Feature Convolutional Networks

He Hu HEHU@RUC.EDU.CN
Information School, Renmin University of China
DEKE of MOE, Beijing, China
#59 Zhongguancun St., Haidian Dist., Beijing, China

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

Convolutional neural networks are among the most successful deep learning models used for image processing, computer vision and natural language processing applications. In this paper, we define convolution operator for numerical tabular features and thus propose feature convolutional network model for machine learning tasks. Feature convolutional networks contain feature convolution layer to extract pairwise feature convolutions in the relational feature spaces. Compared with the baseline multi-layer neural network model, the feature convolutional network gains better performance among all the experiments. The experiments results suggest that feature convolutional networks can generate efficient features automatically and provide better performance through automatic feature learning. The demo code is at https://github.com/info-ruc/FeatConvNet.

Keywords: Feature Convolution; Feature Engineering; Feature Generation

1. Introduction

Convolutional neural networks (CNNs) Lecun et al. (1998) have been very successful in many applications, especially in image classification and object tracking field. Convolutional neural networks are achieving state-of-the-art results across a range of difficult problem domains, such as object classification and detection in photographs, or automatic colorization of black and white images. Over the past decade, many powerful convolutional neural network models for image classification have been proposed, such as AlexNet Krizhevsky et al. (2017), GoogLeNet Szegedy et al. (2015), and ResNet He et al. (2016). Nowadays the CNNs models achieve performance near or better than human beings.

Convolution operation involves applying a convolution kernel (or a filter) to a spatial region of an image. The kernel is of fixed size and maps the pixels to a smaller dimension, for example, by multiplying the corresponding pixel values and summarizing the values within the kernel size. Kernels used this way are a traditional feature extraction method in image processing. They are also very popular with text data based on the Transformer model Vaswani et al. (2017). Kernels perform feature extraction, going from input images or texts to feature map outputs as the kernel slides over the input image or textual data.

On the other hand, tabular (or relational) data is used extensively in machine learning tasks. This brings up the question: Can we transfer convolution operator to tabular data too? Will this transformation be beneficial to us? To answer these questions, we first define convolution operator for numerical tabular features and propose feature convolutional network model for machine learning tasks. We compare feature convolutional network model with the baseline model (multi-layer perceptron) and find out that the feature convolutional network performs constantly better than the baseline model on the famous machine learning dataset (Iris Data Set) Web (2021).

The motivation of the paper is inspired from convolution operator, which is used successfully in Euclidean (CNN) and non-Euclidean spaces (GCN). To our best knowledge, until now there are no public research studying convolution operation for the tabular (relational) data, which is very popular in ML community. Our paper is the first research attempt trying to answer the question: Is convolution also useful for tabular (relational) data? We find the answer is YES, the Feature Convolution Networks definitely improve the performance of the ML task. The study encourages further studies for convolutional tabular features engineering and tabular-spatial mixture handling.

2. Related Works

2.1. Convolution Operator

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, text, or audio signal inputs. They have three main types of layers, which are convolutional layer, pooling layer and fully connected layer.

The convolutional layer is the core building block of a convolutional neural network, As we can see in Fig. 1, each output value in the feature map $\phi(\mathbf{x})$ does not have to connect to each pixel value in the input image X. It only needs to connect to the receptive field, where the filter kernel W is being applied. Since the output array does not need to map directly to each input value, convolutional (and pooling) layers are commonly referred to as ”local connectivity”.

Given a kernel of size $n \times m$, the output feature map is defined as:

$$\phi(\mathrm{x}) = \sum_{j=1}^{\mathbf{n}} \sum_{i=1}^{\mathbf{m}} \mathbf{X}_{i,j} \boldsymbol{W}_{\boldsymbol{i},\boldsymbol{j}}$$

2.2. Feature Transformations

Machine learning models take features as input and make predictions as output. Features are numeric representation of various aspects of raw data, which sit between data and models in the machine learning pipeline. Good features make the subsequent modeling step easy and the resulting model more capable of completing the ultimate task.

