KL-CPD for video data
(Selected Topics in Data Science 2022 Course)

Evgenia Romanenkova [1] **Dmitriy Ermilov** [1] **Alexander Lukashevich** [1]

Abstract

KL-CPD, a novel kernel learning framework for time series CPD that optimizes a lower bound of test power via an auxiliary generative model. With deep kernel parameterization, KL-CPD endows kernel twosample test with the data-driven kernel to detect different types of change-points in real-world applications. However, it requires a lot of memory and resources to work. In this project, we try to improve performance of this method in terms of quality and efficiency to make it applicable to video surveillance.

1. Introduction

Change points are abrupt disorders in the distribution of sequential data. The change-point detection (CPD) model signals the appearance of such changes in data streams. Historically, there were many rigid theoretical results on the data-driven CPD approaches, and, in many cases, precise change detection criteria exist. However, these methods are not suitable for complex video data because of the absence of adequate data representation learning. The example of change in surveillance video (the explosion moment) is presented in Figure 1.

The majority of existing representation-based methods are based on non-principled procedures and consider a CPD problem as a binary-classification problem close to anomaly detection area(Oord et al., 2018; Sultani et al., 2018). In contrast, the KL-CPD method proposed in the paper (Chang et al., 2018) is based on both classic kernel CPD methods (Aminikhanghahi & Cook, 2017; Truong et al., 2020) that use Maximum Mean Discrepancy (MMD) (Li et al., 2015), but a trainable generative model in its core for learning kernels parameters. However, such a model requires many resources for training in case of complex video data. Due to its architecture similar to GAN-based approaches (Goodfellow et al., 2014), we need to encode really high-dimensional sequences and decode them in original space.

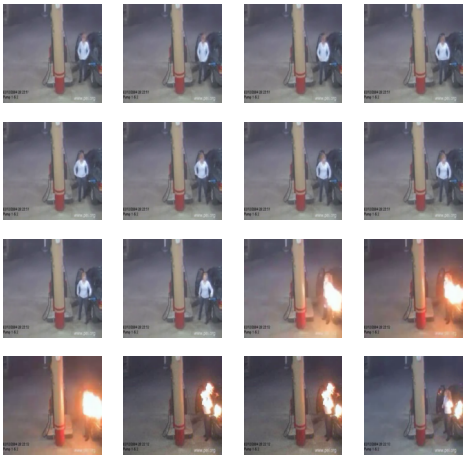


Figure 1. Example with change point (explosion). At each row, a sequence goes from left to right. Then we continue with the most left image in the next row.

Our project aims to overcome this technical issue and compare different approaches for reducing neural network space parameters without great changes in the KL-CPD model's architecture and preserving its quality. In particular, our main claims are the following:

- We compare the different common approaches for video processing, such as reducing the video quality or size, using pre-trained encoders with frozen and trainable parameters as well as adding projection layers in KL-CPD models;
- In addition, we consider reducing the number of model parameters by pruning and/or replacing the core model's layers by their low-rank approximations.

2. Related work

Talking about the cost of neural networks, the number of parameters is surely one of the most widely used metrics,

[*]Equal contribution [1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Evgenia Romanenkova <Evgenia.Romanenkova@skoltech.ru>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

along with FLOPs (floating-point operations). The two obvious ways to reduce the cost of neural networks are to preprocess the initial data and reduce its dimension or to compress the model itself. In this section, we consider common approaches for both tracks.

2.1. Initial data compression

On the side of preprocessing initial data, which would significantly reduce the number of trainable parameters and FLOPS, there exist two streams: *image compression* and *video compression*.

Image Compression
Among *image* compression, there are classical approaches, e.g., JPEG compression standard linearly maps the pixels to another representation by using Discrete Cosine Transform (DCT) and quantizes the corresponding coefficients before entropy coding (Wallace, 1992). However, such methods are not suitable for our purposes since they do not provide a reduction in image height and width but induce sparsity of the DCT image of a picture. Moreover, Manifold Learning approaches are used for image compression to retrieve the estimation of the manifold on which the images lie (Pless & Souvenir, 2009). Such approaches typically include Isometric Feature Mapping (ISOMAP) or Locally Linear Embedding (LLE).

