Deep Learning Workshop - Tabular data classification using CNN

Shaked Caspi and Sagi Polaczek

Computer Science Department, Tel-Aviv University, Israel.

Contributing authors: shakedcaspi@mail.tau.ac.il; sagipolaczek@mail.tau.ac.il;

Abstract

Deep learning algorithms revolutionised problem solving in homogeneous data such as images, videos, sound and text [1–3]. Despite these advances, deep learning is relatively less successful in solving problems related to heterogeneous data including tabular data [4]. In this work we aim to use a well proven deep learning concept, *convolutional neural networks* (CNN) [5], to classify numerical tabular data. We investigated two main ways to do so, both rely on the idea of transforming tabular samples into images and then using deep learning algorithms to perform classification. In the first approach we want to take advantage of a CNN property - extraction of spatial features. By using *Autoencoder* [6] on the original tabular sample, we wish to generate an encoded image with enhanced spatial features. The second approach involves transforming samples into kernels and then performing a convolution between a static base image and the kernel. This algorithm is known as TAC [7]. Note that in both methods we put our focus on the reduction (from tabular sample to an image) itself. Once the reduction is done properly, we can theoretically use any algorithm for image classification.

Keywords: Tabular data, Deep neural networks, Convolutional neural networks , Autoencoding

1 Introduction

One of the most common ways to present data is in a *tabular manner*. Tabular data is common in various domains including medicine, finances, and

marketing [8]. Deep learning algorithms revolutionised problem solving for homogeneous datasets but are relatively less successful in solving problems related to tabular data [4]. One of the main reasons is that tabular datasets contain heterogeneous data presentations such as dichotomous features, categorical features or continuous features. In addition the correlation among the features is sometimes weak in contrast to homogeneous data [9]. Today, to the best of our knowledge, there is no a deep learning model that able is to outperform classical machine learning models on tabular data in general. Models that are based on gradient boosted decision trees, like XGBoost [10], are the current leaders for solving these types of problems. In recent years there is an effort to promote deep learning models for tabular data. Some of these models rely on special hybrid architectures. For example, NODE [11] uses classical machine learning techniques with neural networks [4]. Another popular approach is transformer based models, like TabNet [12]. In this work we focus on applying a transformation from tabular data to a homogeneous domain, such as imaging. By doing so we were able to use various tools to examine tabular data related tasks.

2 Methods

In the following section we wish to describe our proposed methods for applying a reduction from tabular data to images. Please note that while the first method 2.1 is our original novel approach, the second one 2.2 is a POC of an existing one.

2.1 Autoencoder

In this method we aim to examine a novel reduction between tabular data classification task into an image classification one. As stated in the diagram (refer Figure 1) we use an autoencoder for the declared reduction. Traditionally, encoding by an autoencoder is being produced as a result of a *bottleneck* between the encoder's input and the decoder's output [6]. In our approach, due to size limitations, we would like to replace this bottleneck by forcing a loss function on the encoded image. Then the encoded image is being fed into an "off the shelf" untrained ResNet model [1]. The architecture consists three different losses in total, that are being optimize in parallel (see 2.1.2).

2.1.1 Architecture

As stated before, we separate the encoder and decoder with a custom loss function. Both the encoder and the decoder are identical and consist of four convolutional layers that each has a kernel of size 3, stride of size 1, and padding of size 1. As a result, the resolutions of the encoder's input and decoder's output are the same. Note that while in our architecture we encode the image to a tensor with 3 channels to nicely fit the ResNet, it can be set to any other amount.

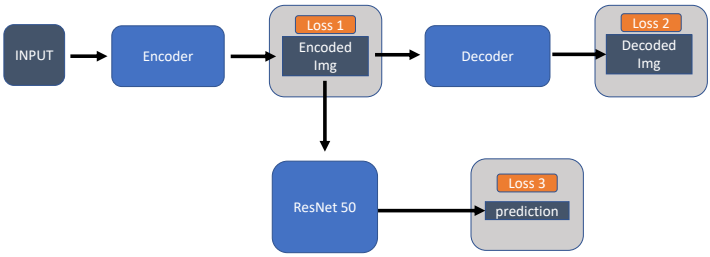

Fig. 1 The model architecture that we used in the Autoencoder method. The input is a 2D tensor as a result from reshaping the tabular sample vector.