It is often seen in machine learning applications when two features combined through an arithmetic operation becomes more significant in explaining variances in the data, than the same two features separately. Creating a new feature through interaction of existing features is known as feature transformations. One of the most important transformations is the cross-product transformation Cheng et al. (2016), which is defined as:

$$\phi_k(\mathbf{x}) = \prod_{i=1}^{d} x_i^{c_{ki}} \quad c_{ki} \in \{0, 1\}$$

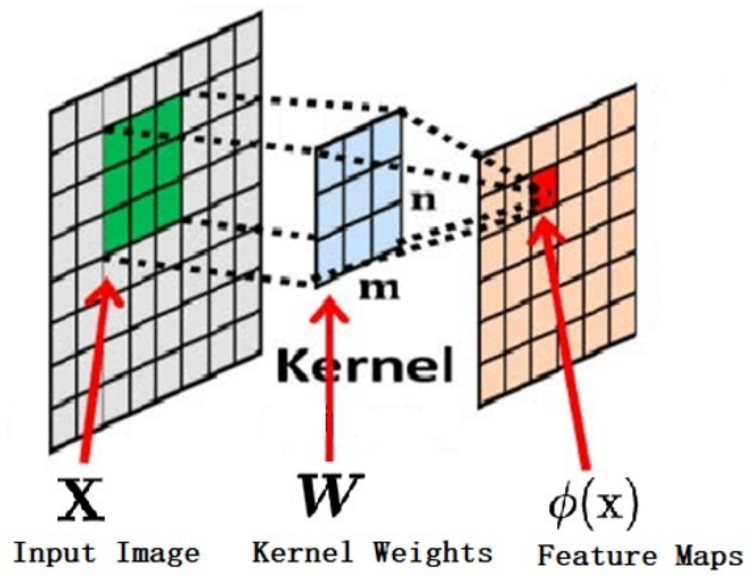


Figure 1: Image Convolution Operator

where c_{ki} is a boolean variable that is 1 if the i-th feature is part of the k-th transformation phi_k, and 0 otherwise. a cross-product transformation captures the interactions between the binary features, and adds nonlinearity to the generalized linear model.

3. Feature Convolution Networks

3.1. Feature Convolution Operator

Inspired by convolution operator and feature transformation techniques, we define 2-way feature convolution operator on tabular data:

$$\phi_{ij}(\mathbf{x}) = \alpha_i \mathbf{x}_i + \beta_j \mathbf{x}_j$$

As illustrated in Fig. 2, we have a tabular data containing M observations and N features. in this paper, we only consider convolutions between two features (2-way convolutions). K-way convolution contains K features in the process of feature convolution and produce C_N^K Feature Maps.

We have one kernel parameter (α and β) for each feature in the convolution operation. Like in convolutional neural network, the kernel parameter is shared on all observations of the corresponding feature. We only consider two features in convolution; this characteristic can be described as 'local connectivity' of convolutional neural networks. That is, we only consider a neighborhood of two features. For a tabular data containing N features, we will get C_N^2 feature maps.

The feature convolution algorithm is provided in algorithm. 1. For n input features and data X, we first initialize α and β as $C_n^2 \times 1$ random values. Then we generate C_n^2 feature