Video Compression
Similarly, for *video* compression, as an ordered sequence of images, there are, similarly to images, classical, DNN and Manifold Learning approaches. Among classical ones are H.265 (Sullivan et al., 2012) and H.264 (Wiegand et al., 2003) standards. These algorithms follow a predictive coding paradigm. Specifically, the major steps of these algorithms are (Lu et al., 2019): 1) Motion estimation; 2) Motion compensation; 3) Transform and quantization; 4) Inverse transform; 5) Entropy coding; 6) Frame reconstruction. Authors in (Lu et al., 2019) moved forward and added compression into step one and used different CNNs for each step of the classical algorithms. The loss function was constructed with reconstruction error and the term that penalized bit rate for motion encoding and residual representation. Finally, an application of Manifold Learning is presented in (Nie et al., 2011). The authors proposed using ISOMAP for video compression for further similarity measures between videos.

Using pre-trained feature extractor
Aside from classical well-known compression algorithms, recently, Deep Neural Network (DNN) compression algorithm became popular. The first widely-spread approach is to use the model pre-trained on a large dataset in a supervised manner as a feature extractor. After applying a pre-trained extractor, we hope that the obtained features are good enough and reflect all the main characteristics of the data. The natural choice of such an extractor for images is ResNet architecture family (He et al., 2016), and for videos is 3D CNN-based (Feichtenhofer et al., 2019; Oord et al., 2018).

Another technique concentrates on direct learning low-dimensional data representation. They are usually based on Recurrent Neural Networks (RNN) (Toderici et al., 2017; 2016) or Convolutional Neural Networks (CNN) (Ballé et al., 2017; 2018). In the case of RNN aims to sequentially make a forward pass to obtain a latent vector that can be used as a low-dimensional representation of as well an input image or input video. Similarly, CNN auto-encoder models are also adopted for image and video compression where the latent representation, i.e., the encoder output, is used as a low-dimensional representation.

While being pretty compelling from a technical point of view, all these scenarios may lead to a significant decrease in the quality of the model because of losing essential data patterns.

2.2. Model compression

On the side of model compression, the number of trainable parameters and FLOPs can be reduced by pruning redundant connections in the neural network or by using weights of the neural network in a low-rank format.

Weights Pruning
Models that perform well in the computer vision domain are often over-parameterized and contain redundant connections. Therefore, it is intuitive to reduce computational model complexity by removing the parameters themselves. Pruning can be learned during training using regularization or as a post-training step.

One training-based method group focuses on learning a pruning mask through auxiliary parameters during training. This kind of method aims at considering pruning as an optimization problem that tends to minimize both network supervised loss and penalty of auxiliary parameters that are multiplied with their corresponding parameter during the forward pass(Guo et al., 2016).

Another group of training-based methods apply various kinds of penalties to weights themselves to make them progressively shrink toward zero to enforce sparsity(Nowlan & Hinton, 1992). Various methods apply different regularizations on top of the weight decay to increase further the sparsity: using L_1 norm(Liu et al., 2017), LASSO (Least Absolute Shrinkage and Selection Operator)(Gao et al., 2019), Variational Dropout(Blum et al., 2015), etc.

Post-training pruning methods prune weights whose abso-

lute value is the smallest. One straightforward way to apply such pruning in a structured way is to remove neurons in linear layers or channels in convolutional layers depending on their norm (L_1 or L_2, for example)(Zhuang et al., 2018).

