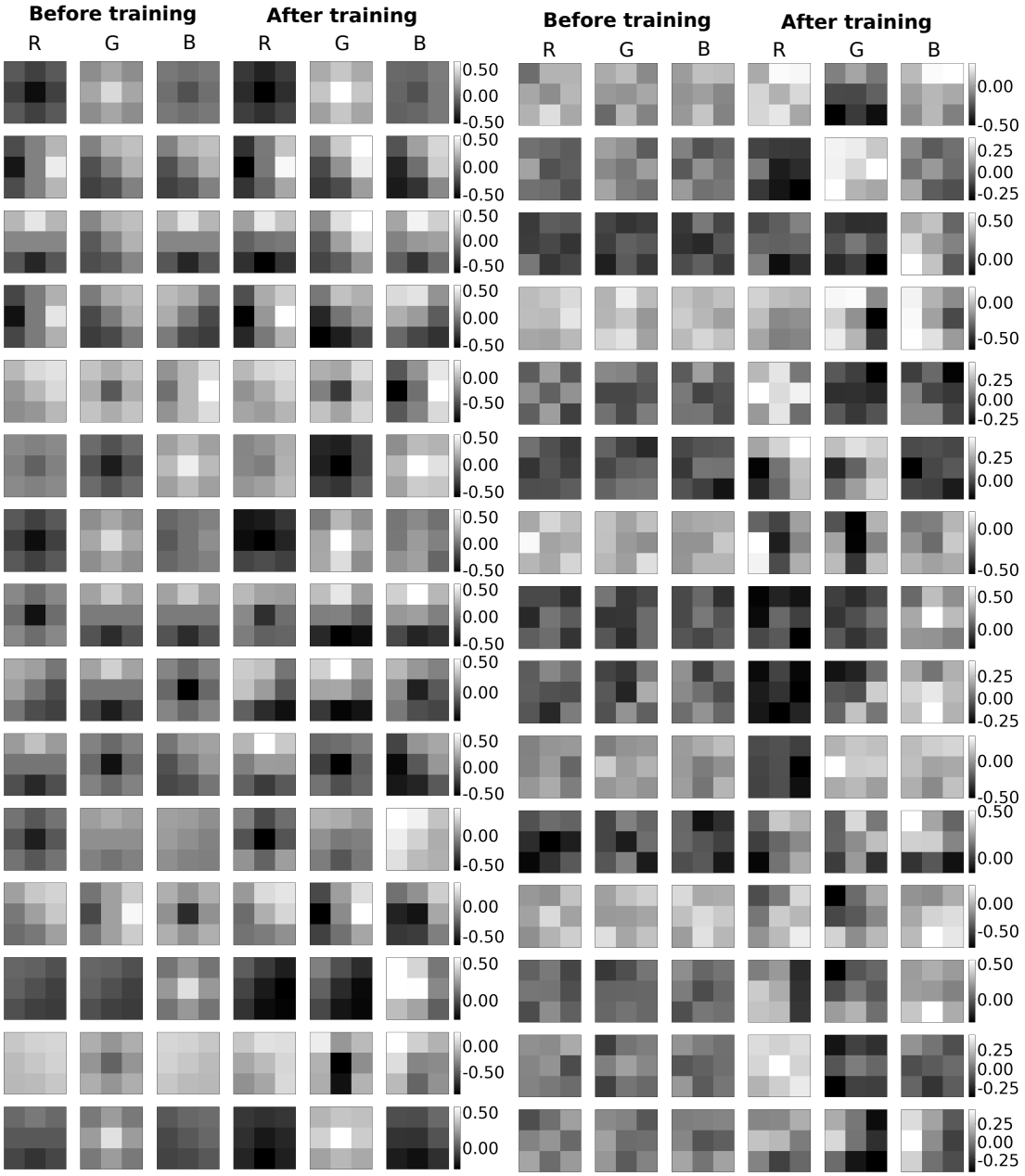
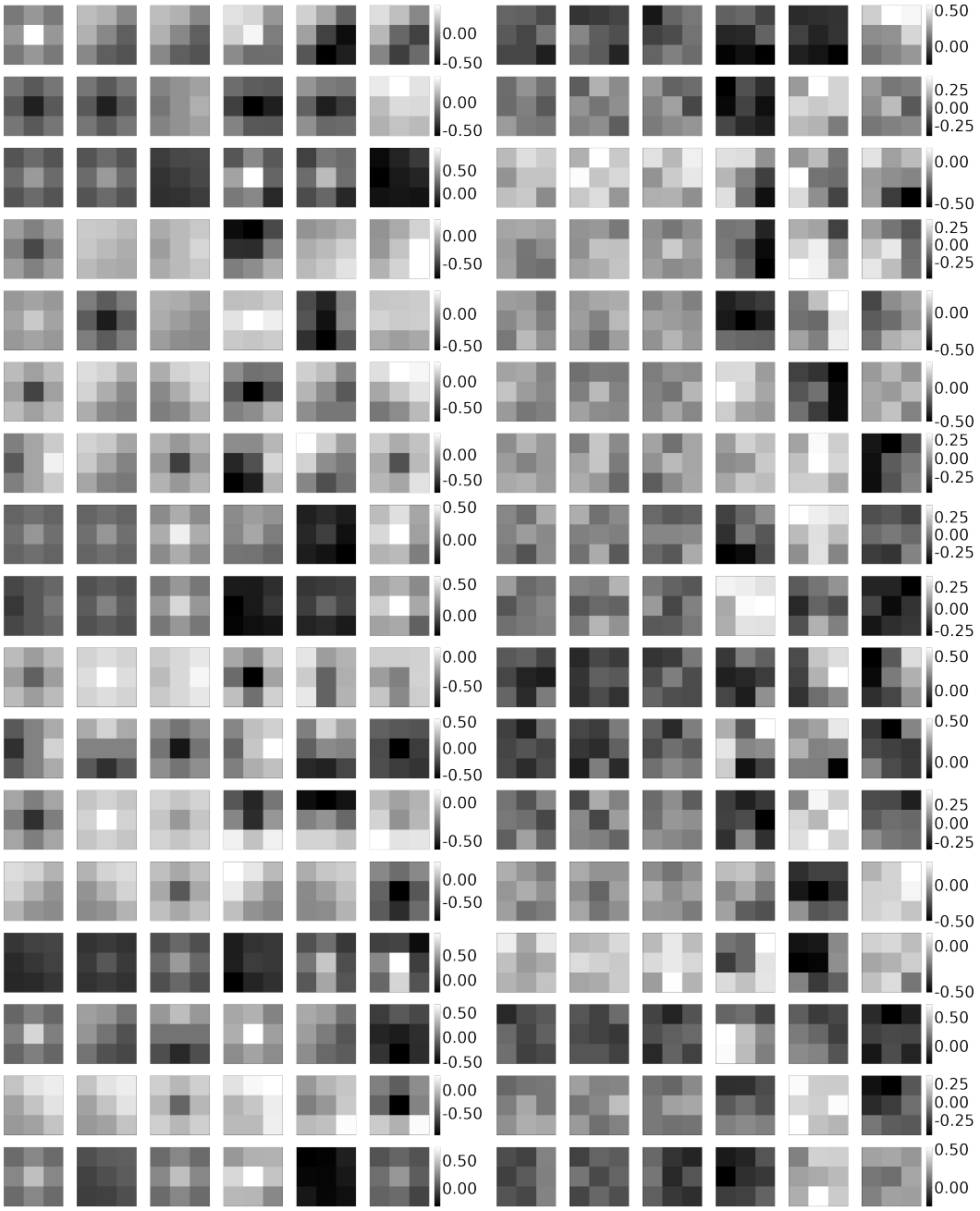