2.1.2 Losses

In this section we wish to elaborate more on the three loss functions that we use in the architecture. Let us follow Figure 1 naming (Loss 1,2,3):
Loss 1 A loss function that is being applied on the encoded image, aiming to take the role of the bottleneck. Different loss functions may force the encoder to behave differently by compelling it to certain criteria. In our approach we use that property to give the encoded image a *spatial meaning*.
Loss 2 A reconstruction loss function, which measures the distance between the input of the encoder and the output of the decoder, thus minimizing this loss function cause the encoder to preserve it's original input data. A common choice is to use the mean squared error (MSE) loss function.
Loss 3 A straight forward classification loss function, measures the distance between the model's class prediction and the label's one.

2.2 TAC (Proof Of Concept)

One of the ways to transform tabular data to images is by using a smart algorithm that transforms each data point to an image based on its features. To do so we use the TAC algorithm [7]. We mark the number of features in the table as N^2. The algorithm now transforms each data point to an image in the size of $N \times N$ called a kernel. If the number of features doesn't have an integer square root we need to increase or decrease the size of it, depends on which is the nearest integer and the properties of the data. To increase the size we can simply use padding, and to decrease the number of features we can perform feature selection over the features based on the training set. Afterward, the process of creating the kernel is simple and consists of two stages.

1. Reshaping the vector.
2. Subtracting the mean value of the vector from each pixel.

In addition, we chose in an independent manner a large permanent image called the base image. Performing an image convolution of the kernel on the base image yields a final image for each data-point the in table. Finally, we use "off the shelf" untrained ResNet model [1] to perform the classification.
As we stated above, this method is a POC of a the TAC algorithm that was described by Buturovic , L., Miljkovic, D [7] .We could not find an implementation of it, so we implemented it ourselves. Our main goal was to show a proof of feasibility and implementing this approach as an open-source contribution for further research projects.

2.2.1 Limitations

While working on the TAC method we encountered two main limitations.

1. The kernel needs to be small. When using a dataset with many features, we will need to decrease the amount of features in order to apply this method. This can be done by using a feature selection technique. In some cases it means reducing the amount of information used for the classification, which implies a loss of data.
2. Transforming all the dataset to images takes a lot of resources (memory).

3 Experiments

3.1 Autoencoder

For this method we used the EPSILON dataset [8]. We chose this dataset because it has a relatively large number of samples (400k for training and 100k for testing) and each sample is represented by 2000 numerical features, which allows us to easily reshape a sample into a small size image: we pad with 25 zeros, and then get a 2025 pixels that reshape nicely to a 1x45x45 image.
For the loss functions 2.1.2 we used:
Loss 1 As stated before, we wish to replace the traditional autoencoder bottleneck with a custom loss function, that in our case, will enhance the spatial features of the input image so the CNNs will be able to take advantage of it. We chose three different custom loss functions, all of them rely on the notion of the standard deviation (STD):

$$\sigma = \sqrt{\frac{\sum (X-\mu)^2}{N}} \tag{1}$$

The lower the STD, the lower the variation in the given measured area. Thus by minimizing the STD, while minimizing the autoencoder reconstruction loss, we wish to force spatial connection between the encoded image's pixels.
This leads us to the following three experimental loss functions:

1. **FULL**: The STD of the full encoded image.

2. **DISJOINT**: The sum of the STD of the encoded image's *disjoint* patches. Each patch with a shape of 3x5x5 when 3 is the number of channels and 5 is the height and width.
3. **OVERLAP**: Same as DISJOINT, but now each patch has an overlap of 2x5 pixels with each of it's neighbours patches.

Loss 2 For the autoencoder reconstruction we use the MSE loss function between the encoder's input image and the decoder's output image. Note that as stated before, we don't change the image resolution in the encoding and decoding process.
Loss 3 For the classification we used the cross-entropy (CE) loss function.

The target loss function we want to optimize is a weighted sum of the three losses, or in other words, we are solving a multi-task learning problem. While there are several approaches (see 4.1) we used a static linear sum of the three loss function:
Let us denote by L_i, W_i the i's loss function and it's corresponding weight respectively. Then our target loss function is:

$$\mathcal{L} = W_1 \cdot L_1 + W_2 \cdot L_2 + W_3 \cdot L_3 \tag{2}$$

Training Data and Parameters
In the two following experiments we used Adam optimizer with a learning rate of 1e-4 and a weight decay of 1e-3.
In addition, we split the training data into balanced folds in a way that 20% of it goes to validation. At each epoch we save the model which preformed best on the validation data by the AUC metric.

3.1.1 W1 vs W100

As stated above, we aim to optimize the weighted sum of the three losses 2. While we've examined several variations for the weights W_1, W_2, W_3, it came down to the following experimental question:

> How large W_1, the encoding loss function, should be relatively to W_2 and W_3 so the model will be able to generalize better?