Weights in Low-Rank Format
Low-rank methods can replace weights of the neural network layer with its low-rank factorization. For example, linear layers or point-wise convolution (1×1) can be factorized using SVD into two sequential linear layers or point-wise convolutions. In this case, more information is preserved than with pruning (before fine-tuning). It is also possible to train a neural network with weights in low-rank format from scratch. Such an approach forces neural network parameters optimization over a smaller number of parameters.

Input in the considered KL-CPD model has a high-dimensional format (frames, channels, height, width). Therefore, it seems natural to apply further transforms using the Tensor Contraction Layer(TCL) and/or Tensor Regression Layer(TRL)(Kossaifi et al., 2020) rather than flattening output and using linear layers, for example, in GRU. In TCL, each input dimension is multiplied with its own matrix, resulting in fewer parameters and computational complexity for linear transformation. In TRL, the inner product between input and layer weight is calculated, allowing learning of more complex interconnections than in TCL.

3. Background. KL-CPD approach

Kernel-based methods remain one of the most popular and well-perform techniques for change detection. The idea of these methods is to compare two different partitions, e.g. two consecutive windows, in low-dimensional kernel space. The common way to detect a change-point is to utilize a two-sample test based on the maximum mean discrepancy or M-statistic between two distributions $\mathbb{P}$ and $\mathbb{Q}$ (Li et al., 2015):

$$M_k(\mathbb{P}, \mathbb{Q}) = ||\mu_{\mathbb{P}} - \mu_{\mathbb{Q}}||^2_{H_k} = \mathbb{E}_{\mathbb{P}}[k(x, x')] - \quad (1)$$
$$2\mathbb{E}_{\mathbb{P},\mathbb{Q}}[k(x, y)] + \mathbb{E}_{\mathbb{Q}}[k(y, y')], \quad (2)$$

where $\mu_{\mathbb{P}}$ and $\mu_{\mathbb{P}}$ are the kernel mean embedding for $\mathbb{P}$ and $\mathbb{Q}$. For the same distributions $\mathbb{P} = \mathbb{Q}$, we expect the statistic $M_k(\mathbb{P}, \mathbb{Q}) = 0$. As $k(\cdot, \cdot)$, ones usually consider reproducing kernel Hilbert space (RKHS).

The challenge question is how to choose the appropriate kernel family for the considered particular task if the available information about distribution after the disorder $\mathbb{Q}$ is narrowed. In the paper (Chang et al., 2018), the authors suggest approximating the distribution $\mathbb{Q}$ with auxiliary generative model $\mathbb{G}_\theta$ and using it for samples simulation. More specifically, they learn $\mathbb{G}_\theta$ to be in between $\mathbb{P}$ and $\mathbb{Q}$. At the same time, as we can generate as many samples as we want, we can train a neural model for deep kernel parametrization of well-known kernel family. The teaser described the overall method is shown in the Figure 2.

After the model training, the M-statistic based on learned kernels is used as anomaly score to detect changes in the sequence.

The obvious bottleneck of such an approach lies in its GAN-based nature. For both samples generator and discriminator, authors of the work apply autoencoders with RNNs in their cores. It works while operating with the multivariate sequences of small dimension, e.g. for human recognition activity, but make impossible the usage of streams with huge dimension on each time steps like video.

4. Methodology

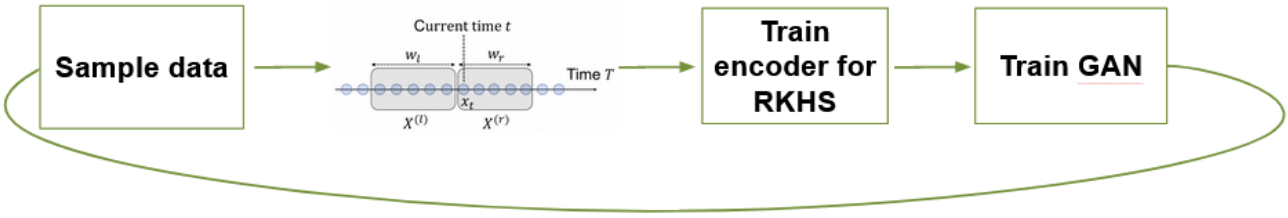
The vital drawback of applying KL-CPD to video anomaly detection is the huge input data dimension. Considered in this work input video data is fed into the model at 30 frames per second (fps), and each frame is a $240 \times 320 = 76800$ pixels image. Obviously, it is impractical to feed such data into the KL-CPD model directly as it leads to the large number of model parameters making the training of this model intractable or even not possible due to the limited amount of memory on GPUs. To this end, we propose several approaches to reduce the number of trainable parameters in the model and provide motivation behind each approach.

