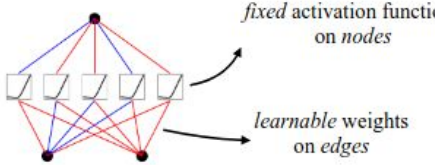
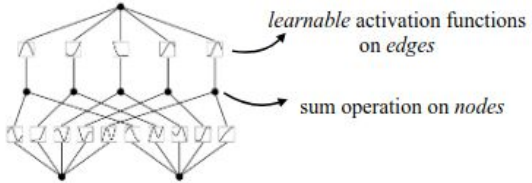
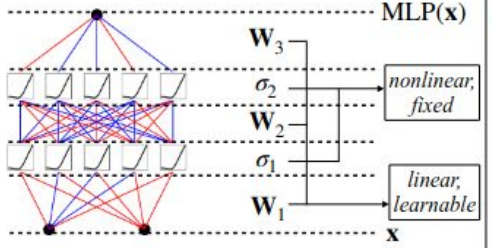
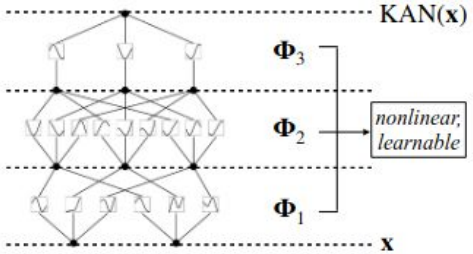
KAN

Применение сетей Колмогорова-Арнольда для решения задач компьютерного зрения

План доклада

1) Ранняя работа (сразу после первого доклада)
2) Дальнейшее продвижение
3) Текущие результаты
4) Текущие проблемы
5) Дальнейшее исследование

Основная идея KAN

Model	Multi-Layer Perceptron (MLP)	Kolmogorov-Arnold Network (KAN)
Theorem	Universal Approximation Theorem	Kolmogorov-Arnold Representation Theorem
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$
Model (Shallow)	(a) 	(b) 
Formula (Deep)	$\mathrm{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\mathrm{KAN}(\mathbf{x}) = (\mathbf{\Phi}_3 \circ \mathbf{\Phi}_2 \circ \mathbf{\Phi}_1)(\mathbf{x})$
Model (Deep)	(c) 	(d) 

Ранняя работа (до первого доклада)

1) Составление пробного литобзора: https://docs.google.com/document/d/1BmdW4PVTmEM5WA25ypgqXOwJL_42EFL2BYlEtCww5f4/edit?tab=t.0 (done!) -> EASY
2) повторение эксперимента (done!) -> MEDIUM