To answer this question we compared the validation AUC graphs of two models, both of them rely on the FULL 1 loss function and in both of them $W_2, W_3 = 1$. The models differ in their encoding's loss weight - in one of them $W_1 = 1$ while in the second $W_1 = 100$.
As we can see in Figure 2, $W_1 = 100$ outperforms it's rival on the validation AUC metric across all the epochs. We can assume it is because a higher weight on the encoding loss causes the encoder to encode the images with a more spatial meaning.

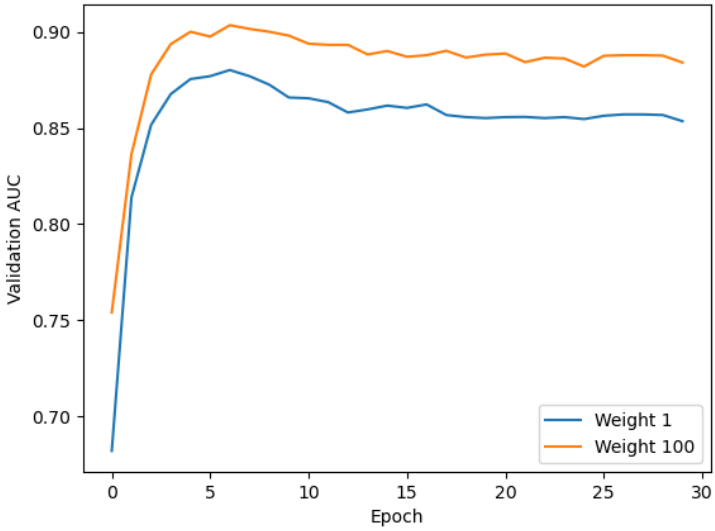

Fig. 2 A comparison between two models, each of them used the FULL 1 loss function. One of them allocate to the encoding loss a weight of 100, while the other a weight of 1.

3.1.2 FULL vs DISJOINT vs OVERLAP

In this experiment we wish to compare between the three different encoding loss functions we suggested above 1. As a conclusion from the previous experiment, we set $W_1 = 100$ for the three cases.
This time we made three comparisons:

- **Validation AUC 3** We compared the three models with their validation AUC across the first 30 epochs. In this comparison we couldn't see any supremacy of one of the models over the others.
- **Test AUC 1** Now we compare how the three models generalize on the test set, which as stated above, contains 100k samples. In this comparison it seems that the FULL loss function is inferior to the DISJOINT and the OVERLAP loss functions.
- **CE loss 2** Lastly, we compared our results to the ones in the paper *Tabular Data: Deep Learning is Not All You Need* [8]. Our approach still lacks behind the **SOTA** models.

3.2 TAC

We have two main goals for this experiment; firstly, to validate TAC as a learning algorithm. Secondly, one was to implement an unimplemented method and to contribute the open source community for future endeavors in this

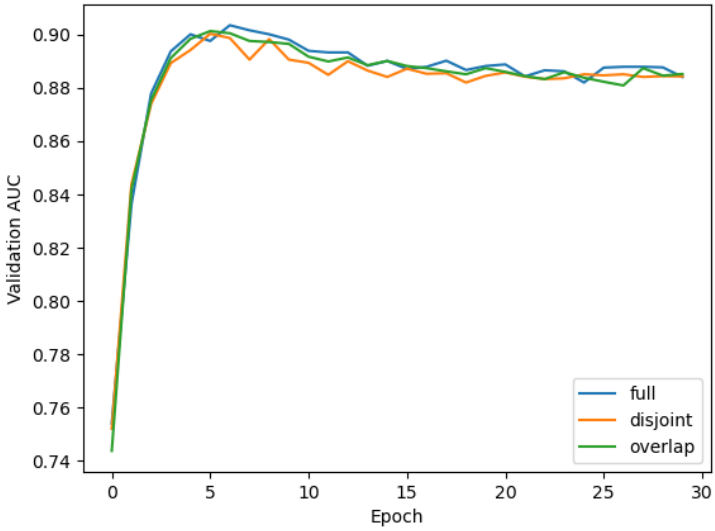

Fig. 3 Comparing between three different models that differ in the encoding loss type - FULL, DIJOINT, OVERLAP.

Table 1 The test AUC of the three models. As stated above, the model that we use for the test data is the one which preformed the best on the validation AUC.

Full	Disjoint	Overlap
0.6945	0.8402	0.8997

Table 2 Cross-Entropy loss metric. XGBoost model is still far superior to our approach.