We separate our research into two main tracks: preliminary prepossessing input data to compress the data via common methods or proper encoder and general KL-CPD model compression.

In detail, the tasks on the first track are the following:

1. Provide reduction of input size with applying standard transformations like side scale, crop video, etc.
2. Perform dimension reduction for the data via Manifold Learning techniques (ISOMAP) as in (Nie et al., 2011).
3. Extract features from initial data by the pre-trained powerful network such as frame-wise applying of similar to ResNet 2D models (He et al., 2016) or models on 3D convolutions (Feichtenhofer et al., 2019).
4. Train our own DNN model to low-dimension representation of input data.

Although classical compression approaches are generally good from a perceptual point of view, their flawlessness is hidden among their generality. They are not optimal for specific tasks and don't consider the specificity of the CPD task. The important properties vary from type of video, e.g.,



$$\min_{\theta}\max_{\phi} \quad M_{f_\phi}(\mathbb{P}, \mathbb{G}_\theta) - \lambda \cdot \hat{M}_{f_\phi}(X, X') - \beta \cdot \mathbb{E}_{\nu \in \mathbb{P} \cup \mathbb{G}_\theta} \|\nu - F_\psi(f_\phi(\nu))\|_2^2.$$

Figure 2. The teaser of KL-CPD approach. Via GAN-based architecture, we approximate the true data distribution. Via encoder for RKHS, we optimize the parameters of the kernels.

for explosions, we expect the CPD model to monitor the intensity of white colours; however, it is not true for car accidents where patterns are much more comprehensive. While compression can not influence intensity, we can lose other essential properties for other types of change points. Similar reasons are suitable for standard transformations and operating with low-dimensional space. So, we should research how different methods affect the overall model's quality.

In experiments on model compression, we model weights of linear layers in low-rank format. Initially, we replace each linear layer in GRU modules in discriminator and generator networks (see Section3) with Tensor Contraction Layer (TCL) (see Figure 3). In this case, the input of size `(batch size, sequence length,` I_1, I_2, I_3`)` is multiplied with three matrices along the last three modes to obtain the output of size `(batch size, sequence length,` O_1, O_2, O_3`)`. It leads to reducing the parameter number in discriminator and generator networks. Later, we replace linear layer with Tensor Regression Layer (TRL) (see Figure 4). In this case, the input of size `(batch size, sequence length,` I_1, I_2, I_3`)` is multiplied with three matrices along the last three modes to obtain output of size `(batch size, sequence length,` R_1, R_2, R_3`)`, then it is contracted with weight's core of size $(R_1, R_2, R_3, R_4, R_5, R_6)$ and finally it is multiplied with three matrices along the last three modes to obtain the output of size `(batch size, sequence length,` O_1, O_2, O_3`)`.

Besides, we conduct experiments on linear layer pruning in discriminator. We train original model with additional ℓ_1 regularization on projection layers, find half of channels, that correspond to embedding dimension, with the lowest ℓ_1 magnitude and model their pruning with corresponding mask. After that we fine-tune masked model for 1 epoch to recover model performance.