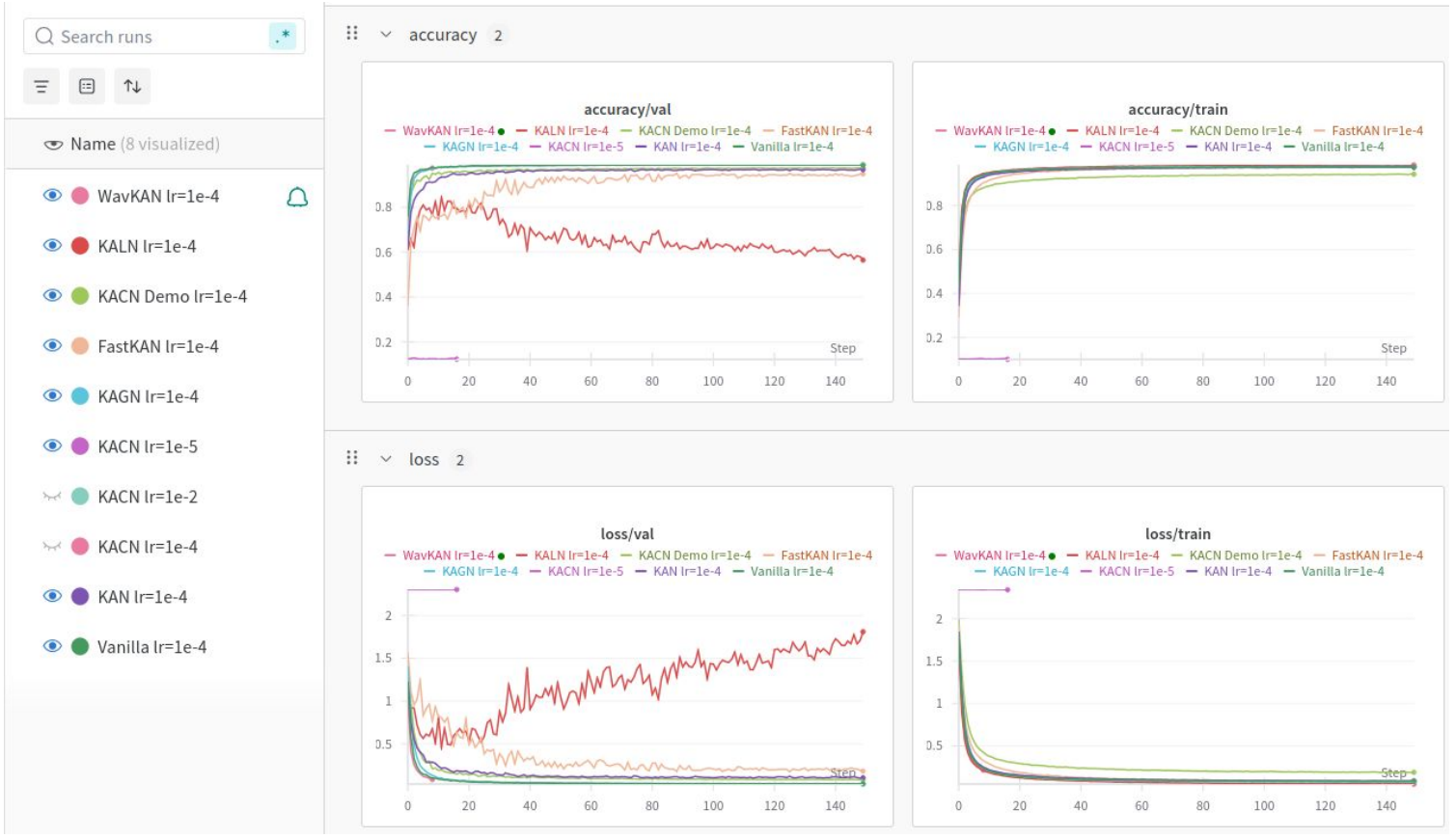
Повторение эксперимента

Model	MNIST			CIFAR10			CIFAR100		
	Val. Accuracy	Params., M	Eval Time, s	Val. Accuracy	Params.,	Eval Time, s	Val. Accuracy	Params.,	Eval Time, s
Conv, 4 layers, baseline	99.42	0.1	0.7008	73.18	0.1	1.8321	42.29	0.12	1.5994
KANConv, 4 layers	99.00	3.49	2.6401	52.08	3.49	3.7972	21.78	3.52	4.0262
FastKANConv, 4 layers	97.65	3.49	1.5999	64.95	3.49	2.3716	34.32	3.52	2.7457
KALNConv, 4 layers	84.85	1.94	1.7205	10.28	1.94	3.0527	5.97	1.97	3.0919
KACNConv, 4 layers	97.62	3.92	1.6710	52.01	3.92	2.3972	23.17	0.42	2.6522
KAGNConv, 4 layers	**99.49**	0.49	1.7253	65.84	0.49	2.2570	**47.36**	1.97	2.3399
WavKANConv, 4 layers	99.23	0.95	7.4622	**73.63**	0.95	11.2276	41.50	0.98	11.4744
Conv, 8 layers, baseline	99.63	1.14	1.2061	83.05	1.14	1.8258	57.52	1.19	1.8265
KANConv, 8 layers	99.37	40.7	4.2011	74.66	40.7	5.4858	36.18	40.74	5.7067
FastKANConv, 8 layers	99.49	40.7	2.1653	74.66	40.7	5.4858	43.32	40.74	2.7771
KALNConv, 8 layers	49.97	22.61	1.7815	15.97	22.61	2.7348	1.74	22.65	2.6863
KACNConv, 8 layers	99.32	18.09	1.6973	62.14	18.09	2.3459	25.01	18.14	2.3826
KAGNConv, 8 layers	**99.68**	22.61	2.2402	84.14	22.61	2.5849	**59.27**	22.66	2.6460
WavKANConv, 8 layers	99.57	10.73	59.1734	**85.37**	10.73	28.0385	55.43	10.78	30.5438