XGBoost	Full	Disjoint	Overlap
0.0010	0.4166	0.4246	0.4218

XGBoost resource: [8]

direction. For this experiment we used the HIGGS dataset [13]. We chose this dataset because it has a relatively small number of features (28), which allows us to easily reshape each input to small size kernel. To create 5×5 kernel we need to delete 3 features using the feature selection technique implemented in the Scikit-Learn package [14]. Because we didn't have enough resources (mainly memory for caching) we decided to use a smaller subset (10k for training and 2.5k for testing). During training we used

- Adam optimizer with learning rate of 1e-4 and a weight decay of 1e-3
- Specific base image (refer appendix B1)

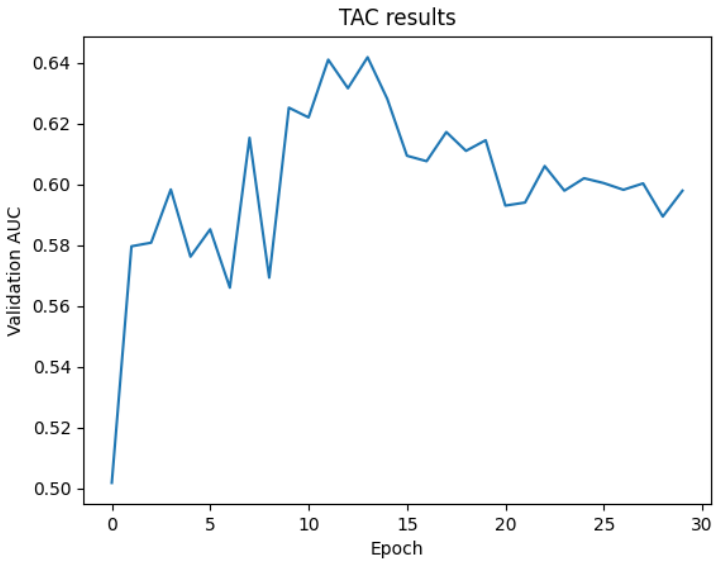

Fig. 4 Validation AUC of TAC over 30 epochs

The results of this experiment can be seen in Figure 4. The peak AUC validation was 0.6418 on epoch number 10 but around epoch 15 the model started to over-fit the train data (refer Figure 5) which resulted in reduction of the validation AUC. Overall, we can see that the model is indeed a learning one but the learning curve was smaller than expected probably due to the simplicity of the transformation and the small subset of the dataset that we've used.

4 Future Directions

We split this section into subsections according to our methods.

4.1 Autoencoder

1. First of all, we think that the results might get better with a more comprehensive hyper-parameters tuning (grid search for example) - which unfortunately we did not have the time and resources to execute.
2. In our approach we aimed to optimize a static linear combination of the three loss functions. A more suphicticated approach will be to find the *Pareto Optimal Solution* [15, 16], or to use dynamic weights which will adapt in the training time.

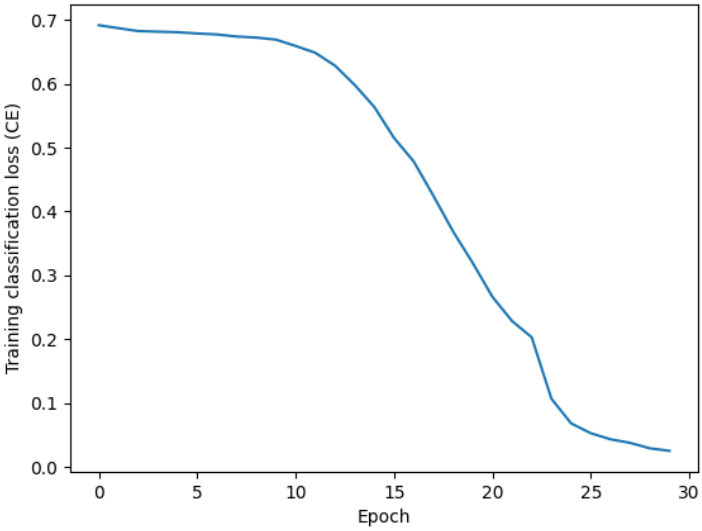

Fig. 5 Training cross-entropy loss of TAC over 30 epochs

3. Last but not least, there is plenty of room to play with the encoding loss function. While we based our functions only on variations of STD, the encoding loss can be easily modified to any other function. For instance, a MSE loss from a static image (such as B1).

4.2 TAC - POC

1. Using better resources will enable us to train over more samples which we assume will increase the results.
2. Improving the transformation of each input to a kernel, for example rearrange the pixels to preserve spatial structure. This can be done by deciding on a similarity function and evaluating it over the features.
3. Normalizing the kernel before subtracting the mean value.
4. Learning the base image from the features using a neural network.