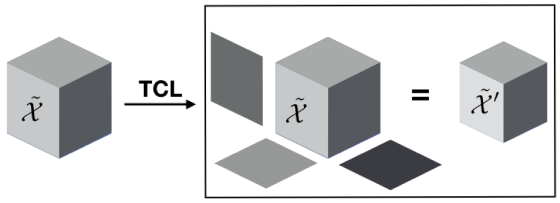


Figure 3. The intuition behind Tensor Contraction Layer (TCL)

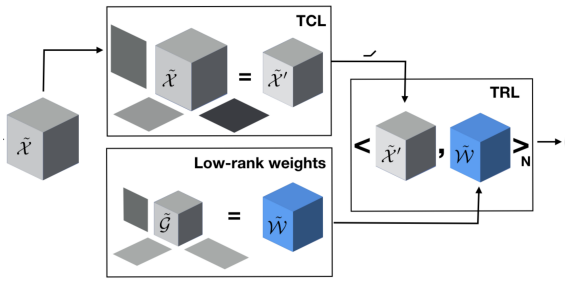


Figure 4. The intuition behind Tensor Regression Layer (TRL)

It's important to note that considered low-rank and pruning methods need accurate hyper-parameter selection. So, the model performance can depend on it.

4.1. Evaluation metrics

Following (Romanenkova et al., 2021), we use the same evaluation strategy for estimating the quality of applied tricks for video change point detection via KL-CPD.

Classification quality metrics. For True Positive (TP), we correctly detect changes no earlier than it appears. True Negative (TN) indicates that we rightly don't detect anything. False Negative (FN) defines a case when we miss the disorder while it appears in the real data. The most significant is the False Positive (FP). The obvious case of

FP is predicting changes for normal signals. In addition, we detect the FP when the model predicts change before it appears in a sequence with the real disorder. Based on these metrics, we calculate $F1 = \frac{TP}{TP+0.5(FP+FN)}$.

Delay detection and Time to False Alarm. We want to meet two competing goals: minimize the Detection Delay and maximize the Time to False Alarm. The Delay Detection $(\tau - \theta)^+$ is the difference between the true change point θ and the model's prediction of change point τ. The Time to False Alarm τ is the difference in time between a first false positive prediction if it exists and the sequence start.

Area under the detection curve. Varying s, we obtain different trade-offs between mean detection delay (x-axis) and mean time to a false alarm (y-axis) and get the detection curve. We want to find an approach that minimizes the area under this curve.

Model Complexity. We use FLOPs count to indicate the computational complexity of the model inference as it can be estimated theoretically, and it doesn't depend on the execution platform. Latency is a more realistic metric; however, it can be hard to reproduce the results even with the same hardware. To get the FLOPs count, we use the `fvcore` package from PyTorch, which works with a computational graph of the model.

The valuable remark is that our research also concentrates on reducing the model parameter number during not only the inference stage but also the training stage. We suppose that a less number of parameters can lead to more stable neural network training and, as a result, better performance.

5. Results

The code reproduced the following results is provided by link.

5.1. Video dataset details

Following (Romanenkova et al., 2021), we use a dataset with real-world 240×320 RGB explosion videos and markup provided in the article for our research. To generate the dataset for the CPD model, we cut videos on small clips with a given step and frequency. We use 30 frame per second, and the number of frames in the clip 16. For training data, we sample clips with step 5, so there are overlapping sections. For test data, we cut the video on non-overlapping clips. We keep these parameters constant for all our experiments, varying transformations that could be applied to the data (e.g. normalization, scaling etc).

5.2. Results on preliminary data preprocessing

The most simple approach to processing the video via the KL-CPD approach is to reduce the initial dimension of the input data. We presents the results the following preliminary data preprocessing techniques:

1. First of all, we consider a simple reduction of input size by resizing each frame of the video to a smaller one via common resizing from torchvision package.
2. We also evaluate the embedding of each frame via manifold learning methods (ISOMAP and Spectral Embedding),
3. Finally, we present a results for a custom feature extractor, namely CNN that is applied to each frame.