Figure 27: EfficientNetB0 first convolutional layer kernels before (first three columns) and after (last three columns) learning. Note that three columns visualization corresponds to RGB channels in that order. The rows are sorted with respect to the sum of absolute element-wise differences between before, and after training. The figure continues on the next page.



(a) F-transform kernels. (b) He initialization.

Figure 27

EfficientNetB0 is one of the most recent CNN. We trained EfficientNetB0 on CIFAR-10, and its kernels are shown in Fig. 27. In the figure, even the kernels at the bottom with a higher difference between before and after training mostly keep their function. Notably, the kernels after training have higher contrast. For example, the first six rows keep their edge detection and Gaussian/Laplacian shapes. The last row in the figure with the highest difference still retains Gaussian shape in the first and the third channel.

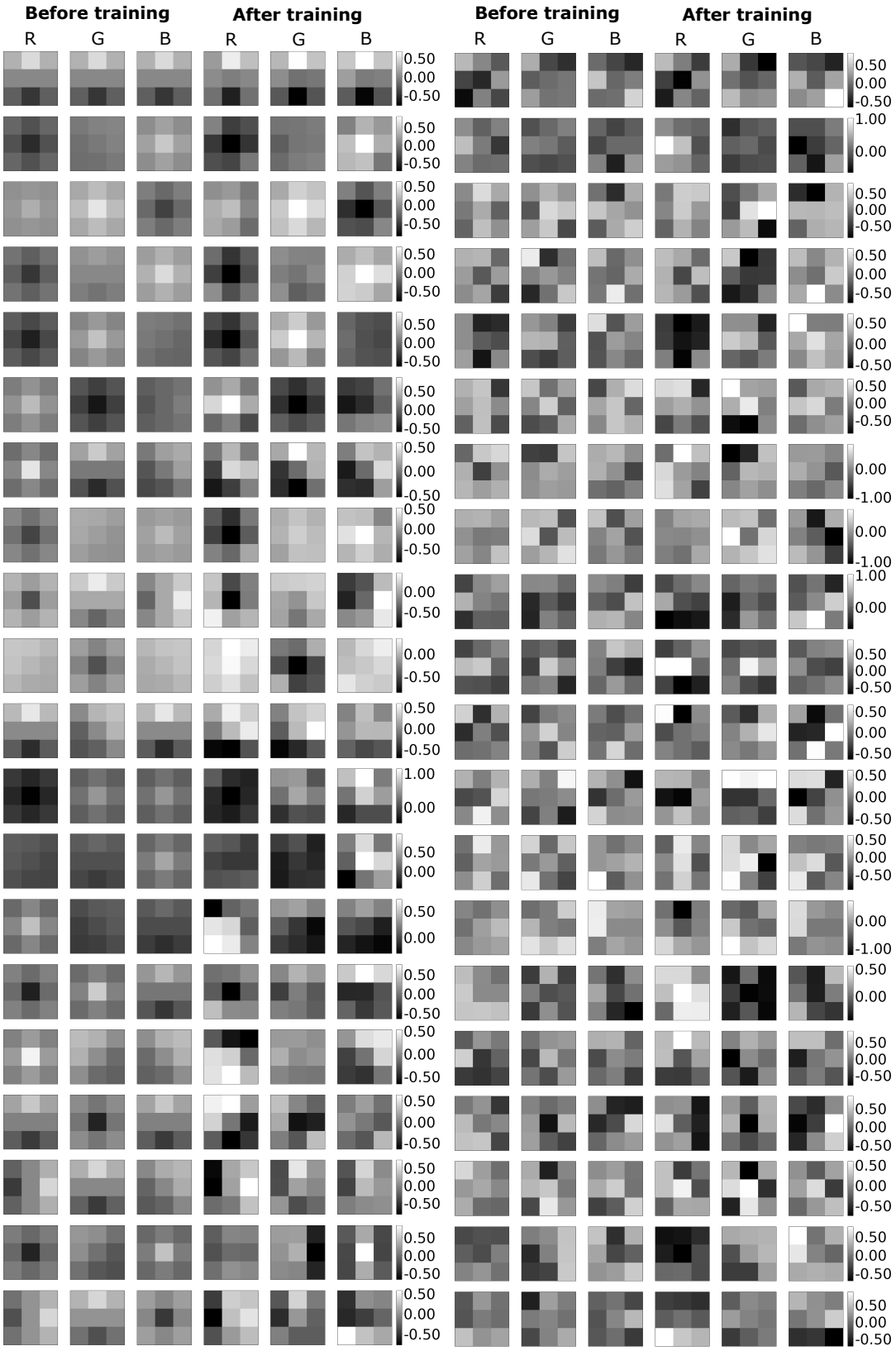
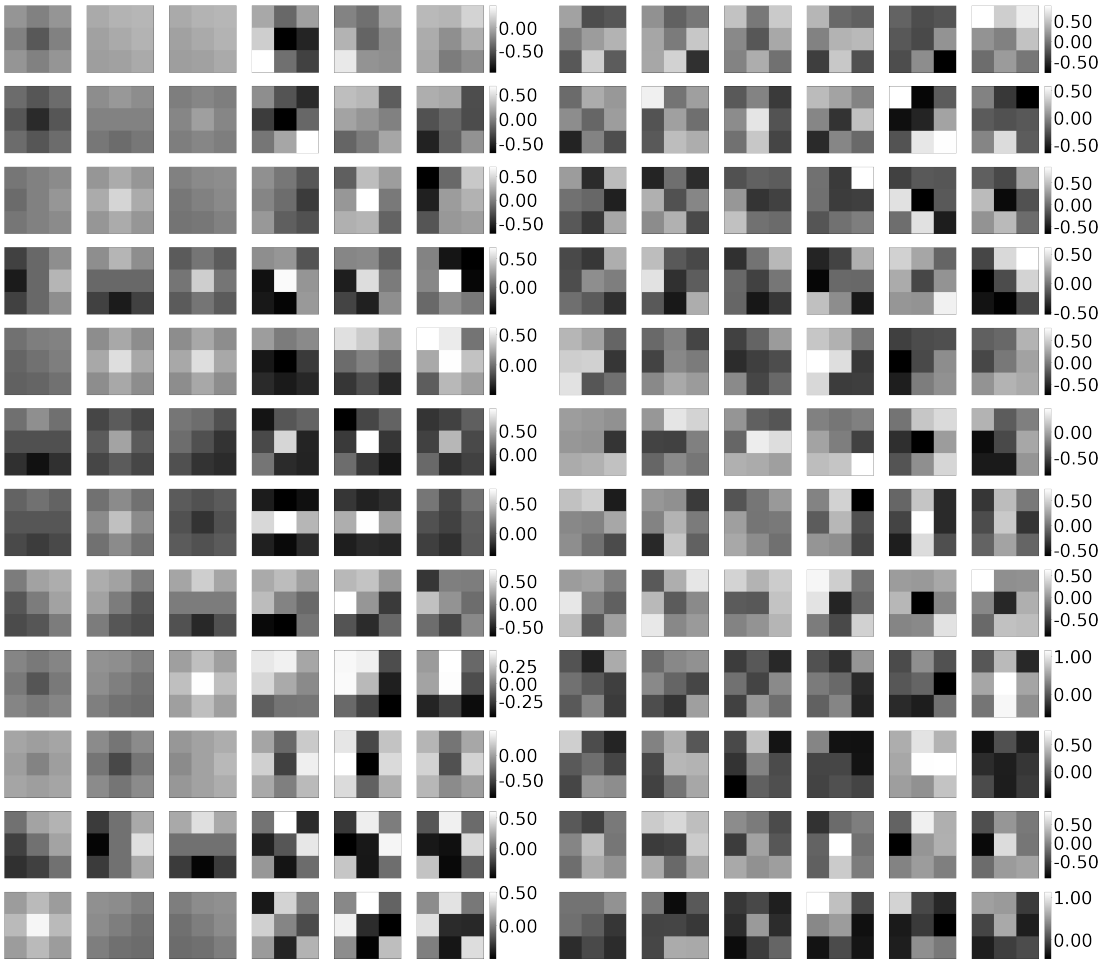


Figure 28: MobileNetV2 first convolutional layer kernels before (first three columns) and after (last three columns) learning. Note that three columns visualization corresponds to RGB channels in that order. The rows are sorted with respect to the sum of absolute element-wise differences between before, and after training. The figure continues on the next page.



(a) F-transform kernels. (b) He initialization.