Appendix A Source Code

Our code [17] is based on the FuseMedML open-source library [18]. FuseMedML is a python framework accelerating Machine Learning based discovery in the medical field by encouraging code reuse. The library based on PyTorch Lightning and also provides key components for the data pipelines, caching and a standalone library for evaluation. While working on the workshop we were able to contribute back to the library and the open-source community.

Appendix B TAC base image

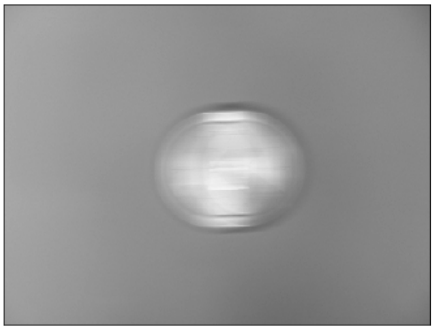

Fig. B1 Took from scikit-image.

References

[1] He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. arXiv (2015). https://doi.org/10.48550/ARXIV.1512.03385. https://arxiv.org/abs/1512.03385

[2] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention Is All You Need. arXiv (2017). https://doi.org/10.48550/ARXIV.1706.03762. https://arxiv.org/abs/1706.03762

[3] Oord, A.v.d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K.: WaveNet: A Generative Model for Raw Audio. arXiv (2016). https://doi.org/10.48550/ARXIV.1609.03499. https://arxiv.org/abs/1609.03499

[4] Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., Kasneci, G.: Deep Neural Networks and Tabular Data: A Survey. arXiv (2021). https://doi.org/10.48550/ARXIV.2110.01889. https://arxiv.org/abs/2110.01889

[5] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (2017). https://doi.org/10.1145/3065386

[6] Bank, D., Koenigstein, N., Giryes, R.: Autoencoders. arXiv (2020). https://doi.org/10.48550/ARXIV.2003.05991. https://arxiv.org/abs/2003.05991

[7] Buturovic´, L., Miljkovic, D.: A NOVEL METHOD FOR CLASSIFICATION OF TABULAR DATA USING CONVOLUTIONAL NEURAL NETWORKS. bioRxiv (2020). https://doi.org/10.1101/2020.05.02.074203. https://www.biorxiv.org/content/10.1101/2020.05.02.074203v1.full.pdf

[8] Shwartz-Ziv, R., Armon, A.: Tabular Data: Deep Learning is Not All You Need. arXiv (2021). https://doi.org/10.48550/ARXIV.2106.03253. https://arxiv.org/abs/2106.03253

[9] Somepalli, G., Goldblum, M., Schwarzschild, A., Bruss, C.B., Goldstein, T.: SAINT: Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training. arXiv (2021). https://doi.org/10.48550/ARXIV.2106.01342. https://arxiv.org/abs/2106.01342

[10] Chen, T., Guestrin, C.: XGBoost. ACM (2016). https://doi.org/10.1145/2939672.2939785. https://doi.org/10.1145.2F2939672.2939785

[11] Popov, S., Morozov, S., Babenko, A.: Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data. arXiv (2019). https://doi.org/10.48550/ARXIV.1909.06312. https://arxiv.org/abs/1909.06312

[12] Arik, S.O., Pfister, T.: TabNet: Attentive Interpretable Tabular Learning. arXiv (2019). https://doi.org/10.48550/ARXIV.1908.07442. https://arxiv.org/abs/1908.07442

[13] Baldi, P.S. P., Whiteson, D.: Searching for Exotic Particles in High-energy Physics with Deep Learning (2014). https://archive.ics.uci.edu/ml/datasets/HIGGS

[14] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

[15] Sener, O., Koltun, V.: Multi-task learning as multi-objective optimization. Advances in neural information processing systems **31** (2018)

[16] Ngatchou, P., Zarei, A., El-Sharkawi, A.: Pareto multi objective optimization. In: Proceedings of the 13th International Conference On, Intelligent Systems Application to Power Systems, pp. 84–91 (2005). https://doi.org/10.1109/ISAP.2005.1599245

[17] Sagi Polaczek, Shaked Caspi: Deep Learning Workshop - Tabular Data Classification Using CNN (Source Code). https://zenodo.org/badge/latestdoi/483145134

[18] IBM Research, I.: FuseMedML: https://github.com/BiomedSciAI/fuse-med-ml. Zenodo. https://doi.org/10.5281/zenodo.5146491 (2021). https://doi.org/10.5281/ZENODO.5146491. https://zenodo.org/record/5146491