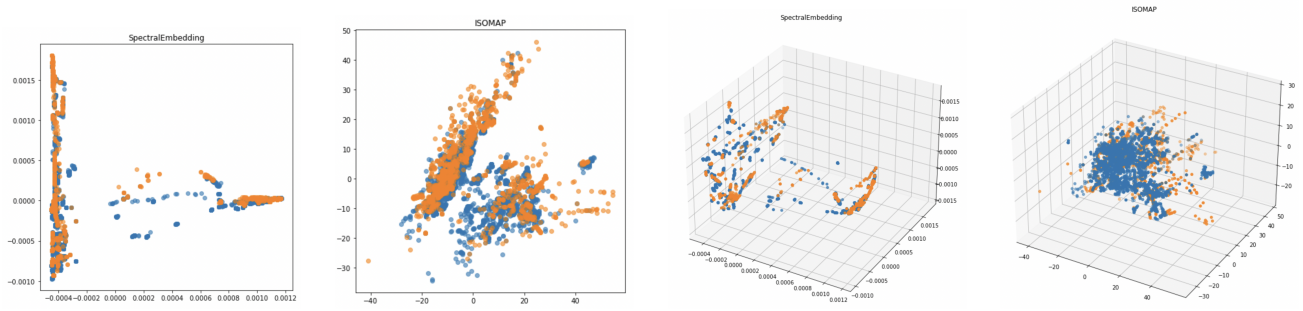
Resizing and Manifold learning. Currently, for all preprocessing experiments, we work on *frame level*, i.e. we apply the method to each frame of the input video. Then, the processed frames are stacked together to form a sequence of multidimensional vectors. The obtained sequence is the input in KL-CPD architecture. During *resizing*, we simply reduced the height by the ratio of target height and the width proportional to the reduction of the height.

For manifold learning experiments, we compress frames into 2D and 3D embedded spaces to reveal if separation can be made easily in those spaces. We evaluate the ISOMAP and SpectralEmbedding models fitted on the frames from the training dataset. Figure 5 shows the obtained low-dimensional representations. The blue points correspond to normal frames, and the orange one corresponds to frames with an explosion. As we can see, there is no tangible structure - "exploded" frames are well hidden beneath ordinary frames.

Using feature extractors. In these experiments, we follow the procedure described in the (Romanenkova et al., 2021) to run KL-CPD for the explosion dataset. In particular, we use 3D CNN (Feichtenhofer et al., 2019) as a feature extractor and add Linear layers with dimensions 100 to encoders and a decoder that makes an embedding size small enough to be stored and processed. We consider a set of pre-trained 3D CNN extractors with different parameters number denotes as *3D CNN XS, 3D CNN S, 3D CNN M or Original, 3D CNN L*. More details on the particularities of these models are provided by the link. The number of trained parameters remains the same as we freeze the layers of the feature extractor.

The considered custom CNN feature extractor has the following architecture:

- Conv2D with 3 input and 12 output channels, kernel



(a) 2D Spectral Embedding (b) 2D ISOMAP (c) 3D Spectral Embedding (d) 3D ISOMAP

Figure 5. Manifold learning methods' embedding of frames. Frames after the change point (explosion) are coloured with orange, the normal ones — with blue.

size equals 5, stride equals 2, and dilation equals 2 followed by ReLU activation;

- Conv2D with 6 output channels, kernel size equals 3, stride equals 2, and dilation equals 2 followed by ReLU activation and MaxPool2D over 3 cells;
- Finally, the output is flattened.

The results on experiments with preprocessing are summarized in Table 1. As one can observe, trainable CNN provides the best result among resizing approaches and using pre-trained extractors. The results from frame-wise ISOMAP compression look promising since they are better than plain resize and simultaneously require a ridiculously low number of parameters. However, compression of each frame does not fit PyTorch pipelines, thus, making such compression time demanding. Using pre-trained feature extractor are highly sensitive to choose of its type. It seems that the small model can't provide any reasonable insights for KL-CPD, while the large model produces complicated features that GAN-based KL-CPD fails to recover.