Figure 28

MobileNetV2 is a network that was designed while keeping mobile device capabilities in mind. Out of all kernels analyzed in this section, MobileNetV2 kernels vary the most. While the top rows of Fig. 28 are almost identical, except for increasing contrast, the bottom rows show significant changes. Considering these changes, the majority of the kernels still contain edge detectors (rows 29, 23, or 1) and Gaussians/Laplacians (for example, rows 2, 3, 4, and 5).

Let us discuss the kernels, shown in this section, in terms of statistical characteristics, shown in Table 10. Even though the most important characteristic is the semantic of the kernels, He initialization relies heavily on the statistic, and we apply the same to F-transform kernels. While it is impossible to draw a conclusion that would apply to all kernels, the table results follow specific patterns. The average, minimum, mean, max, and std values after training are identical for F-transform kernels and He initialization for FTNet and ResNet20. The average mean values tend to converge towards zero. The minimum and maximum of FTNet and

Table 10: Average absolute change of kernel statistics before and after training. Each value is calculated as an absolute difference of the given statistic, averaging over all kernels.

Kernels	**min** before/after	**mean** before/after	**max** before/after	**std** before/after
Fig. 23(a)	-0.5996/-0.4614	0.0000/-0.0060	0.5996/0.5019	0.3510/0.3501
Fig. 23(b)	-0.5557/-0.4990	0.1347/0.0260	0.7865/0.5207	0.4143/0.3537
Fig. 23(c)	-0.3720/-0.3406	0.0000/-0.0080	0.3720/0.3887	0.2057/0.1946
Fig. 23(d)	-0.5310/-0.3111	-0.0141/-0.0087	0.5315/0.3716	0.2698/0.1938
Fig. 25(a)	-0.5996/-0.5005	0.0000/0.0168	0.5996/0.4984	0.3759/0.3707
Fig. 25(b)	-0.3720/-0.3862	-0.0002/0.0011	0.3720/0.4174	0.1962/0.2262
Fig. 26(a)	-0.4463/-0.5679	-0.0207/-0.0027	0.3363/0.4988	0.1799/0.2361
Fig. 26(b)	-0.5041/-0.5650	0.0137/-0.0004	0.4980/0.4883	0.2700/0.2576
Fig. 27(a)	-0.3732/-0.5875	-0.0015/-0.0291	0.3460/0.5081	0.1594/0.2740
Fig. 27(b)	-0.1491/-0.3986	0.0013/0.0088	0.1586/0.4150	0.0811/0.2159
Fig. 28(a)	-0.3743/-0.6539	0.0093/-0.0013	0.4046/0.6758	0.1743/0.3244
Fig. 28(b)	-0.4908/-0.7404	0.0122/-0.0035	0.5297/0.7739	0.2724/0.3743

ResNet20 converge to zero, while EfficientNetB0 and MobileNetV2 minimum and maximum move away from zero, which explains the higher contrast (also seen in increasing std).

Section summary: Results in this section are related to semantics and explainability. We analyzed filters from the first convolutional layers of multiple CNNs pretrained on ImageNet and clustered them. The resulting clusters were divided by their semantic meaning: Gaussian-like, edge detectors, and texture detectors. The cluster semantic meaning aligns with the semantics of F-transform kernels. F-transform kernels and He initialization were compared through 4 different CNNs trained on MNIST and CIFAR-10. F-transform kernels retained their shapes mostly and thus their interpretability. After initial experiments on MNIST, we further filtered the kernels to contain F^1 only. The F^1 changed insignificantly after training. We argue that F^1 suitability comes from the MNIST nature, where edges are the most dominant features. Lastly, throughout the section, we could observe similarities between F-transform kernels and He initialization kernels after training. The fact that He initialization kernels converge to the same shapes as F-transform proves their good descriptivity.

7 Conclusion

The main goal of this thesis is to demonstrate the possibilities of the Fuzzy transform (F-transform) methodology in combination with the technology of convolutional neural networks. We have studied F-transform from the perspective of convolutional neural network initialization using its semantic clearness and descriptiveness.

We have developed a new initialization tool for a convolutional neural network that provides a unique view of the initialization process. Section 4 defines the F-transform of various degrees and discusses its properties. Namely, a good approximation property and clear semantic meaning of F-transform components. The emphasis was put on semantic meaning, as the explainability of a deep neural network is challenging. Furthermore, in this section, we showed a different way to obtain the F-transform components using convolutional kernels. These kernels are defined in the form of convolutional matrices.

The F-transform initialization tool is explained in section 6.1 on the example of the CNN with LeNet5-like architecture. As a result, we obtained FTNet with F-transform initialization. F-transform initialization was compared with He initialization, resulting in similar accuracy. Unlike the He initialization, the F-transform initialization keeps the major part of the kernels unchanged due to their extracted feature expressiveness, leading to faster training. Faster training is linked to less energy consummation and CO_2 emission, putting less pressure on the environment.

Section 6.2 generalizes the F-transform initialization to initialize any kind of convolutional neural network. The experiments in the section show better performance of two-state-of-the-art convolutional neural networks, ResNet and EfficientNetB0, initialized with F-transform initialization than He initialization.

Finally, section 6.3 the studies semantics of kernels in the first convolutional layer of networks. First, we described the clusters of kernels from selected networks and put the clusters into correspondence with F-transform initialization. The second part of the section analyzed how the first convolutional layer kernels initialized with F-transform kernels changed through the training. It concluded with F-transform initialization kernels largely keeping their functionality, which confirms their suitability for convolutional neural network initialization.