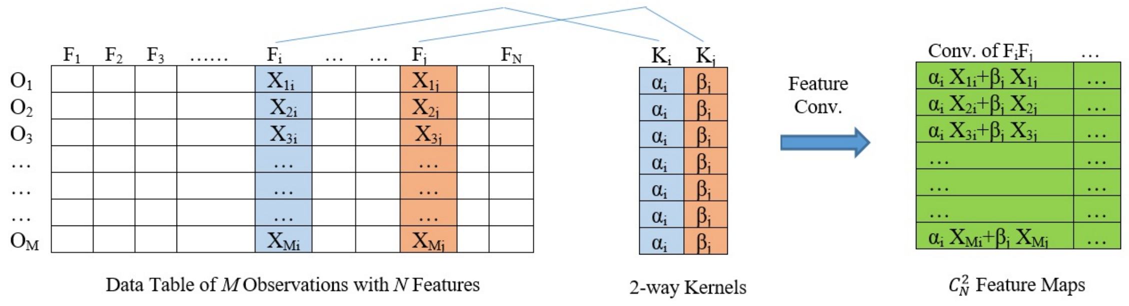


Figure 2: Feature Convolution Operator

pairs and calculates the feature convolution of i and j as $\alpha_k \times X_i + \beta_k \times X_j$, where k is from 1 to C_n^2. Then the calculated feature convolutions are all added to the output result: ϕ. The algorithm output the ϕ as feature convolution result, note that the value ϕ is added with X, the original features, before it is returned. That means the feature convolution will result in a set that contains both original features and the convoluted features.

```
Algorithm 1: FeatureConvolution
Input: $n$, $X$
Output: $\phi$
$\phi \leftarrow []$; $s \leftarrow C_n^2$; $k \leftarrow 1$
$\alpha(s) \leftarrow rand(0,1); \beta(s) \leftarrow rand(0,1)$
for $i \leftarrow 1; i \leq n; i++$ do
    for $j \leftarrow i+1; j \leq n; j++$ do
        $\nu_k \leftarrow \alpha_k \times X_i + \beta_k \times X_j$
        $\phi \leftarrow \phi \bigcup \nu_k$
        $k \leftarrow k+1$
    end
end
$\phi \leftarrow \phi \bigcup X$
return $\phi$
```

3.2. Features Normalization

Features in isolation can be difficult to compare with each other. Neural network algorithms perform simple mathematical operations with them that assume their values are comparable. That means that if two features are of widely different dimensions, the machine learning algorithm will require training data to learn to scale them accordingly. That unnecessary burden can be alleviated by normalizing the features in various forms. Feature normalization is a great way of reducing the variations on feature values, and it ensures that the feature values fall within a specific range during training. For example, we can center the variable at zero and rescale the distribution to the value range. This procedure involves subtracting the mean from each observation and then dividing the result by the difference between the minimum and maximum values. This is crucial for neural networks that need input data scaled to certain specific intervals.

Models that involves a matrix are affected by the scale of the input and are sensitive to the scale of input features. Feature normalization changes the scale of the feature and is usually done individually to each feature. In our experiments, features normalization apply every time data is forward propagated through the feature input layer, specified as one of the following:

- 'zerocenter' — Subtract the mean from each of feature dimension.
- 'zscore' — Subtract the mean and divide by standard deviation from each of feature dimension.
- 'symmetric' — Rescale the input to be in the range [-1, 1] using the minimum and maximum values, respectively.
- 'zero-one' — Rescale the input to be in the range [0, 1] using the minimum and maximum values, respectively.

4. Experiments

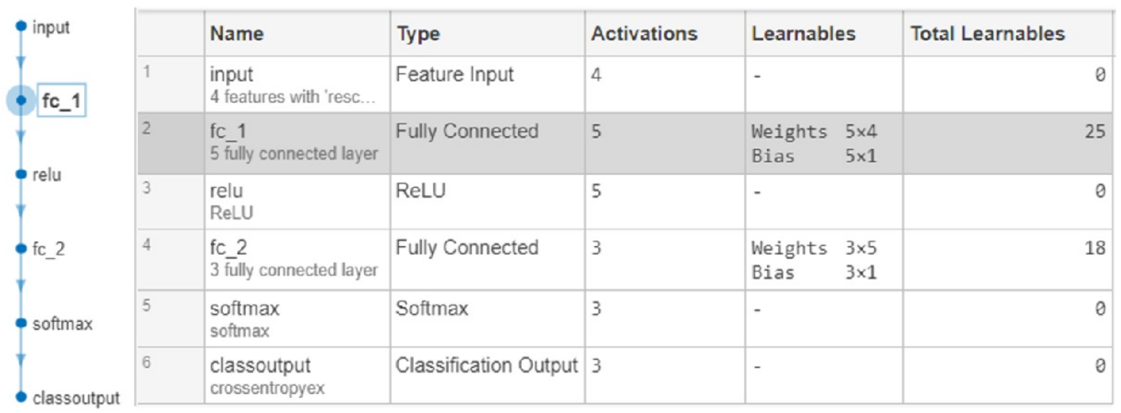
4.1. Dataset

We use Iris Data Set Web (2021), this is perhaps the best-known database to be found in the machine learning literature. This dataset contains 150 observations with four input features representing the parameters of the plant and one categorical response representing the plant species. Each observation is classified as one of the three species: setosa, versicolor, or virginica. Each observation has four measurements: sepal width, sepal length, petal width, and petal length. We set aside 15% of the data for validation and 15% for testing, and split the data into training, validation and test sets.