Table 1. Performance of KL-CPD after video data preprocessing.

Method	TN	FP	FN	TP	F1	# param
Resize 48	241	63	10	1	0.027	0.296M
Resize 128	238	64	10	3	0.075	0.296M
ISOMAP	206	95	4	10	0.168	0.069M
CNN	**290**	**11**	**6**	**8**	**0.485**	0.817M
3D CNN XS	0	315	0	0	0.000	4M
3D CNN S	281	20	8	6	0.300	4M
3D CNN M	285	16	8	6	0.333	4M
3D CNN L	0	315	0	0	0.000	4M

5.3. Results on Model Compression

In experiments on modelling layer weights in a low-rank format, we got models with a smaller number of parameters by tensor decomposition. Hence, we have to train compressed models for a more significant number of epochs. We tried two sets of embedding and hidden dimensions for TCL and one set for TRL. In general, using more trainable parameters usually give a better F1 score. In addition, using more parameters for bias usually increases the performance. It gives intuition that we train bias as some inner model state. TRL models more complex interactions and can achieve comparable performance without bias. To sum up, with TCL and TRL, it is possible to reduce the number of trainable parameters in discriminator and generator networks more than $5 - 10$ times with preserving and even increasing the model's quality, as presented in Table 2.

In experiments on pruning, decreasing embedding projection dimension in discriminator network by 90% results in $10\times$ model size reduction while F1 drops only for 16%, as presented in Table 3.

6. Discussions and Conclusion

We consider different approaches to overcome bottleneck problems in KL-CPD's original architecture to make it possible to detect changes in the video data. We show that model decomposition reduces the number of parameters that allow a researcher to train the model via limited GPU and increases the performance from the original F1 equals 0.3333 to 0.5714.

We've observed that decreasing the number has a beneficial effort on model performance during all of the experiments. We hypothesise that such behaviour happens due to the explosion video nature. For such a video, it's unnecessary to keep a lot of complex data information to detect changes. We expect that model will concentrate on changes in the mean of frames, ignoring other deviations. So, the simple model, the easier it is to detect primitive changes. We can apply the same logic for many other changes and train the principled sparsification for the CPD model in future research.

Table 2. Results on replacing linear layers with TCLs layers in GRU modules. # param denoted number of (trainable) parameters in discriminator and generator networks. bias rank = full means that for each linear layer bias isn't compressed, bias rank = R means that bias compressed with Canonical Polyadic Decomposition of rank R.

Model	Embedding size	Hidden size	Bias rank	# param, M	FLOPs, M	F1	AUC
Original	100	16	full	4	81.8	0.3333	1.8753
TLC	(32, 8, 8)	(16, 4, 4)	1	0.236	58.0	0.3636	0.7900
TLC	(32, 8, 8)	(16, 4, 4)	2	0.237	58.0	0.4848	0.6785
TLC	(32, 8, 8)	(16, 4, 4)	4	0.240	58.0	0.4118	0.7458
TLC	(32, 8, 8)	(16, 4, 4)	8	0.244	58.0	0.4444	0.8158
TLC	(32, 8, 8)	(16, 4, 4)	full	0.280	58.0	0.5333	0.6804
TLC	(64, 8, 8)	(32, 4, 4)	8	0.407	114.5	0.5714	0.6322
TLC	(64, 8, 8)	(32, 4, 4)	full	0.459	114.5	0.4348	0.7095
TRC	(32, 8, 8)	(16, 4, 4)	0	0.892	-	0.5263	0.7817

Table 3. Results on pruning projection layers in discriminator. # param denoted number of parameters in discriminator at inference time.

Model	Embedding size	Hidden size	# param, M	F1	AUC
Original	100	16	2.46	0.3333	1.8753
Pruned	50	16	1.23	0.3	1.0511
Pruned	10	16	0.25	0.2807	1.0866

References

Aminikhanghahi, S. and Cook, D. J. A survey of methods for time series change point detection. *Knowledge and information systems*, 51(2):339–367, 2017.