Summary

F-transform kernel initialization is a novel way of initializing the CNNs. It brings an intelligent way of creating convolutional layer filters. The created filters utilize F-transform components (extracted features) connected with good object separability and consequent reconstruction and classification. Other noteworthy initializations, such as He or Xavier initialization, are based on normal and uniform probability distributions. Creating filters by sampling from the probability distribution considers the statistical characteristics but omits any semantical properties. Our approach is built on semantic information in the F-transform kernels, and it is deliberately used, e.g., MNIST dominant features are edges, so F^1 is the most efficient kernel in the first convolutional layer. Incorporating semantic information into initialization makes it a conscious process that allows analyzing network behavior – explainability.

F-transform kernel initialization compared to He and Xavier achieved better accuracy and loss when tested on multiple datasets using state-of-the-art networks. Apart from better performance, it also allows for freezing convolutional layers, which speed up training. Faster training allows deep learning practitioners to reduce energy consumption and CO_2 emissions.

After extracting and clustering kernels from the first convolutional layer of prominent CNNs, F-transform kernels matched the discovered categories. When used for initialization, F-transform kernels stay mostly the same. All said results lead to the conclusion that F-transform kernels are indeed suitable for initialization, help CNNs with classification problems, are easily explainable, and more efficient at training.

Resumé

Inicializace pomocí F-transformačních jader je nová metoda inicializování konvolučních neuronových sítí. Ta přináší inteligentní způsob jak vytvářet filtry konvolučních sítí. Vytvořené filtry využívají komponenty F-transformace (extrahované rysy) spojené s dobrou separabilitou objektů a následnou rekonstrukcí a klasifikací. Ostatní inicializace, které stojí za povšimnutí, jako například He nebo Xavier inicializace, jsou založené na normálním a rovnoměrném pravděpodobnostním rozdělení. Tvoření filtrů vzorkováním z pravděpodobnostních rozdělení bere v potaz pouze statistické charakteristiky, ale zanedbává vlastnosti sémantické. Náš přístup je postaven na sémantické informaci, která je v F-transformačních kernelech a je cíleně využívána, např. dominantní rysy databáze MNIST jsou hrany, proto je F^1 nejefektivnější kernel v první konvoluční vrstvě. Zanesení sémantické informace do inicializace umožňuje analýzu chování sítí – vysvětlitelnost.

Inicializace pomocí F-transformačních jader ve srovnání s He a Xavier inicializací dosahuje lepší přesnosti a menší chybě během testování na několika datasetech s využitím nejmodernějších sítí. Krom lepšího výkonu, naše inicializace taktéž umožňuje zamrazení konvolučních vrstev, což vede ke zrychlení učení. Rychlejší učení je šetrnější k životnímu prostředí, to znamená nižší CO_2 emise a snížení spotřeby elektřiny.

Po extrakci a shlukování jader z prvních vrstev několika prominentních konvolučních neuronových sítí byly F-transformační kernely sdruženy s objevenými shluky. Během využití F-transformačních kernelů zůstaly kernely povětšinou beze změny. Všechny zmíněné výsledky vedou k závěru, že F-transformační kernely jsou vskutku vhodné k inicializaci, napomáhají konvolučním neuronovým sítím k řešení klasifikačních problémů, jsou jednoduše vysvětlitelné a vedou k vyšší efektivitě trénování.

References

[1] M. Abadi et al. “Tensorflow: A system for large-scale machine learning”. In: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 2016, pp. 265–283.

[2] S. Ahlawat and R. Rishi. “Handwritten Digit Recognition using Adaptive Neuro-Fuzzy System and Ranked Features”. In: *2018 International Conference on Computing, Power and Communication Technologies (GUCON)*. IEEE. 2018, pp. 1128–1132.

[3] Y. Assiri. “Stochastic Optimization of Plain Convolutional Neural Networks with Simple methods”. In: *arXiv preprint arXiv:2001.08856* (2020).

[4] A. I. Aviles et al. “A deep-neuro-fuzzy approach for estimating the interaction forces in robotic surgery”. In: *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE. 2016, pp. 1113–1119.

[5] D. H. Ballard. “Modular Learning in Neural Networks.” In: *AAAI*. 1987, pp. 279–284.

[6] A. B. Bayat. “Recognition of Handwritten Digits Using Optimized Adaptive Neuro-Fuzzy Inference Systems and Effective Features”. In: *Journal of Pattern Recognition and Intelligent Systems* 1.2 (2013), pp. 25–37.

[7] Y. Bengio, A. Courville, and P. Vincent. “Representation learning: A review and new perspectives”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[8] Y. Bengio et al. “Greedy layer-wise training of deep networks”. In: *Advances in neural information processing systems*. 2007, pp. 153–160.

[9] J. Canny. “A computational approach to edge detection”. In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), pp. 679–698.

[10] N. Carion et al. “End-to-End Object Detection with Transformers”. In: *arXiv preprint arXiv:2005.12872* (2020).

[11] B. Cetişli and R. Edizkan. “Use of wavelet-based two-dimensional scaling moments and structural features in cascade neuro-fuzzy classifiers for handwritten digit recognition”. In: *Neural Computing and Applications* 26.3 (2015), pp. 613–624.

[12] K. Chellapilla, S. Puri, and P. Simard. “High performance convolutional neural networks for document processing”. In: *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft. 2006.

[13] Y. Chen et al. “Dual path networks”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 4467–4475.