4.2. Models

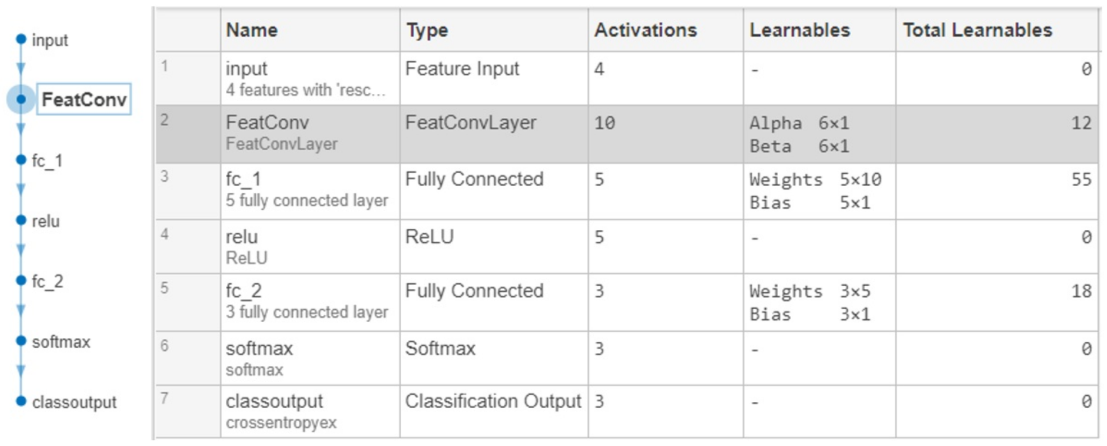
We create a baseline model with multi-layer perceptron. The baseline model is illustrated in Fig. 3. The model contains a featureInputLayer, a fullyConnectedLayer with five hidden units, a reluLayer, a fullyConnectedLayer with three hidden units corresponding to three classes, followed by a softmaxLayer and a classificationLayer. The feature input layer accepts iris data set containing numeric scalars representing features.

We add feature convolution layer to the baseline model to create the feature convolutional network. The feature convolutional network model is illustrated in Fig. 4. The model contains a featureInputLayer, a feature convolution layer with four features input, a fullyConnectedLayer with five hidden units, a reluLayer, a fullyConnectedLayer with three hidden units corresponding to three target classes, followed by a softmaxLayer and a classificationLayer. There are two parameters in feature convolutional layer: α and β. We can see in Fig. 4 α and β have size of 6×1 , because $C_4^2 = 6$. In general, for n features, α and β will have size of $C_n^2 \times 1$.



	Name	Type	Activations	Learnables	Total Learnables
1	input 4 features with 'resc...	Feature Input	4	-	0
2	fc_1 5 fully connected layer	Fully Connected	5	Weights 5×4 Bias 5×1	25
3	relu ReLU	ReLU	5	-	0
4	fc_2 3 fully connected layer	Fully Connected	3	Weights 3×5 Bias 3×1	18
5	softmax softmax	Softmax	3	-	0
6	classoutput crossentropyex	Classification Output	3	-	0

Figure 3: Baseline Model, Multi-layer Perceptron



	Name	Type	Activations	Learnables	Total Learnables
1	input 4 features with 'resc...	Feature Input	4	-	0
2	FeatConv FeatConvLayer	FeatConvLayer	10	Alpha 6×1 Beta 6×1	12
3	fc_1 5 fully connected layer	Fully Connected	5	Weights 5×10 Bias 5×1	55
4	relu ReLU	ReLU	5	-	0
5	fc_2 3 fully connected layer	Fully Connected	3	Weights 3×5 Bias 3×1	18
6	softmax softmax	Softmax	3	-	0
7	classoutput crossentropyex	Classification Output	3	-	0