Table 1: Results on MNIST, CIFAR10, and CIFAR100 datasets

- **Slim**: 16, 32, 64, 128, 256, 256, 512, 512
- **Wide**: 32, 64, 128, 256, 512, 512, 1024, 1024

Повторение эксперимента



Дальнейшее исследование

1) Апгрейд литобзора: https://docs.google.com/document/d/1BmdW4PVTmEM5WA25ypgqXOwJL_42EFL2BYIEtCww5f4/edit?tab=t.0 (done!) -> EASY
2) повторение экспериментов (proccessing…) -> MEDIUM
3) имплементация KAN на c++ (ending…) -> VERY HARD
4) формулировка проблем (done!) -> EASY

Апгрейд литобзора

KAN not Work: Investigating the Applicability of Kolmogorov-Arnold Networks in Computer Vision

Yueyang Cang[1*] Yuhang Liu[1*] Shi Li[1†]

[1]Tsinghua University, Beijing, China

cangyy23@mails.tsinghua.edu.cn, yh-liu23@mails.tsinghua.edu.cn, shilits@tsinghua.edu.cn

---

KOLMOGOROV-ARNOLD CONVOLUTIONS: DESIGN PRINCIPLES AND EMPIRICAL STUDIES

A PREPRINT

Ivan Drokin
Deep Learning Researcher
Seath the Scaleless Research Group
ivan.s.drokin@gmail.com

Повторение эксперимента

Task	Model	Replacement	Accuracy / mIoU (%)
Image Classification	MobileNet	Baseline	61.14
	MobileNet	KAN (MLP replaced)	60.98
	MobileNet	First Layer CKAN	58.27
	MobileNet	Last Layer CKAN	58.64
	MobileNet	two CKANs	57.46
Semantic Segmentation	UNet	Baseline	63.28
	UNet	CKAN (Final CNN replaced)	59.13

Table 1. **Performance Comparison of Baseline Models with KAN and CKAN on CIFAR-100 and PASCAL VOC2012**.

Model	First Layer	Second Layer	Final Layer
CNN+MLP	CNN (3→32, 3x3)	CNN (32→64, 3x3)	MLP
CNN+CKAN+MLP	CKAN (3→32, 3x3)	CNN (32→64, 3x3)	MLP
CNN+KAN	CNN (3→32, 3x3)	CNN (32→64, 3x3)	KAN

Table 2. **Experimental Model Configurations**. The baseline model (CNN+MLP) serves as a control for comparison with the CKAN and KAN-modified models.