[14] M. Cho et al. “PowerAI DDL”. In: *arXiv preprint arXiv:1708.02188* (2017).

[15] F. Chollet et al. *Keras*. `https://keras.io`. 2015.

[16] D. C. Ciresan et al. “Flexible, high performance convolutional neural networks for image classification”. In: *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[17] D. C. Cireşan et al. “Deep, big, simple neural nets for handwritten digit recognition”. In: *Neural computation* 22.12 (2010), pp. 3207–3220.

[18] D. Cireşan, U. Meier, and J. Schmidhuber. “Multi-column deep neural networks for image classification”. In: *arXiv preprint arXiv:1202.2745* (2012).

[19] D. Cireşan et al. “A committee of neural networks for traffic sign classification”. In: *Neural Networks (IJCNN), The 2011 International Joint Conference on*. IEEE. 2011, pp. 1918–1921.

[20] D. Ciresan et al. *Deep big simple neural nets excel on handwritten digit recognition. CoRR abs/1003.0358 (2010)*.

[21] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[22] Y. N. Dauphin and S. Schoenholz. “MetaInit: Initializing learning by learning to initialize”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 12624–12636.

[23] L. Deng. “The MNIST database of handwritten digit images for machine learning research [best of the web]”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[24] Y. Deng et al. “A hierarchical fused fuzzy deep neural network for data classification”. In: *IEEE Transactions on Fuzzy Systems* 25.4 (2017), pp. 1006–1012.

[25] Y. Deng et al. “Visual words assignment via information-theoretic manifold embedding”. In: *IEEE transactions on cybernetics* 44.10 (2014), pp. 1924–1937.

[26] J. Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[27] J. Duchi, E. Hazan, and Y. Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[28] L. Fei-Fei, R. Fergus, and P. Perona. "One-shot learning of object categories". In: *IEEE transactions on pattern analysis and machine intelligence* 28.4 (2006), pp. 594–611.

[29] S. Feng, C. P. Chen, and C.-Y. Zhang. "A Fuzzy Deep Model Based on Fuzzy Restricted Boltzmann Machines for High-dimensional Data Classification". In: *IEEE Transactions on Fuzzy Systems* (2019).

[30] K. Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological cybernetics* 36.4 (1980), pp. 193–202.

[31] D. Gabor. *Theory of communication: Journal of the Institute of Electrical Engineers, 93, 429–457*. 1946.

[32] R. Girshick. "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[33] R. Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[34] 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.

[35] X. Glorot, A. Bordes, and Y. Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 315–323.

[36] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[37] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[38] I. Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[39] P. J. Grother. "NIST special database 19 handprinted forms and characters database". In: *National Institute of Standards and Technology* (1995).

[40] R. H. Hahnloser et al. "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit". In: *Nature* 405.6789 (2000), pp. 947–951.

[41] K. He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[42] K. He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *CoRR* abs/1502.01852 (2015). arXiv: 1502.01852. URL: http://arxiv.org/abs/1502.01852.

[43] K. He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[44] K. He et al. “Identity mappings in deep residual networks”. In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[45] G. E. Hinton, S. Osindero, and Y.-W. Teh. “A fast learning algorithm for deep belief nets”. In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[46] G. E. Hinton and R. R. Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: *science* 313.5786 (2006), pp. 504–507.

[47] P. Hodáková. “Fuzzy (F-) transform of functions of two variables and its applications in image processing”. PhD thesis. University of Ostrava, 2014.

[48] A. G. Howard et al. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861* (2017).

[49] F. J. Huang, Y.-L. Boureau, Y. LeCun, et al. “Unsupervised learning of invariant feature hierarchies with applications to object recognition”. In: *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE. 2007, pp. 1–8.

[50] G. Huang et al. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[51] D. H. Hubel and T. N. Wiesel. “Receptive fields of single neurones in the cat's striate cortex”. In: *The Journal of physiology* 148.3 (1959), pp. 574–591.

[52] D. H. Hubel and T. N. Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat's visual cortex”. In: *The Journal of physiology* 160.1 (1962), pp. 106–154.

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

[54] S. Ioffe and C. Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015).

[55] G. James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[56] Y. Jia et al. “Caffe: Convolutional Architecture for Fast Feature Embedding”. In: *arXiv preprint arXiv:1408.5093* (2014).

[57] C.-F. Juang and C.-T. Lin. “An online self-constructing neural fuzzy inference network and its applications”. In: *IEEE transactions on Fuzzy Systems* 6.1 (1998), pp. 12–32.

[58] A. B. Khalifa and H. Frigui. “Multiple Instance Fuzzy Inference Neural Networks”. In: *arXiv preprint arXiv:1610.04973* (2016).

[59] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[60] D. P. Kingma and M. Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[61] P. Krähenbühl et al. “Data-dependent initializations of convolutional neural networks”. In: *arXiv preprint arXiv:1511.06856* (2015).

[62] A. Krizhevsky, G. Hinton, et al. *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer, 2009.

[63] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[64] S. K. Kumar. “On weight initialization in deep neural networks”. In: *arXiv preprint arXiv:1704.08863* (2017).

[65] T. Kuremoto et al. “Time series forecasting using a deep belief network with restricted Boltzmann machines”. In: *Neurocomputing* 137 (2014), pp. 47–56.

[66] Z. Lan et al. “Albert: A lite bert for self-supervised learning of language representations”. In: *arXiv preprint arXiv:1909.11942* (2019).