Figure 4: Feature Convolutional Network Model

Table 1: Training Processing of Baseline Model without Normalization

Epoch	Mini-batch Acc.	Mini-batch Loss	Validation Acc.	Validation Loss
1	20.00%	1.8573	27.27%	1.6829
8	60.00%	0.8254	68.18%	0.8416
15	80.00%	0.5743	72.73%	0.5972
22	86.67%	0.4695	81.82%	0.4968
29	66.67%	0.4588	72.73%	0.4630
30	80.00%	0.4393	86.36%	0.4385

During training, the feature convolution layer iteratively performs forward and backward passes through the network. When making a forward pass through the network, the layer takes the outputs of the previous layers, applies a forward function, the forward function will add C_n^2 convolution features to the results, and then outputs (forward propagates) the results to the next layers. At the end of a forward pass of the network, the output layer calculates the loss L between the predictions Y and the true targets T.

During the backward pass of a network, each layer takes the derivatives of the loss with respect to the outputs of the layer, computes the derivatives of the loss L with respect to the inputs, and then backward propagates the results. The feature convolution layer has learnable parameters: α and β . Given a train set of inputs and expected outputs, an error metric and a given configuration of learnable weights, the network can be evaluated and its gradients with respect to the errors computed. The gradients can then be employed to change the learnable weights (α and β).

The following Fig. 5 describes the flow of data through a deep neural network and highlights the data flow through feature convolution layer with a input X, an output Z, and a learnable parameter W. We implement the models with Matlab custom layer function Matlab (2021), and the demo code is shared at: https://github.com/info-ruc/FeatConvNet.

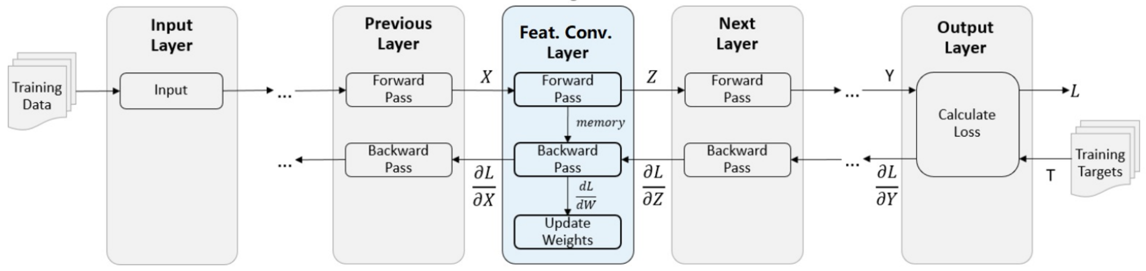


Figure 5: Intermediate Feature Convolutional Layer Architecture

We collect training data of baseline model in Tab. 1. The table shows the training process of baseline model without feature normalization. When the training is finished, the mini-batch accuracy hits 80.00%, with the loss drops to 0.4393 ; and the validation accuracy hits 86.36%, with the corresponding loss drops to 0.4385 .

Table 2: Training Processing of FeatConvNet Model without Normalization

Epoch	Mini-batch Acc.	Mini-batch Loss	Validation Acc.	Validation Loss
1	13.33%	1.4584	27.27%	1.2585
8	53.33%	0.7412	77.27%	0.7750
15	100.00%	0.5603	90.91%	0.5550
22	73.33%	0.4871	90.91%	0.4540
29	100.00%	0.4345	90.91%	0.3948
30	100.00%	0.4037	90.91%	0.3864

Table 3: Performance Comparison of Baseline and FeatConvNet Model

Model	Mini-batch Acc.	Mini-batch Loss	Valid. Acc.	Valid. Loss
baseline(none)	80.00%	0.4393	86.36%	0.4385
FeatConvNet(none)	100.00%	0.4037	90.91%	0.3864
baseline(zerocenter)	86.67%	0.2156	95.45%	0.1888
FeatConvNet(zerocenter)	100.00%	0.0844	95.45%	0.1634
baseline(zscore)	86.67%	0.2153	95.45%	0.1928
FeatConvNet(zscore)	100.00%	0.0712	95.45%	0.1627
baseline(symmetric)	86.67%	0.2501	90.91%	0.2317
FeatConvNet(symmetric)	100.00%	0.0866	95.45%	0.1845
baseline(zero-one)	86.67%	0.2153	95.45%	0.1928
FeatConvNet(zero-one)	100.00%	0.0712	95.45%	0.1627