Ballé, J., Laparra, V., and Simoncelli, E. P. End-to-end optimized image compression. 2017.

Ballé, J., Minnen, D., Singh, S., Hwang, S. J., and Johnston, N. Variational image compression with a scale hyperprior. 2018.

Blum, A., Haghtalab, N., and Procaccia, A. D. Variational dropout and the local reparameterization trick. In *NIPS*, 2015.

Chang, W.-C., Li, C.-L., Yang, Y., and Póczos, B. Kernel change-point detection with auxiliary deep generative models. In *International Conference on Learning Representations*, Vancouver, Canada, 2018. ICLR.

Feichtenhofer, C., Fan, H., Malik, J., and He, K. Slowfast networks for video recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6202–6211, 2019.

Gao, S., Liu, X., Chien, L.-S., Zhang, W., and Álvarez, J. M. Vacl: Variance-aware cross-layer regularization for pruning deep residual networks. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 2980–2988, 2019.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Guo, Y., Yao, A., and Chen, Y. Dynamic network surgery for efficient dnns. In *NIPS*, 2016.

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

Kossaifi, J., Lipton, Z. C., Khanna, A., Furlanello, T., and Anandkumar, A. Tensor regression networks. *ArXiv*, abs/1707.08308, 2020.

Li, S., Xie, Y., Dai, H., and Song, L. M-statistic for kernel change-point detection. *Advances in Neural Information Processing Systems*, 28, 2015.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2755–2763, 2017.

Lu, G., Ouyang, W., Xu, D., Zhang, X., Cai, C., and Gao, Z. Dvc: An end-to-end deep video compression framework. volume 2019-June, 2019. doi: 10.1109/CVPR.2019.01126.

Nie, X., Liu, J., Sun, J., and Zhao, H. Key-frame based robust video hashing using isometric feature mapping. *Journal of Computational Information Systems*, 7, 2011. ISSN 15539105.

Nowlan, S. J. and Hinton, G. E. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4:473–493, 1992.

Oord, A. v. d., Li, Y., and Vinyals, O. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Pless, R. and Souvenir, R. A survey of manifold learning for images. volume 1, 2009. doi: 10.2197/ipsjtcva.1.83.

Romanenkova, E., Zaytsev, A., Zainulin, R., and Morozov, M. Principled change point detection via representation learning. *arXiv preprint arXiv:2106.02602*, 2021.

Sullivan, G. J., Ohm, J. R., Han, W. J., and Wiegand, T. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22, 2012. ISSN 10518215. doi: 10.1109/TCSVT.2012.2221191.

Sultani, W., Chen, C., and Shah, M. Real-world anomaly detection in surveillance videos. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6479–6488, Salt Lake City, Utah, USA, 2018. IEEE.

Toderici, G., O'Malley, S. M., Hwang, S. J., Vincent, D., Minnen, D., Baluja, S., Covell, M., and Sukthankar, R. Variable rate image compression with recurrent neural networks. 2016.

Toderici, G., Vincent, D., Johnston, N., Hwang, S. J., Minnen, D., Shor, J., and Covell, M. Full resolution image compression with recurrent neural networks. volume 2017-January, 2017. doi: 10.1109/CVPR.2017.577.

Truong, C., Oudre, L., and Vayatis, N. Selective review of offline change point detection methods. *Signal Processing*, 167:107299, 2020.

Wallace, G. K. The jpeg still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38, 1992. ISSN 00983063. doi: 10.1109/30.125072.

Wiegand, T., Sullivan, G. J., Bjøntegaard, G., and Luthra, A. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13, 2003. ISSN 10518215. doi: 10.1109/TCSVT.2003.815165.

Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., and Zhu, J.-H. Discrimination-aware channel pruning for deep neural networks. In *NeurIPS*, 2018.