[67] Y. LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[68] C.-T. Lin et al. “Support-vector-based fuzzy neural network for pattern classification”. In: *IEEE Transactions on Fuzzy Systems* 14.1 (2006), pp. 31–41.

[69] Z. C. Lipton et al. “Learning to diagnose with LSTM recurrent neural networks”. In: *arXiv preprint arXiv:1511.03677* (2015).

[70] D. G. Lowe. “Object recognition from local scale-invariant features”. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[71] R. K. McConnell. *Method of and apparatus for pattern recognition*. US Patent 4,567,610. Jan. 1986.

[72] T. Mikolov et al. “Distributed representations of words and phrases and their compositionality”. In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[73] D. Mishkin and J. Matas. “All you need is a good init”. In: *arXiv preprint arXiv:1511.06422* (2015).

[74] D. Misra. “Mish: A self regularized non-monotonic neural activation function”. In: *arXiv preprint arXiv:1908.08681* (2019).

[75] 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.

[76] V. Molek and I. Perfilieva. “Deep Learning and Higher Degree F-Transforms: Interpretable Kernels Before and After Learning”. In: *International Journal of Computational Intelligence Systems* 13.1 (2020), pp. 1404–1414.

[77] V. Molek and I. Perfilieva. “F-Transform and Convolutional NN: Cross-Fertilization and Step Forward”. In: *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE. 2020, pp. 1–6.

[78] V. Molek and I. Perfilieva. “Scale-Space Theory, F-transform kernels and CNN Realization”. In: *International Work-Conference on Artificial Neural Networks*. 2019, pp. 320–332.

[79] V. MOLEK and I. PERFILIEVA. “RELATIONSHIP BETWEEN CONVOLUTIONAL NEURAL NETWORKS AND F-TRANSFORM”. In: *Uncertainty Modelling in Knowledge Engineering and Decision Making: Proceedings of the 12th International FLINS Conference (FLINS 2016)*. Vol. 10. World Scientific. 2016, p. 325.

[80] A. Nøkland. “Direct feedback alignment provides learning in deep neural networks”. In: *Advances in neural information processing systems*. 2016, pp. 1037–1045.

[81] K.-S. Oh and K. Jung. “GPU implementation of neural networks”. In: *Pattern Recognition* 37.6 (2004), pp. 1311–1314.

[82] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu. “Pixel recurrent neural networks”. In: *arXiv preprint arXiv:1601.06759* (2016).

[83] K. K. Pal and K. Sudeep. “Preprocessing for image classification by convolutional neural networks”. In: *Recent Trends in Electronics, Information & Communication Technology (RTEICT), IEEE International Conference on*. IEEE. 2016, pp. 1778–1781.

[84] A. Paszke et al. “Automatic differentiation in PyTorch”. In: *NIPS-W*. 2017.

[85] 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.

[86] K. Pearson. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572.

[87] I. Perfilieva and E. Haldeeva. “Fuzzy transformation and its applications”. In: *4th Czech-Japan Seminar on Data Analysis and Decision Making under Uncertainity*. 2001, pp. 116–124.

[88] I. Perfilieva and V. Molek. “Discovering basic kernels in the first layer of a CNN”. In: *ISCAMI*. 2018.

[89] I. Perfilieva, M. Danková, and B. Bede. “Towards F-transform of a Higher Degree.” In: *IFSA/EUSFLAT Conf.* 2009, pp. 585–588.

[90] I. Perfilieva, P. Hodáková, and P. Hurtík. “Differentiation by the F-transform and application to edge detection”. In: *Fuzzy Sets and Systems* 288 (2016), pp. 96–114.

[91] I. Perfilieva, P. Hodáková, and P. Hurtík. “F 1-transform edge detector inspired by canny’s algorithm”. In: *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*. Springer. 2012, pp. 230–239.

[92] I. Perfilieva, M. Holčapek, and V. Kreinovich. “A new reconstruction from the F-transform components”. In: *Fuzzy Sets and Systems* 288 (2016), pp. 3–25.

[93] I. Perfilieva and V. Kreinovich. “Fuzzy transforms of higher order approximate derivatives: A theorem”. In: *Fuzzy Sets and Systems* 180.1 (2011), pp. 55–68.

[94] I. Perfilieva et al. “F-transform based image fusion”. In: *Image Fusion*. IntechOpen, 2011.

[95] E. Popko and I. Weinstein. “Fuzzy Logic Module of Convolutional Neural Network for Handwritten Digits Recognition”. In: *Journal of Physics: Conference Series*. Vol. 738. 1. IOP Publishing. 2016, p. 012123.

[96] J. M. Prewitt. “Object enhancement and extraction”. In: *Picture processing and Psychopictorics* 10.1 (1970), pp. 15–19.

[97] A. Rakitianskaia and A. Engelbrecht. “Measuring saturation in neural networks”. In: *2015 IEEE Symposium Series on Computational Intelligence*. IEEE. 2015, pp. 1423–1430.

[98] P. Ramachandran, B. Zoph, and Q. V. Le. “Searching for activation functions”. In: *arXiv preprint arXiv:1710.05941* (2017).

[99] J. Redmon and A. Farhadi. “YOLO9000: better, faster, stronger”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.

[100] J. Redmon and A. Farhadi. “Yolov3: An incremental improvement”. In: *arXiv preprint arXiv:1804.02767* (2018).