Feature convolutional network (FCN) model has one layer extra than the baseline model. We add a fully connected layer with 10 hidden units to the baseline which has the same output size with feature convolution layer in FCN for fair comparison. Compared with the original baseline, the validation accuracy of the new baseline model drops from 86.36% to 31.82%; the validation loss increases from 0.4385 to 1.1154. The results indicate that simply adding more parameters with fully connected layer may not improve the baseline model performance.

Tab. 2 illustrates training data of the feature convolution network model. The table shows the training process of feature convolutional network model without feature normalization. When the training is finished, the mini-batch accuracy hits 100.00%, with the loss drops to 0.4037 ; and the validation accuracy hits 90.91%, with the corresponding loss drops to 0.3864 . Compared with Tab. 1, the feature convolutional network model obviously gives a better performance.

We also test the baseline model and the feature convolutional network model with various normalization methods. We apply zerocenter, zscore, symmetric and zero-one normalizations to the baseline model and the feature convolutional network. The performance comparison of two models is reported in Tab. 3. From Tab. 3, we can see that the feature convolutional network performs constantly better than the baseline model, whether with or without feature normalizations.

5. Discussion

Feature engineering is a representation problem of adjusting the representation of the data to improve the efficacy of the machine learning models. It uses domain knowledge and knowledge about the machine learning method. Feature engineering requires expert knowledge and is usually difficult, time consuming and expensive. In this paper, we consider auto generate features with feature convolutions. That is, we introduce convolution operator into feature spaces. New features can be automatically generated with feature convolution.

Feature convolution extends spatial convolution operator to numerical tabular features, which are used extensively in machine learning tasks. In fact, Numeric features come from a variety of sources: measurements from a sensor, geolocation of a person, the price of a book, customer counts, etc. Numeric feature engineering techniques are fundamental to machine learning models. With feature convolution network, we can facilitate the process of handcraft numeric feature engineering and utilize the convoluted numeric features in an end-to-end fashion.

Feature Convolution Networks (FCN) is indeed a feature transformation technique using feature convolutions, and different observations of the same feature share the model parameter. The feature convolution network is a supervised model. On the other hand, there are unsupervised feature engineering methods seek to modify the feature space in ways that expose the structure present in the data. For example, restricted Boltzmann machines (RBMs) Hinton and Salakhutdinov (2006) and autoencoders Hinton and Zemel (1994). Usually, they require very large amounts of training data. Moreover, these methods usually learn a compressed feature representation, while feature convolution network is trained to generate new features with parametric feature convolution.

6. Conclusion

Numeric features are fundamental to machine learning tasks. In this paper, we extend spatial convolution operator to numerical tabular features and propose feature convolutional network. Feature convolutional networks contain feature convolution layer to extract pairwise feature convolutions in the relational feature spaces. Compared with the baseline multilayer neural network model, the feature convolutional network gains better performance in our experiments. Through the experiments, we found feature convolutional networks can automatically generate efficient features through parametric feature learning. The demo code can be found at https://github.com/info-ruc/FeatConvNet.

Acknowledgments

We want to thank the anonymous reviewers for their constructive comments. The professional and useful suggestions have enabled us to improve our paper.

References

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, page 7–10, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Geoffrey E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Communications of The ACM*, 313(5786):504–507, 2006.

Geoffrey E. Hinton and Richard S. Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Proceedings of the International Conference on Neural Information Processing Systems*, NIPS'94, page 3–10, 1994.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of The ACM*, 60(6):84–90, 2017.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Matlab. Define custom deep learning layer with learnable parameters, 2021.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015. doi: 10.1109/CVPR.2015.7298594.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

UCI Web. Iris data set, 2021.