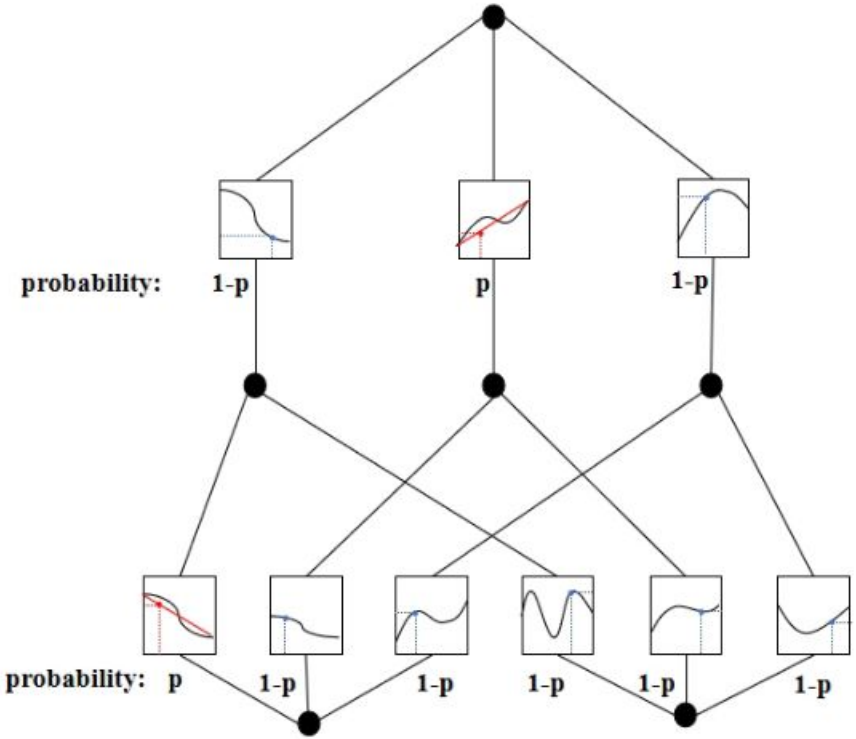
Model	20%	40%	60%	80%	100%
CNN+MLP	62.31%	67.26%	69.93%	71.33%	71.96%
CKAN+CNN+MLP	61.82%	67.05%	69.03%	70.91%	71.88%
CNN+KAN	64.02%	68.15%	70.41%	72.13%	72.58%

Table 3. **Performance of Different Models on CIFAR-10 with Varying Dataset Sizes**. Each column shows the accuracy (%) of each model as the dataset size increases from 20% to 100%.

Model	10%	20%	30%	40%	50%
CNN+MLP	69.13%	67.93%	65.38%	63.88%	62.87%
CKAN+CNN+MLP	69.59%	66.73%	64.19%	62.56%	58.43%
CNN+KAN	69.95%	67.81%	64.33%	62.84%	59.20%

Table 4. **Performance of Different Models with Increasing Label Noise**. Each column represents the accuracy (%) as the label noise increases from 10% to 50%.

Повторение эксперимента



Имплементация на c++

1) CUDA KERNELS:
2) SE craft torch
3) Ivan Drokin implenentation

CUDA C++ Programming Guide

```python
def forward(self, x):
    # Process each layer using the defined base weights, spline weights, norms, and activations.
    grid = self.grid.to(x.device)
    # Move the input tensor to the device where the weights are located.

    # Perform the base linear transformation followed by the activation function.
    base_output = F.linear(self.base_activation(x), self.base_weight)
    x_uns = x.unsqueeze(-1)  # Expand dimensions for spline operations.
    # Compute the basis for the spline using intervals and input values.
    bases = ((x_uns >= grid[:, :-1]) & (x_uns < grid[:, 1:])).to(x.dtype).to(x.device)

    # Compute the spline basis over multiple orders.
    for k in range(1, self.spline_order + 1):
        left_intervals = grid[:, :-(k + 1)]
        right_intervals = grid[:, k:-1]
        delta = torch.where(right_intervals == left_intervals, torch.ones_like(right_intervals),
                            right_intervals - left_intervals)
        bases = ((x_uns - left_intervals) / delta * bases[:, :, :-1]) + \
                ((grid[:, k + 1:] - x_uns) / (grid[:, k + 1:] - grid[:, 1:(-k)]) * bases[:, :, 1:])
    bases = bases.contiguous()

    # Compute the spline transformation and combine it with the base transformation.
    spline_output = F.linear(bases.view(x.size(0), -1), self.spline_weight.view(self.spline_weight.size(0), -1))
    # Apply layer normalization and PReLU activation to the combined output.
    x = self.prelu(self.layer_norm(base_output + spline_output))

    return x
```

Формулировка проблем

1) Подбор датасета
2) Выбор оптимизации
3) Имплементация