[101] J. Redmon et al. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[102] S. Ren et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[103] O. Ronneberger, P. Fischer, and T. Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[104] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[105] O. Russakovsky et al. “Imagenet large scale visual recognition challenge”. In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[106] S. Sabour, N. Frosst, and G. E. Hinton. “Dynamic routing between capsules”. In: *Advances in neural information processing systems*. 2017, pp. 3856–3866.

[107] M. Sandler et al. “Mobilenetv2: Inverted residuals and linear bottlenecks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[108] A. M. Saxe, J. L. McClelland, and S. Ganguli. “Exact solutions to the nonlinear dynamics of learning in deep linear neural networks”. In: *arXiv preprint arXiv:1312.6120* (2013).

[109] D. S. Shukla. “Deep Neuro-fuzzy Classification method to Alzheimer ” s disease Detection”. In: *International Journal of Engineering Research* Volume 2 (June 2017), pp. 3905–3909. DOI: 10.23956/ijarcsse/V7I6/0259.

[110] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556 (2014). URL: http://arxiv.org/abs/1409.1556.

[111] I. Sobel and G. Feldman. “A 3x3 isotropic gradient operator for image processing”. In: *a talk at the Stanford Artificial Project in* (1968), pp. 271–272.

[112] J. Stallkamp et al. “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition”. In: *Neural networks* 32 (2012), pp. 323–332.

[113] E. Strubell, A. Ganesh, and A. McCallum. “Energy and policy considerations for deep learning in NLP”. In: *arXiv preprint arXiv:1906.02243* (2019).

[114] D. Sussillo and L. Abbott. “Random walk initialization for training very deep feedforward networks”. In: *arXiv preprint arXiv:1412.6558* (2014).

[115] C. Szegedy et al. “Going Deeper with Convolutions”. In: *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

[116] C. Szegedy et al. “Inception-v4, inception-resnet and the impact of residual connections on learning”. In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[117] C. Szegedy et al. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[118] M. Tan and Q. V. Le. “Efficientnet: Rethinking model scaling for convolutional neural networks”. In: *arXiv preprint arXiv:1905.11946* (2019).

[119] J. Tang, C. Deng, and G.-B. Huang. “Extreme learning machine for multilayer perceptron”. In: *IEEE transactions on neural networks and learning systems* 27.4 (2015), pp. 809–821.

[120] A. Torralba, R. Fergus, and W. T. Freeman. “80 million tiny images: A large data set for nonparametric object and scene recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* 30.11 (2008), pp. 1958–1970.

[121] J. Turian, J. Bergstra, and Y. Bengio. “Quadratic features and deep architectures for chunking”. In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*. 2009, pp. 245–248.

[122] M. Vajgl, I. Perfilieva, and P. Hod’áková. “Advanced F-transform-based image fusion”. In: *Advances in Fuzzy Systems* 2012 (2012), p. 4.

[123] A. Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[124] P. Vlašánek and I. Perfilieva. “Image reconstruction with usage of the F-Transform”. In: *International Joint Conference CISIS’12-ICEUTE´ 12-SOCO´ 12 Special Sessions*. Springer. 2013, pp. 507–514.

[125] P. Vlašánek and I. Perfilieva. “The F-transform in Terms of Image Processing Tools”. In: *Journal of Fuzzy Set Valued Analysis* 2016 (2016), pp. 54–62.

[126] J. Weng, N. Ahuja, and T. S. Huang. “Cresceptron: a self-organizing neural network which grows adaptively”. In: *Neural Networks, 1992. IJCNN., International Joint Conference on*. Vol. 1. IEEE. 1992, pp. 576–581.

[127] D. R. G. H. R. Williams and G. Hinton. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536.

[128] L. Xiao et al. “Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks”. In: *arXiv preprint arXiv:1806.05393* (2018).

[129] O. Yazdanbakhsh and S. Dick. “A Deep Neuro-Fuzzy Network for Image Classification”. In: *arXiv preprint arXiv:2001.01686* (2019).

[130] T. Ye. “Visual object detection from lifelogs using visual non-lifelog data”. PhD thesis. Dublin City University, 2018.

[131] M. D. Zeiler. “Adadelta: an adaptive learning rate method”. In: *arXiv preprint arXiv:1212.5701* (2012).

[132] M. D. Zeiler and R. Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[133] B. Zoph et al. “Learning transferable architectures for scalable image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8697–8710.

Author's contributions

[75] 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.

[76] V. Molek and I. Perfilieva. "Deep Learning and Higher Degree F-Transforms: Interpretable Kernels Before and After Learning". In: *International Journal of Computational Intelligence Systems* 13.1 (2020), pp. 1404–1414.

[77] V. Molek and I. Perfilieva. "F-Transform and Convolutional NN: Cross-Fertilization and Step Forward". In: *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE. 2020, pp. 1–6.

[78] V. Molek and I. Perfilieva. "Scale-Space Theory, F-transform kernels and CNN Realization". In: *International Work-Conference on Artificial Neural Networks*. 2019, pp. 320–332.

[79] V. MOLEK and I. PERFILIEVA. "RELATIONSHIP BETWEEN CONVOLUTIONAL NEURAL NETWORKS AND F-TRANSFORM". In: *Uncertainty Modelling in Knowledge Engineering and Decision Making: Proceedings of the 12th International FLINS Conference (FLINS 2016)*. Vol. 10. World Scientific. 2016, p. 325.

[88] I. Perfilieva and V. Molek. "Discovering basic kernels in the first layer of a CNN". In: *ISCAMI*. 2018.

List of Figures

List of Tables